BIRZEIT UNIVERSITY

Computer Science Department

Instructor's Manual

COMP242

LABORATORY WORK BOOK

Data Structure and Algorithm

Prepared by:

Mr. Iyad Jaber

Approved By:

Computer Science Department

2015/2016

Table of Contents

Lab 1: Recursion

Exercises

Question 1

Write a Recursive Function to print number from Given input down to 0.

Solution Q1

```
public static void print (int n)
{
	if (n == 0)
		System.out.println(n);
	else
	{
		System.out.println(n);
		print (n-1);
	}
}
```

Question 2

Use a recursive function to write a program that performs the following tasks.

1. Find The Factorial Number for an input n.
2. Check If a Given Array Values Are Palindrome or not.

Solution Q2

```
public static int factorial (int n)
{
	if (n==0)
		return 1;
	else
		return (n*factorial (n-1));
}

public static boolean checkPalindrome(int a[], int n)
{
	if ((a.length == 0) || (a.length == 1))
		return true;
	if (a.length % 2 == 0)
		return false;
	else
		if (n*2 > a.length)
			return true;
		else
		{
			if (a[n] != a[a.length - n-1])
				return false;
			else
				return checkPalindrome(a,n+1);
		}
}
```

Question 3

(**Anagrams**) Here's a different kind of situation in which recursion provides a neat solution to a problem. Suppose you want to list all the anagrams of a specified word; that is, all possible letter combinations (whether they make a real English word or not) that can be made from the letters of the original word. We'll call this anagramming a word.

Anagramming cat, for example, would produce

- cat
- cta
- atc
- act
- tca
- tac

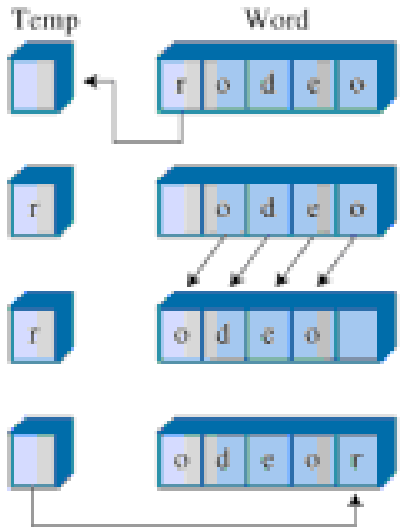
Try anagramming some words yourself. You'll find that the number of possibilities is the factorial of the number of letters. For 3 letters there are 6 possible words, for 4 letters there are 24 words, for 5 letters 120 words, and so on. (This assumes that all letters are distinct; if there are multiple instances of the same letter, there will be fewer possible words.)

How would you write a program to anagram a word? Here's one approach. Assume the word has n letters.

1. Anagram the rightmost n–1 letters.
2. Rotate all n letters.
3. Repeat these steps n times.

To rotate the word means to shift all the letters one position left, except for the leftmost letter, which "rotates" back to the right, as shown in Figure below.

Rotating the word n times gives each letter a chance to begin the word. While the selected letter occupies this first position, all the other letters are then anagrammed (arranged in every possible position). For cat, which has only 3 letters, rotating the remaining 2 letters simply switches them.



Solution Q3

```
// anagram.java
// creates anagrams
// to run this program: java AnagramApp
import java.io.*; // for I/O

class AnagramApp
{
	static int size;
	static int count;
	static char[] arrChar = new char[100];

	//--------------------------------------------------------------
	public static void main(String[] args) throws IOException
	{
		System.out.print("Enter a word: "); // get word
		System.out.flush();
		String input = getString();
		size = input.length(); // find its size
		count = 0;
		for(int j=0; j<size; j++) // put it in array
			arrChar[j] = input.charAt(j);
		doAnagram(size); // anagram it
	} // end main()

	//--------------------------------------------------------------
	public static void doAnagram(int newSize)
	{
		if(newSize == 1) // if too small,
			return; // go no further
		for(int j=0; j<newSize; j++) // for each position,
		{
			doAnagram(newSize-1); // anagram remaining
			if(newSize==2) // if innermost,
				displayWord(); // display it
			rotate(newSize); // rotate word
		}
	}
```

```
	//--------------------------------------------------------------
	// rotate left all chars from position to end
	public static void rotate(int newSize)
	{
		int j;
		int position = size - newSize;
		char temp = arrChar[position]; // save first letter
		for(j=position+1; j<size; j++) // shift others left
			arrChar[j-1] = arrChar[j];
		arrChar[j-1] = temp; // put first on right
	}

	//--------------------------------------------------------------
	public static void displayWord()
	{
		if(count < 99)
			System.out.print(" ");
		if(count < 9)
			System.out.print(" ");
		System.out.print(++count + " ");
		for(int j=0; j<size; j++)
			System.out.print( arrChar[j] );
		System.out.print(" ");
		System.out.flush();
		if(count%6 == 0)
			System.out.println("");
	}

	//--------------------------------------------------------------
	public static String getString() throws IOException
	{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);
		String s = br.readLine();
		return s;
	}
	//--------------------------------------------------------------
} // end class AnagramApp
```

Question 4

Write a recursive function to reverse a string. Write a recursive function to reverse the words in a string, i.e., "cat is running" becomes "running is cat".

Solution Q4

```
public class Lab1_Q4
{

   public static void main(String[] args)
   {
      System.out.println(wordReverse("DATA STRUCTURES AND ALGORITHMS"));

   }
   public static String wordReverse(String s)
   {
        int idx = s.indexOf(" ");
        if (idx < 0)
        {
            // no space char found, thus, s is just a single word, so
            // return just s itself
            return s;
        }
        else
        {
            // return at first the recursively reversed rest, followed by a
            // space char and the first extracted word
             return wordReverse(s.substring(idx + 1)) + " " +
                                s.substring(0, idx);
        }
    }

}
```

Question 5

Write a recursive Java method that counts the number of occurrences of the character 'a' in a string. Hint: a method signature that works is public static int countA (String s).

You can test your method in Eclipse. Consider using the charAt or startsWith methods in String.

1. Prove that your method is correct.
2. Prove that your method terminates.

Solution Q5

```
public class Lab1_Q5
{
	public static void main(String[] args)
	{
		System.out.println(count("iyad fawzi jaber",'a'));

	}
	public static int count (String line, char c)
	{
		if (line.length() != 0)
		{
			if (line.charAt(0) == c)
				return 1+count(line.substring(1),c);
			else
				return count(line.substring(1),c);
		}
		else
			return 0;
	}

}
```

Lab 2: Singly Linked List Implementation of the List ADT

Exercises

Question 1

Write a program to create a **Linked List** and different functionality related to the following operations:

1. Traversing Linked List.
2. Searching in Linked List.
3. Insertion in Linked List.
4. Deletion from Linked List.

Solution Q1

```
public void traversList()
{
	System.out.print("List (first-->last): ");
	Node current = first; // start at beginning of list
	while(current != null) // until end of list,
	{
		current.displayNode(); // print data
		current = current.next; // move to next link
	}
	System.out.println("");
}



public Node searchList(Object x)
{

	Node current = first; // start at beginning of list
	while((current != null) && (!current.element.equals(x)))
		current = current.next; // move to next Node
	return current;
}
```

```
public Node insert(Node p, Object x)
{

        Node current = new Node(x);
        current.next = p.next;
        p.next = current;
}



public boolean delete(Object x)
{
        If (first == null)
                return false;
        If (first.element.equals(x))
        {
                first = first.next;
                return true;
        }
        Node previous = first;
        Node current = first.next;
        while (current != null)
        {
                If (current.element.equals(x))
                {
                        previous.next = current.next;
                        return true;
                }
                previous = current;
                current = current.next;
        }
        return false;
}
```

Question 2

Write a program which creates two ordered list, based on an integer key, and then merges them together.

Solution Q2

```
public Node insertOrder(int x)
{
        Node temp = new Node(x);
        If (x <  first.element)
        {
                temp.next = first;
                first = temp;
        }
        else
        {
                Node previous = first;
                Node current = first.next;
                while ((current != null ) && (current.element < x))
                {
                        previuos = current;
                        current = current.next;
                }
                temp.next = current;
                previous.next = temp;
        }
}
```

```
Public List merge(List second)
{
        List result;
        Node current1 = first;
        Node current2 = second.first;
        while ((current1 != null) && (current2 != null))
        {
                if (current1.element <= current2.element)
                {
                        result.addLast(current1.element);
                        current1 = current1.next;
```

```
                }
                else
                {
                        result.addLast(current2.element);
                        current2 = current2.next;
                }
        }
        While (current1 != null)
        {
                        result.addLast(current1.element);
                        current1 = current1.next;
        }
        While (current2 != null)
        {
                        result.addLast(current2.element);
                        current2 = current2.next;
        }
        return result;
}
```

Question 3

Imagine an effective dynamic structure for storing polynomials. Write operations for addition, subtraction, and multiplication of polynomials.

Solution Q3

```
//////////////////////////////////////
// Addition
public static LinkedPoly add(LinkedPoly list1,LinkedPoly list2)
{
    LinkedPoly addList=new LinkedPoly();
    Node temp1=list1.head;
    Node temp3=temp1;
    Node temp2=list2.head;
    Node temp4=temp2;
    while(temp1.next!=null)
    {
      while(temp2.next!=null)
      {
        if(temp1.exp==temp2.exp)
        {
           addList.createList((temp1.coef+temp2.coef),temp1.exp);
           exponent+=temp1.exp;
        }
        temp2=temp2.next;
      }
      temp1=temp1.next;
      temp2=temp4;
      addList.print();
    }

    String[] array=exponent.split("");

    while(temp3!=null)
    {
        boolean exponentPresent = false;
        for(int i=1;i<array.length;i++)
        {
            if(temp3.exp==Integer.parseInt(array[i]))
                exponentPresent = true;
        }
        if (!exponentPresent)
            addList.createList(temp3.coef,temp3.exp);

        temp3=temp3.next;
     }
```

```
      while(temp4!=null)
      {
         boolean exponentPresent = false;
         for(int i=1;i<array.length;i++)
         {
            if(temp4.exp==Integer.parseInt(array[i]))
                 exponentPresent = true;
         }
         if (!exponentPresent)
             addList.createList(temp4.coef,temp4.exp);

         temp4=temp4.next;
      }
      return addList;
}

public void print()
{
    current = head;
    System.out.print(current.coef + "x^" + current.exp);
    while (current.next != null)
    {
        current = current.next;
        System.out.print(" + " + current.coef + "x^" + current.exp);
    }
    System.out.println();
}
//////////////////////////////////////////////////////////////////////

// Subtraction
public static LinkedPoly sub(LinkedPoly list1,LinkedPoly list2)
{
    LinkedPoly addList=new LinkedPoly();
    Node temp1=list1.head;
    Node temp3=temp1;
    Node temp2=list2.head;
    Node temp4=temp2;
    while(temp1.next!=null)
    {
      while(temp2.next!=null)
      {
         if(temp1.exp==temp2.exp)
         {
            addList.createList((temp1.coef-temp2.coef),temp1.exp);
            exponent+=temp1.exp;
         }
         temp2=temp2.next;
       }
       temp1=temp1.next;
       temp2=temp4;
       addList.print();
    }

    String[] array=exponent.split("");
```

```
    while(temp3!=null)
    {
        boolean exponentPresent = false;
        for(int i=1;i<array.length;i++)
        {
            if(temp3.exp==Integer.parseInt(array[i]))
                exponentPresent = true;
        }
        if (!exponentPresent)
            addList.createList(temp3.coef,temp3.exp);

        temp3=temp3.next;
     }


      while(temp4!=null)
      {
        boolean exponentPresent = false;
        for(int i=1;i<array.length;i++)
        {
           if(temp4.exp==Integer.parseInt(array[i]))
                exponentPresent = true;
        }
        if (!exponentPresent)
            addList.createList(-1*temp4.coef,temp4.exp);

        temp4=temp4.next;
      }
      return addList;
}

public void print()
{
    current = head;
    System.out.print(current.coef + "x^" + current.exp);
    while (current.next != null)
    {
        current = current.next;
        System.out.print(" + " + current.coef + "x^" + current.exp);
    }
    System.out.println();
}
//////////////////////////////////////////////////////////////////////
```

// Multiplication

```
public LinkedPoly multiply(LinkedPoly p)
{
      Node temp1 = head;
      Node temp2 = p.head;
      Node front = null;
      Node last = null;

      while(temp1!=null)
      {
          if(temp2==null)
             temp2 = p.head;
          while(temp2!=null)
          {
                 Node ptr = new
                 Node((temp1.term.coeff*temp2.term.coeff),
                       (temp1.term.degree+temp2.term.degree), null);

                 if(last!=null)
                    last.next = ptr;
                 else
                    front = ptr;

                 last = ptr;
                 temp2=temp2.next;
           }
           temp1 = temp1.next;

      }
      LinkedPoly productPoly = new LinkedPoly ();
      productPoly.head = front;
      return productPoly;
}
```

Question 4

Write Java programs to implement the List ADT using arrays

List ADT

The elements in a list are of generic type Object. The elements form a linear structure in which list elements follow one after the other, from the beginning of the list to its end. The list ADT supports the following operations:

createList(int n): Creates (initially) a list with n nodes.

Input: integer; Output: None

insertFirst(obj): Inserts object obj at the beginning of a list.

Input: Object; Output: None

insertAfter(obj, obj p): Inserts object obj after the obj p in a list.

Input: Object and position; Output: None

obj deleteFirst(): Deletes the object at the beginning of a list.

Input: None; Output: Deleted object obj.

obj deleteAfter(obj p): Deletes the object after the obj p in a list.

Input: Position; Output: Deleted object obj.

boolean isEmpty(): Returns a boolean indicating if the list is empty.

Input: None; Output: boolean (true or false).

int size(): Returns the number of items in the list.

Input: None; Output: integer.

Type Object may be any type that can be stored in the list. The actual type of the object will be provided by the user.

Solution Q4

```
package listpackage;


class ANode
{
        private Object data;
        private int next;

        public ANode(Objext d, int l)
        {
                data =  d;
                next = l;
        }
```

```
        public Object getData()
        {
                return data;
        }

        public int getLink()
        {
                return next;
        }
}

public class ArrayLL
{
        private int maxCapacity = 100;
        private ANode[] list;
        private int size;


        public ArrayLL()
        {
                list = new ANode[maxCapacity];
                list[list.length-1] = new ANode(null, -1);
                for(int i = 0; i < list.length-1; i++)
                        list[i] = new ANode(null, i+1);

                size = 0;
        }

        public createList (int n)
        {
                maxCapacity = n;
                list = new ANode[MAX_CAP];
                list[list.length-1] = new ANode(null, -1);
                for(int i = 0; i < list.length-1; i++)
                        list[i] = new ANode(null, i+1);

                size = 0;
        }


        public void inseertFirst(Object s)
        {
                if(size == 0)
                {
                        ANode a = new ANode(s, -1);
                        list[0] = a;
                        size++;
                }
                else
                if ( size < maxCapacity)
                {
                        ANode b = new ANode(s,1);
                        for(int j = size+1; j > 0; j--)
                                list[j] = list[j-1];
                        size++;
                }
                else
                        System.out.println("Out of memory");
        }
```

```
        public void insertAfter (Object s, Object p)
        {
                for(int i = 0; i< size; i++)
                {
                        if(list[i].getData().equals(p))
                        {
                                ANode b = new ANode(s,i+1);
                                for(int j = size; j > i; j--)
                                        list[j+1] = list[j];
                                list[i] = b;
                                size++;
                                break;
                        }
                }
        }

        public object deleteFirst()
        {
                if(size == 0)
                        return null;

                else
                {
                        ANode b = list[0];
                        for(int j = 0; j > size; j++)
                                list[j] = list[j+1];
                        size--;
                        return b;
                }
        }
        public object deleteAfter (Object p)
        {
                for(int i = 0; i< size; i++)
                {
                        if(list[i].getData().equals(p))
                        {
                                Anode temp = list[i+1];
                                for(int j = i+1; j < size; j++)
                                        list[j] = list[j+1];
                                size--;
                                return temp;
                        }
                }
                return null;
        }

        Public Boolean isEmpty ()
        {
                return (size ==0);
        }

        Public int size ()
        {
                return (size);
        }
}
```

Lab 3: Doubly Linked List Implementation of the List ADT

Exercises

Question 1

Write a program to create a **doubly linked list** and different functionality related to the following operations:

- Insert after a specific object.
- Delete a specific object.
- Insert element in the front of doubly linked list.
- Insert element at last in a doubly linked list.
- Delete first element in a doubly linked list
- Display elements in a double linked list

Solution Q1

```
/*
*  Java Program to Implement Doubly Linked List
*/
import java.util.Scanner;

/*  Class Node  */
class Node
{
        protected int data;
        protected Node next, prev;

        /* Constructor */
        public Node()
        {
            next = null;
            prev = null;
            data = 0;
        }

        /* Constructor */
        public Node(int d, Node n, Node p)
        {
            data = d;
            next = n;
            prev = p;
        }

        /* Function to set link to next node */
        public void setLinkNext(Node n)
        {
            next = n;
        }

        /* Function to set link to previous node */
        public void setLinkPrev(Node p)
```

```
        {
            prev = p;
        }

        /* Funtion to get link to next node */
        public Node getLinkNext()
        {
            return next;
        }

        /* Function to get link to previous node */
        public Node getLinkPrev()
        {
            return prev;
        }

        /* Function to set data to node */
        public void setData(int d)
        {
            data = d;
        }

        /* Function to get data from node */
        public int getData()
        {
            return data;
        }
}

/* Class linkedList */
class linkedList
{
        protected Node start;
        protected Node end ;
        public int size;

        /* Constructor */
        public linkedList()
        {
            start = null;
            end = null;
            size = 0;
        }

        /* Function to check if list is empty */
        public boolean isEmpty()
        {
            return start == null;
        }

        /* Function to get size of list */
        public int getSize()
        {
            return size;
        }

        /* Function to insert element at begining */
        public void insertAtStart(int val)
        {
            Node nptr = new Node(val, null, null);
            if(start == null)
```

```
        {
            start = nptr;
            end = start;
        }
        else
        {
            start.setLinkPrev(nptr);
            nptr.setLinkNext(start);
            start = nptr;
        }
        size++;
    }

    /* Function to insert element at end */
    public void insertAtEnd(int val)
    {
        Node nptr = new Node(val, null, null);
        if(start == null)
        {
            start = nptr;
            end = start;
        }
        else
        {
            nptr.setLinkPrev(end);
            end.setLinkNext(nptr);
            end = nptr;
        }
        size++;
    }

    /* Function to insert element at position */

    public void insertAtPos(int val , int pos)
    {
        Node nptr = new Node(val, null, null);
        if (pos == 1)
        {
            insertAtStart(val);
            return;
        }
        Node ptr = start;
        for (int i = 2; i <= size; i++)
        {
            if (i == pos)
            {
                Node tmp = ptr.getLinkNext();
                ptr.setLinkNext(nptr);
                nptr.setLinkPrev(ptr);
                nptr.setLinkNext(tmp);
                tmp.setLinkPrev(nptr);
            }
            ptr = ptr.getLinkNext();
        }
        size++ ;
    }

    /* Function to delete node at begining */
    public void deleteFirst()
    {
        if (size == 1)
```

```
        {
            start = null;
            end = null;
            size = 0;
            return;
         }
         start = start.getLinkNext();
         start.setLinkPrev(null);
         size--;
         return ;
    }

    /* Function to delete node at position */
    public void deleteAtPos(int pos)
    {
        if (pos == 1)
        {
            if (size == 1)
            {
                start = null;
                end = null;
                size = 0;
                return;
            }
            start = start.getLinkNext();
            start.setLinkPrev(null);
            size--;
            return ;
        }
        if (pos == size)
        {
            end = end.getLinkPrev();
            end.setLinkNext(null);
            size-- ;
        }
        Node ptr = start.getLinkNext();
        for (int i = 2; i <= size; i++)
        {
            if (i == pos)
            {
                Node p = ptr.getLinkPrev();
                Node n = ptr.getLinkNext();
                p.setLinkNext(n);
                n.setLinkPrev(p);
                size-- ;
                return;
            }
            ptr = ptr.getLinkNext();
        }
    }

    /* Function to display status of list */
    public void display()
    {
        System.out.print("\nDoubly Linked List = ");
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLinkNext() == null)
```

```
            {
                System.out.println(start.getData() );
                return;
            }
            Node ptr = start;
            System.out.print(start.getData()+ " <-> ");
            ptr = start.getLinkNext();
            while (ptr.getLinkNext() != null)
            {
                System.out.print(ptr.getData()+ " <-> ");
                ptr = ptr.getLinkNext();
            }
            System.out.print(ptr.getData()+ "\n");
        }
}
```

Question 2

Read a paragraph containing words from an input file. Then create a doubly-linked list containing the distinct words read, where the words of the same length are placed in the same list, in ascending order.

Print the lists in descending order of their word length, case insensitive.

Example data:

Input:

Everything LaTeX numbers for you has a counter associated with it. The name of the counter is the same as the name of the environment or command that produces the number, except with no \. Below is a list of some of the counters used in LaTeX's standard document styles to control numbering.

Output:

1: a

2: \. as in is no of or to

3: for has it. the you

4: list name same some that used with

5: Below LaTeX

6: except styles

7: command control counter LaTeX's number, numbers

8: counters document produces standard

10: associated Everything numbering

11: environment

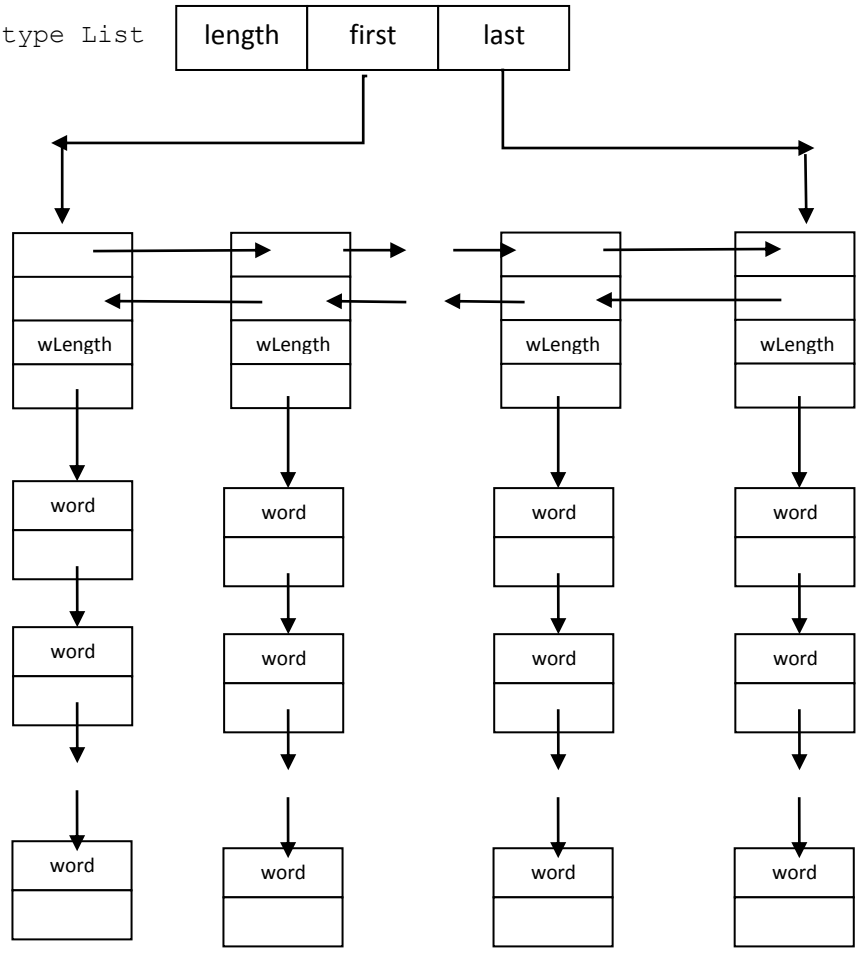
Solution Q2

```
public class Data
{
      String word;
      Data next;
}

Public class Node
{
      int  wLength;           //wordLength
      Node next, prev;
      Data words;
}
```

```
public class List
{
      Node first, last;
      int length;
}
```

Header is of type List



Lab 4: Cursor Implementation of List ADT

Exercises

Question 1

Write a program to create a cursor implementation of linked lists and different functionality related to the following operations:

1. Initialized CURSOR_SPACE.
2. Cursor-alloc and cursor-free.
3. Test whether a linked list is empty--cursor implementation.
4. Test whether p is last in a linked list--cursor implementation.
5. Find routine--cursor implementation.
6. Deletion routine for linked lists--cursor implementation.
7. Insertion routine for linked lists--cursor implementation.

Solution Q1

```
public class CursorNode
{
	Object element;
	int next;

	public CursorNode(Object x, int i)
	{
		element = x;
		next = i;
	}
}


public class CursorList
{
    static CursorNode[] cursorSpace;

    private static final int SPACE_SIZE = 100;

    static
    {
        cursorSpace = new CursorNode[SPACE_SIZE];
        for( int i = 0; i < SPACE_SIZE; i++ )
            cursorSpace[i] = new CursorNode( null, i + 1);
        cursorSpace[SPACE_SIZE - 1].next = 0;
    }

    static private  int cursorAlloc()
    {
      int p = cursorSpace[0].next;
      cursorSpace[0].next = cursorSpace[p].next;
      if( p == 0 )
           System.out.println("Out of memory");
      cursorSpace[p].next = 0;
```

```
    return p;
  }

  static private  void free(int p)
  {
      cursorSpace[p].element = null;
      cursorSpace[p].next = cursorSpace[0].next;
      cursorSpace[0].next = p;
  }

  static public int buildList()
  {
    return cursorAlloc();
  }

  static public void removeList(int header)
  {
    while( !isEmpty(header))
          remove(header, cursorSpace[header].next);
    free(header);
  }

  static public boolean isEmpty( int header )
  {
    return cursorSpace[header].next == 0;
  }

  static public void makeEmpty( int header  )
  {
    while( !isEmpty(header))
          remove(header, cursorSpace[header].next);
  }

  static public void insert(int header, Object x, int p )
  {
    int current = cursorSpace[header].next;
    int previous = header;
    int i= 0;
    while ((current != 0) && (i< p))
    {
          previous = current;
          current = cursorSpace[current].next;
    }
      int tmp = cursorAlloc();

      cursorSpace[tmp].element = x;
      cursorSpace[tmp].next = cursorSpace[previous].next;
      cursorSpace[previous].next = tmp;
  }

  static public int find(int header, Object x )
  {
    int current = cursorSpace[header].next;

    while(current != 0 && !cursorSpace[current].element.equals(x))
                current = cursorSpace[current].next;

    return ( current );
  }
```

```
    static public int findPrevious(int header,Object x )
    {
      int current = header;

      while( cursorSpace[current].next != 0 &&
            !cursorSpace[cursorSpace[current].next].element.equals(x))
                  current = cursorSpace[current].next;

      return ( current );
    }

    static public void remove(int header, Object x )
    {
      int pos = findPrevious(header, x );

      if( cursorSpace[pos].next != 0 )
      {
          int tmp = cursorSpace[pos].next;
          cursorSpace[pos].next = cursorSpace[tmp].next;
          free( tmp );
      }
    }

    static public void printList(int header)
    {
      if( isEmpty(header ) )
          System.out.print( "Empty list" );
      else
      {
          int current = cursorSpace[header].next;
          while (current != 0)
          {
             System.out.print( cursorSpace[current].element + " " );
             current = cursorSpace[current].next;
          }
      }
      System.out.println( );
    }
}
```

Question 2

Given a linked list and two keys in it, swap nodes for two given keys. Nodes should be swapped by changing links. Swapping data of nodes may be expensive in many situations when data contains many fields. It may be assumed that all keys in linked list are distinct.

Example:

Input: 10 -> 15 -> 12 -> 13 -> 20 -> 14, x = 12, y = 20

Output: 10 -> 15 -> 20 -> 13 -> 12 -> 14

This may look a simple problem, but is interesting question as it has following cases to be handled.

1. X and y may or may not be adjacent.
2. Either x or y may be a head node.
3. Either x or y may be last node.
4. X and/or y may not be present in linked list.

How to write a clean working code that handled all of the above possibilities.

Solution Q2

```
static public void swapNodes(int header, Object x, Object y )
{
      int pos1 = findPrevious(header, x );
      int pos2 = findPrevious(header, y );
      if ((pos1 != 0) && (pos2 != 0))
      {
            int temp1 = cursorSpace[pos1].next;
            int temp2 = cursorSpace[pos2].next;
            int k = cursorSpace[temp2].next;
            cursorSpace[pos1].next = temp2;
            cursorSpace[pos2].next = temp1;
            cursorSpace[temp2].next = cursorSpace[temp1].next;
            cursorSpace[temp1].next = k;
      }
}
```

Lab 5: Stack ADT

Exercises

Question 1

We commonly write arithmetic expressions in infix form, that is, with each operator placed between its operands, as in the following expression:

(3+4)*(5/2)

Although we are comfortable writing expressions in this form, infix form has the disadvantage that parentheses must be used to indicate the order in which operators are to be evaluated.

These parentheses, in turn, greatly complicate the evaluation process.

Evaluation is much easier if we can simply evaluate operators from left to right. Unfortunately, this left-to-right evaluation strategy will not work with the infix form of arithmetic expressions. However, it will work if the expression is in postfix form. In the postfix form of an arithmetic expression, each operator is placed immediately after its operands. The expression above is written below in postfix form as

34+52/*

Note that both forms place the numbers in the same order (reading from left to right). The order of the operators is different, however, because the operators in the postfix form are positioned in the order that they are evaluated. The resulting postfix expression is hard to read at first, but it is easy to evaluate. All you need is a stack on which to place intermediate results.

Suppose you have an arithmetic expression in postfix form that consists of a sequence of single digit, nonnegative integers and the four basic arithmetic operators (addition, subtraction, multiplication, and division). This expression can be evaluated using the following algorithm in conjunction with a stack of floating-point numbers.

Read in the expression character by character. As each character is read in,

- If the character corresponds to a single digit number (characters '0' to '9'), then push the corresponding floating-point number onto the stack.
- If the character corresponds to one of the arithmetic operators (characters '+', '-', '*', and '/'), then

- Pop a number off of the stack. Call it operand1.
- Pop a number off of the stack. Call it operand2.
- Combine these operands using the arithmetic operator, as follows

  result = operand2 operator operand1

- Push result onto the stack.

- When the end of the expression is reached, pop the remaining number off the stack. This number is the value of the expression.

Applying this algorithm to the arithmetic expression

34+52/*

Yields the following computation:

'3': Push 3.0

'4': Push 4.0

'+': Pop, operand1 = 4.0

Pop, operand2 = 3.0

Combine, result = 3.0 + 4.0 = 7.0

Push 7.0

'5': Push 5.0

'2': Push 2.0

'/': Pop, operand1 = 2.0

Pop, operand2 = 5.0

Combine, result = 5.0 / 2.0 = 2.5

Push 2.5

'*': Pop, operand1 = 2.5

Pop, operand2 = 7.0

Combine, result = 7.0 * 2.5 = 17.5

Push 17.5

'\n': Pop, Value of expression = 17.5

Step 1: Create a program (call it PostFix.java) that reads the postfix form of an arithmetic expression, evaluates it, and outputs the result. Assume that the expression consists of

single digit, nonnegative integers ('0' to '9') and the four basic arithmetic operators ('+', '-', '*', and '/').

Further assume that the arithmetic expression is input from the keyboard with all the characters on one line. In PostFix.java, values of type float will be large enough for our purposes.

Hints:

1. Review the code for TestAStack.java to recall how to read in characters, deal with whitespace and push an Object onto the stack.
2. To convert from Object to Float and then to the primitive float requires a cast and the Float.floatValue( ) method. For example,

```
float operand1;
operand1 = ((Float)resultStack.pop( )).floatValue( );
```

3. To set precision to two decimal places, import java.text.DecimalFormat and use code similar to the following:

```
float outResult;
DecimalFormat fmt = new DecimalFormat("0.##");
System.out.println(fmt.format(outResult));
```

Step 2: Complete the following test plan by filling in the expected result for each arithmetic expression. You may wish to include additional arithmetic expressions in this test plan.

Step 3: Execute the test plan. If you discover mistakes in your program, correct them and execute the test plan again.

Test Plan for the Postfix Arithmetic Expression Evaluation Program

Test case	Arithmetic expression	Expected result	Checked
One operator	34+		
Nested operators	34+52/*		
Uneven nesting	93*2+1-		
All operators at end	4675-+*		
Zero dividend	02/		
Single-digit number	7		

Solution Q1

```
import java.io.*; // for I/O

class Link
{
	public int Data; // data item
	public Link next; // next link in list

	public Link(int dd) // constructor
	{
		Data = dd;
	}

	public void displayLink() // display ourself
	{
		System.out.print(Data + " ");
	}
} // end class Link

class LinkList
{
	private Link first; // ref to first item on list

	public LinkList() // constructor
	{
		first = null;
	} // no items on list yet

	public boolean isEmpty() // true if list is empty
	{
		return (first==null);
	}

	public void insertFirst(int dd) // insert at start of list
	{ // make new link
		Link newLink = new Link(dd);
		newLink.next = first; // newLink --> old first
		first = newLink; // first --> newLink
	}

	public int deleteFirst() // delete first item
	{ // (assumes list not empty)
		if (first != null)
		{
			Link temp = first; // save reference to link
			first = first.next; // delete it: first-->old next
			return temp.Data; // return deleted link
		}
		return 0;
	}

	public void displayList()
	{
		Link current = first; // start at beginning of list
		while(current != null) // until end of list,
		{
			current.displayLink(); // print data
			current = current.next; // move to next link
		}
```

```
            System.out.println("");
      }

} // end class LinkList

////////////////////////////////

class LinkStack
{
      private LinkList theList;

      public LinkStack() // constructor
      {
            theList = new LinkList();
      }

      public void push(int j) // put item on top of stack
      {
            theList.insertFirst(j);
      }

      public int pop() // take item from top of stack
      {
            return theList.deleteFirst();
      }

      public boolean isEmpty() // true if stack is empty
      {
            return ( theList.isEmpty() );
      }

      public void displayStack(String s)
      {
            System.out.print(s);
            System.out.print("Stack (top-->bottom): ");
            theList.displayList();
      }

} // end class LinkStack

////////////////////////////////


class ParsePost
{
      private LinkStack theStack;
      private String input;

      public ParsePost(String s)
      {
            input = s;
      }

      public int doParse()
      {
            theStack = new LinkStack(); // make new stack
            char ch;
            int j;
            int num1, num2, interAns;
            for(j=0; j<input.length(); j++) // for each char,
            {
                  ch = input.charAt(j); // read from input
```

```
                    theStack.displayStack(""+ch+" "); // *diagnostic*
                    if(ch >= '0' && ch <= '9') // if it's a number
                          theStack.push( (int)(ch-'0') ); // push it
                    else // it's an operator
                    {
                          num2 = theStack.pop(); // pop operands
                          num1 = theStack.pop();
                          switch(ch) // do arithmetic
                          {
                          case '+':
                                interAns = num1 + num2;
                                break;
                          case '-':
                                interAns = num1 - num2;
                                break;
                          case '*':
                                interAns = num1 * num2;
                                break;
                          case '/':
                                if (num2 != 0)
                                      interAns = num1 / num2;
                                else
                                {
                                      System.out.println("*** Error*** Divide
                          by zero");
                                      interAns = 0;
                                }
                                break;
                          default:
                                interAns = 0;
                          } // end switch
                          theStack.push(interAns); // push result
                    } // end else
              } // end for
              interAns = theStack.pop(); // get answer
              return interAns;
        } // end doParse()
} // end class ParsePost

////////////////////////////////////////

import java.io.*; // for I/O

class Lab5_Q1
{
      public static void main(String[] args) throws IOException
      {
            String input;
            int output;
            while(true)
            {
                  System.out.print("Enter postfix: ");
                  System.out.flush();
                  input = getString(); // read a string from kbd
                  if( input.equals("") ) // quit if [Enter]
                        break;
                  //make a parser
                  ParsePost aParser = new ParsePost(input);
                  output = aParser.doParse(); // do the evaluation
                  System.out.println("Evaluates to " + output);
            } // end while
      } // end main()
```

```
	public static String getString() throws IOException
	{
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);
		String s = br.readLine();
		return s;
	}

} // end class Lab5_Q1
```

Question 2

Checking 'Balanced Parentheses' using Array implementation of stack

Aim: To check whether an expression has balanced parentheses using array implementation of a stack.

Theory:

Compilers check your programs for syntax errors, but frequently a lack of one symbol will cause the compiler to spill out a hundred lines of diagnostics without identifying the real error. Thus, every right brace, bracket and parentheses must correspond to its left counterpart.

This can be verified using a stack.

Algorithm:

1. Make an empty stack implemented as an array.
2. Scan the expression from left to right, character by character.
3. During your scanning:
    a. If you find a left parentheses push it into the stack.
    b. If you find a right parentheses examine the status of the stack:
        I. If the stack is empty then the right parentheses does not have a matching left parentheses. So stop scanning and print expression is invalid.
        II. If the stack is not empty, pop the stack and continue scanning.
4. When the end of the expression is reached, the stack must be empty. Otherwise one or more left parentheses has been opened and not closed.

Solution Q2

```
import java.io.*; // for I/O

class Stack
{
	private int maxSize;
	private char[] stackArray;
	private int top;

	public Stack(int size) // constructor
	{
		maxSize = size;
		stackArray = new char[maxSize];
		top = -1;
	}

	public void push(char j) // put item on top of stack
	{
		stackArray[++top] = j;
	}

	public char pop() // take item from top of stack
	{
		return stackArray[top--];
	}

	public char peek() // peek at top of stack
	{
		return stackArray[top];
	}

	public boolean isEmpty() // true if stack is empty
	{
		return (top == -1);
	}

	public boolean isFull() // true if stack is full
	{
		return (top == maxSize-1);
	}

	public int size() // return size
	{
		return top+1;
	}

	public char peekN(int n) // peek at index n
	{
		return stackArray[n];
	}

	public void displayStack(String s)
	{
		System.out.print(s);
		System.out.print("Stack (bottom-->top): ");
		for(int j=0; j<size(); j++)
		{
			System.out.print( peekN(j) );
			System.out.print(' ');
		}
```

```
            System.out.println("");
      }
}

//////////////////////////////////////

import java.io.*; // for I/O

class Lab5_Q2
{
      public static void main(String[] args) throws IOException
      {
            String input;
            char ch;
            while(true)
            {
                  Stack stack = new Stack(20);
                  System.out.print("Enter expression: ");
                  System.out.flush();
                  input = getString(); // read a string from kbd
                  if( input.equals("") ) // quit if [Enter]
                        break;
                  //make a parser
                  for (int i= 0; i< input.length();i++)
                  {
                        if (input.charAt(i) == '('
                              stack.push('(');
                        else
                              if (input.charAt(i) == ')'
                              {
                                    ch = stack.pop();
                                    if (ch != '('
                                    {
                                          System.out.println("Error");
                                          break;
                                    }
                              }
                  }
                  if (! stack.isEmpty())
                        System.out.println("Invalid Expression");
                  else
                        System.out.println("Valid Expression");
            } // end while
      } // end main()

      public static String getString() throws IOException
      {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            String s = br.readLine();
            return s;
      }

} // end class Lab5_Q2
```

Lab 6: Queue ADT

Exercises

Question 1

In this exercise, you use a queue to simulate the flow of customers through a checkout line in a store. In order to create this simulation, you must model both the passage of time and the flow of customers through the line. You can model time using a loop in which each pass corresponds to a set time interval-one minute, for example. You can model the flow of customers using a queue in which each element corresponds to a customer in the line.

In order to complete the simulation, you need to know the rate at which customers join the line, as well as the rate at which they are served and leave the line. Suppose the checkout line has the following properties.

- One customer is served and leaves the line every minute (assuming there is at least one customer waiting to be served during that minute).
- Between zero and two customers join the line every minute, where there is a 50% chance that no customers arrive, a 25% chance that one customer arrives, and a 25% chance that two customers arrive.

You can simulate the flow of customers through the line during a time period n minutes long

Using the following algorithm.

Initialize the queue to empty.
for (minute = 0 ; minute < n ; minute++)
{

    If the queue is not empty, then remove the customer at the front of the queue.
    Compute a random number k between 0 and 3.
    If k is 1, then add one customer to the line. If k is 2, then add two customers to the line. Otherwise (if k is 0 or 3), do not add any customers to the line.

}

Step 1: Create a program that uses the Queue ADT to implement the model described above. Your program should update the following information during each simulated minute, that is, during each pass through the loop:

- The total number of customers served
- The combined length of time these customers spent waiting in line
- The maximum length of time any of these customers spent waiting in line

In order to compute how long a customer waited to be served, you need to store the "minute" that the customer was added to the queue as part of the queue element corresponding to that customer.

Step 2: Use your program to simulate the flow of customers through the line and complete the following table. Note that the average wait is the combined waiting time divided by the total number of customers served.

Take special note that, in this program shell, the length of the simulation, simLength, is read in as an argument to the program itself. In other words, it is a value assigned to the args parameter of the main method-now you know the use of the args array in the main method of your program. This argument is entered as an additional string when you run your program from the command line prompt. If your Java development system does not allow you to invoke your program from the command line, you will need to determine how this argument is passed in the particular Java development system you are using.

Since each args in main is a String array, args[0] must be converted to an int, which is the data type assigned to the simLength. As illustrated in the statement, this conversion is commonly done in Java by using the static method parseInt in the class Integer.

```
simLength = Integer.parseInt( args[0] );
```

Java programmers use this parseInt method on a daily basis as a quick and easy way to convert a String to an int in Java.

Time in minutes	Total number of customers served	Average wait	Longest wait
30			
60			
120			
480			

Solution Q1

```
import java.io.*; // for I/O

class Link
{
	public int data; // data item
	public Link next; // next link in list

	public Link(int dd) // constructor
	{
		data = dd;
	}

	public void displayLink() // display ourself
	{
		System.out.print(Data + " ");
	}
} // end class Link

class LinkList
{
	private Link first; // ref to first item on list

	public LinkList() // constructor
	{
		first = null;
	} // no items on list yet

	public boolean isEmpty() // true if list is empty
	{
		return (first==null);
	}

	public void insertFirst(int dd) // insert at start of list
	{ // make new link
		Link newLink = new Link(dd);
		newLink.next = first; // newLink --> old first
		first = newLink; // first --> newLink
	}

	public int deleteLast() // delete last item
	{
		int temp;
		Link current = first;
		if (first == null)
			return 0;
		if (first.next == null)
		{
			temp = first.data;
			first = null;
			return (temp);
		}
		while (current.next.next != null)
			current = current.next;
		temp = current.next.data;
		current.next = null;
		return temp;
	}
```

```
	public void displayList()
	{
		Link current = first; // start at beginning of list
		while(current != null) // until end of list,
		{
			current.displayLink(); // print data
			current = current.next; // move to next link
		}
		System.out.println("");
	}

} // end class LinkList

////////////////////////////////
class Queue
{
	private LinkList theList;

	public Queue() // constructor
	{
		theList = new LinkList();
	}

	public void enqueue(int j)
	{
		theList.insertFirst(j);
	}

	public int dequeue() // take item from top of stack
	{
		return theList.deleteLast();
	}

	public boolean isEmpty() // true if stack is empty
	{
		return ( theList.isEmpty() );
	}


} // end class LinkStack

////////////////////////////
```

```
import java.io.*;
import java.util.Scanner;
import java.util.Random;
import java.text.DecimalFormat;

class StoreSim
{   public static void main (String args[]) throws IOException
      {
            Queue custQ = new Queue( ); // Line (queue) of customers
                                        // containing
            // the time that each customer arrived
            // and joined the line
            Random rand // Initialize random number generator
                              = new Random(System.currentTimeMillis());
                        // seed is system
            // clocktime to limit
            // sequence repetition

            int simLength, // Length of simulation (minutes)
                  minute, // Current minute
                  timeArrived, // Time dequeued customer arrived
                  waitTime, // How long dequeued customer waited
                  totalServed = 0, // Total customers served
                  totalWait = 0, // Total waiting time for all customers
                  maxWait = 0, // Longest wait
                  numArrivals = 0, // Number of new arrivals
                  customer,   // customer in Queue
                  j; // Loop counter


            // Value read in as one of main's args[]
            // which will simplify redirecting the output to a file
            //simLength = Integer.parseInt(args[0]);
            System.out.println("Please enter the Length of simulation
(minutes)\n");
            Scanner scan = new Scanner(System.in);
            simLength = scan.nextInt();

            System.out.println("\nNumber of minutes the Simulator will run:
"
                        + simLength);

            for ( minute = 0 ; minute < simLength ; minute++ )
            {
                  // Output time & number of customers waiting in line
                  if (!custQ.isEmpty())
                  {
                        customer = custQ.dequeue();
                        totalServed ++;
                        waitTime = minute- customer;
                        totalWait = totalWait + waitTime;
                        if (waitTime >maxWait)
                              maxWait = waitTime;
                  }
                  // Dequeue the first customer in line (if any). Increment
                  // totalServed, add the time that this customer waited to
                  // totalWait, and update maxWait if this customer waited
                  // longer than any previous customer.
                  // Determine the number of new customers.
                  // Uses a random number generator.
```

```
                    switch ( rand.nextInt(4) ) // Randomly generate a number
                    // from 0 to 3
                    {  case 1:  System.out.println("Add 1 customer to the
line");
                                custQ.enqueue(minute);
                                break;
                          case 2: System.out.println("Add 2 customers to the
line");
                                custQ.enqueue(minute);
                                custQ.enqueue(minute);
                                break;
                          default : System.out.println("Do not add any
                          customer to the line");
                    }

                    // Add the new customers to the line

                    //custQ.enqueue(minute);

              }

              // set precision to 2 decimal places
              DecimalFormat fmt = new DecimalFormat("0.##");

              System.out.println("\nCustomers served : " + totalServed);
              System.out.println("Average wait : " +
                    fmt.format((double)totalWait/totalServed));
              System.out.println("Longest wait : " + maxWait);

        } // main

} // class StoreSim
```

Question 2

Write a Java program to perform the implementation of queue using array

Algorithm

Step 1: start the program

Step 2: initialize the array variable a () for storing elements and declare the required variable

Step 3: Define a function to insert, delete display the data items for data

Step 4: For insert () function

Get the new elements

Create the rear is greater than array size display queue is "overflow".

Else

Insert the new element & increment the top pointer

Step 5: For delete () function

If the queue is empty than display "queue is over flow"

Else

Decrement the top pointer

Step 6: For display () function

Apply the loop for to display queue elements.

Solution Q2

```
import java.util.Arrays;

public class Queue<T>
{

    private int front;
    private int rear;
    int size;
    T[] queue;

    public Queue(int inSize)
    {
        size = inSize;
        queue = (T[]) new Object[size];
        front = -1;
        rear = -1;
    }

    public boolean isempty()
    {
        return (front == -1 && rear == -1);
    }

    public void enQueue(T value)
    {
        if ((rear+1)%size==front) {
            throw new IllegalStateException("Queue is full");

        } else if (isempty()) {
            front++;
            rear++;
            queue[rear] = value;

        } else {
            rear=(rear+1)%size;
            queue[rear] = value;

        }
    }

    public T deQueue()
    {
        T value = null;
        if (isempty())
        {
            throw new IllegalStateException("Queue is empty, cant
dequeue");
        }
        else
            if (front == rear)
            {
                  value = queue[front];
                  front = -1;
                  rear = -1;

            }
            else
            {
                  value = queue[front];
```

```
                front=(front+1)%size;

            }
        return value;

    }

    @Override
    public String toString()
    {
        return "Queue [front=" + front + ", rear=" + rear + ", size=" +
size
                + ", queue=" + Arrays.toString(queue) + "]";
    }

}
////////////////////////////////////

public class Lab6_Q2
{
          public static <T> void main(String[] args)
          {
              Queue newQueue = new Queue(5);
              newQueue.enQueue(10);
              newQueue.enQueue(20);
              newQueue.enQueue(30);
              newQueue.enQueue(40);
              newQueue.enQueue(50);
              System.out.println((T) newQueue.toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.toString());
              newQueue.enQueue(60);
              newQueue.enQueue(70);
              System.out.println((T) newQueue.toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.deQueue().toString());
              System.out.println((T) newQueue.toString());
          }
}
```

Lab 7: Binary Search Tree

EXERCISES

Question 1

Write a JAVA program to perform the following operations:

a) To construct a binary search tree of integers.
b) To traverse the tree using all the methods i.e., **inorder**, **preorder** and **postorder**.
c) To display the elements in the tree.

ALGORITHM FOR TREES

MAIN FUNCTION ()

Step 1: initialize a list called tree using tree data structures, consider root as the start node.

Step 2: Read the operation from the keyboard, if the operation says **preorder** display then go to the **preorder** display function else if the operation specifies post order then go to the post order display function else if the operation specifies in order then go to the in order display function else if the operation specified is exit then go to step step3.

Step 3: Return to the main program.

INORDER FUNCTION ()

Step 1: Traverse the left subtree by calling **Inorder** function recursively.

Step 2: Visit root and print root information.

Step 3: Traverse the right subtree by calling **Inorder** function recursively.

Step 4: Return to the main program.

PREORDER DISPLAY FUNCTON ()

Step 1: Visit root and print root information.

Step 2: Traverse the left subtree by calling **Preorder** function recursively.

Step 3: Traverse the right subtree by calling **Preorder** function recursively.

Step 4: Return to the main program

POST ORDER DISPLAY FUNCTION ()

Step 1: Traverse the left subtree by calling **Postorder** function recursively.

Step 2: Traverse the right subtree by calling **Postorder** function recursively.

Step 3: Visit root and print root information.

Step 4: Return to the main program.

INSERTION FUNCTION ()

Step 1: Create a node called *newnode* with a data field and a left and the right link.

Step 2: Insert the value entered from the user to the data field of the *newnode* and assign the right and left links to the null value.

Step3: Check if the tree is empty, if yes then *newnode* is the root node and go to step9 else go to step4.

Step 4: Consider a temporary node p and assign it with root.

Step 5: Repeat S6 and S7 till p becomes Null.

Step 6: Consider a temporary node q and assign it with p.

Step 7: Check if *newnode* data is greater than p node data, if yes proceed towards right subtree else proceed towards left subtree.

Step 8: If new node data is less than q node data insert *newnode* as left child of q node else insert *newnode* as right child of q node.

Step 9: Return to the main program.

Solution Q1

```
/*
*  Java Program to Implement Binary Search Tree
*/

/* Class BSTNode */
class BSTNode
{
      BSTNode left, right;
    int data;

    /* Constructor */
    public BSTNode()
    {
      left = null;
        right = null;
        data = 0;
    }

    /* Constructor */
    public BSTNode(int n)
    {
      left = null;
        right = null;
        data = n;
```

```
    }

    /* Function to set left node */
    public void setLeft(BSTNode n)
    {
        left = n;
    }

    /* Function to set right node */
    public void setRight(BSTNode n)
    {
        right = n;
    }

    /* Function to get left node */
    public BSTNode getLeft()
    {
      return left;
    }

    /* Function to get right node */
    public BSTNode getRight()
    {
        return right;
    }

    /* Function to set data to node */
    public void setData(int d)
    {
        data = d;
    }

    /* Function to get data from node */
    public int getData()
    {
        return data;
    }
}
/////////////////////////////////////////

/* Class BST */

class BST
{
      private BSTNode root;

    /* Constructor */
    public BST()
    {
      root = null;
    }

    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
      return root == null;
    }

    /* Functions to insert data */
    public void insert(int data)
    {
```

```
    root = insert(root, data);
  }

  /* Function to insert data recursively */
  private BSTNode insert(BSTNode node, int data)
  {
    if (node == null)
          node = new BSTNode(data);
      else
      {
          if (data <= node.getData())
                node.left = insert(node.left, data);
          else
              node.right = insert(node.right, data);
      }
          return node;
   }

   /* Functions to delete data */
   public void delete(int k)
   {
     if (isEmpty())
           System.out.println("Tree Empty");
       else
           if (search(k) == false)
                 System.out.println("Sorry "+ k +" is not present");
           else
           {
               root = delete(root, k);
               System.out.println(k+ " deleted from the tree");
           }
   }

   private BSTNode delete(BSTNode root, int k)
   {
     BSTNode p, p2, n;
       if (root.getData() == k)
       {
           BSTNode lt, rt;
           lt = root.getLeft();
           rt = root.getRight();
           if (lt == null && rt == null)
                 return null;
           else
                 if (lt == null)
               {
                   p = rt;
                   return p;
               }
               else
                 if (rt == null)
                 {
                       p = lt;
                       return p;
                 }
                 else
                 {
                       p2 = rt;
                       p = rt;
                       while (p.getLeft() != null)
                             p = p.getLeft();
```

```
                    p.setLeft(lt);
                    return p2;
                }
        }
        if (k < root.getData())
        {
            n = delete(root.getLeft(), k);
            root.setLeft(n);
        }
        else
        {
            n = delete(root.getRight(), k);
            root.setRight(n);
        }
        return root;
    }

    /* Functions to count number of nodes */
    public int countNodes()
    {
      return countNodes(root);
    }

    /* Function to count number of nodes recursively */
    private int countNodes(BSTNode r)
    {
      if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }

    /* Functions to search for an element */
    public boolean search(int val)
    {
      return search(root, val);
    }

    /* Function to search for an element recursively */
    private boolean search(BSTNode r, int val)
    {
      boolean found = false;
        while ((r != null) && !found)
        {
            int rval = r.getData();
            if (val < rval)
                  r = r.getLeft();
            else
                  if (val > rval)
                        r = r.getRight();
                else
                {
                    found = true;
                    break;
                }
            found = search(r, val);
```

```
            }
            return found;
        }

        /* Function for inorder traversal */
        public void inorder()
        {
          inorder(root);
        }

        private void inorder(BSTNode r)
        {
          if (r != null)
            {
                inorder(r.getLeft());
                System.out.print(r.getData() +" ");
                inorder(r.getRight());
            }
        }

        /* Function for preorder traversal */
        public void preorder()
        {
          preorder(root);
        }

        private void preorder(BSTNode r)
        {
          if (r != null)
            {
                System.out.print(r.getData() +" ");
                preorder(r.getLeft());
                preorder(r.getRight());
            }
        }

        /* Function for postorder traversal */
        public void postorder()
        {
          postorder(root);
        }

        private void postorder(BSTNode r)
        {
          if (r != null)
            {
                postorder(r.getLeft());
                postorder(r.getRight());
                System.out.print(r.getData() +" ");
            }
        }
}

/////////////////////////////////////

import java.util.Scanner;

/* Class BinarySearchTree */

public class Lab7_Q1
{
```

```
      public static void main(String[] args)
    {
            Scanner scan = new Scanner(System.in);

        /* Creating object of BST */
        BST bst = new BST();
        System.out.println("Binary Search Tree Test\n");
        char ch;

        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. delete");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");

            int choice = scan.nextInt();
            switch (choice)
            {
                  case 1 :
                        System.out.println("Enter integer element to
insert");
                    bst.insert( scan.nextInt() );
                    break;
                case 2 :
                    System.out.println("Enter integer element to delete");
                    bst.delete( scan.nextInt() );
                    break;
                case 3 :
                    System.out.println("Enter integer element to search");
                    System.out.println("Search result : "+ bst.search(
scan.nextInt() ));
                    break;
                case 4 :
                    System.out.println("Nodes = "+ bst.countNodes());
                    break;
                case 5 :
                    System.out.println("Empty status = "+ bst.isEmpty());
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
            }

            /*  Display tree  */
            System.out.print("\nPost order : ");
            bst.postorder();
            System.out.print("\nPre order : ");
            bst.preorder();
            System.out.print("\nIn order : ");
            bst.inorder();
            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');

    }
}
```

Question 2

You have to maintain information for a shop owner. For each of the products sold in his/hers shop the following information is kept: a unique code, a name, a price, amount in stock, date received, and expiration date. For keeping track of its stock, the clerk would use a computer program based on a search tree data structure. Write a program to help this person, by implementing following the following operations:

- Insert an item with all its associated data.
- Find an item by its code, and support updating of the item found.
- List valid items in alphabetical order of their names.
- List expired items in alphabetical order of their names.
- List all items.
- Delete an item given by its code.
- Delete all expired items.
- Create a separate search tree for expired items.
- Save stock in file stock.data.
- Exit

If file *stock.data* exists, your program must automatically load its contents.

I/O description. This program should be interactive.

Solution Q2

```
import java.util.Date;
/* Class BSTNode */
class Item
{
    int code;
    String name;
    double price;
    int amount;
    Date receivedDate, expirationDate;

    /* Constructor */
    public Item()
    {
      this.code = 0;
        this.name = null;
        this.price = 0;
        this.amount = 0;
        this.receivedDate = null;
        this.expirationDate = null;
    }
```

```
    /* Constructor */
    public Item(int code, String name, double price, int amount,
                  Date receivedDate, Date expirationDate )
    {
      this.code = code;
        this.name = name;
        this.price = price;
        this.amount = amount;
        this.receivedDate = receivedDate;
        this.expirationDate = expirationDate;
    }
    public String toString()
    {
        return "Item information [code=" + code + ", name=" + name + ",
price=" + price
                + ", amount=" + amount +
                ", receivedDate=" + receivedDate + ", expirationDate=" +
expirationDate + "]";
    }

}

////////////////////////////////

/*  Java Program to Implement Binary Search Tree

*/

import java.util.Date;

/* Class BSTNode */

class BSTNode

{

      BSTNode left, right;

      Item data;


    /* Constructor */

    public BSTNode(Item element)

    {

      left = null;

        right = null;

        data = new Item(element.code, element.name, element.price,

                  element.amount, element.receivedDate,
element.expirationDate);

    }


    /* Function to set left node */

    public void setLeft(BSTNode n)
```

```
    {
        left = n;
    }

    /* Function to set right node */
    public void setRight(BSTNode n)
    {
        right = n;
    }

    /* Function to get left node */
    public BSTNode getLeft()
    {
      return left;
    }

    /* Function to get right node */
    public BSTNode getRight()
    {
        return right;
    }

    /* Function to set data to node */
    public void setData(Item d)
    {
        data = d;
    }
    /* Function to get data from node */
    public Item getData()
    {
        return data;
    }
}
/////////////////////////////
```

```
/* Class BST */
import java.util.Date;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

class BST
{
      private BSTNode root;

    /* Constructor */
    public BST()
    {
      root = null;
    }

    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
      return root == null;
    }

    /* Functions to insert data */
    public void insert(Item data)
    {
      root = insert(root, data);
    }

    /* Function to insert data recursively */
    private BSTNode insert(BSTNode node, Item data)
    {
      if (node == null)
            node = new BSTNode(data);
       else
       {
           if (data.code <= node.getData().code)
                 node.left = insert(node.left, data);
           else
               node.right = insert(node.right, data);
       }
           return node;
     }

     /* Functions to delete data */
     public void delete(int k)
     {
       if (isEmpty())
             System.out.println("Tree Empty");
        else
           if (search(k) == null)
                 System.out.println("Sorry "+ k +" is not present");
           else
           {
               root = delete(root, k);
               System.out.println(k+ " deleted from the tree");
           }
     }

     private BSTNode delete(BSTNode root, int k)
     {
       BSTNode p, p2, n;
```

```
        if (root.getData().code == k)
        {
            BSTNode lt, rt;
            lt = root.getLeft();
            rt = root.getRight();
            if (lt == null && rt == null)
                  return null;
            else
                  if (lt == null)
                {
                    p = rt;
                    return p;
                }
                else
                  if (rt == null)
                  {
                        p = lt;
                        return p;
                  }
                  else
                  {
                        p2 = rt;
                        p = rt;
                        while (p.getLeft() != null)
                              p = p.getLeft();
                        p.setLeft(lt);
                        return p2;
                  }
        }
        if (k < root.getData().code)
        {
            n = delete(root.getLeft(), k);
            root.setLeft(n);
        }
        else
        {
            n = delete(root.getRight(), k);
            root.setRight(n);
        }
        return root;
    }

    /* Functions to count number of nodes */
    public int countNodes()
    {
      return countNodes(root);
    }

    /* Function to count number of nodes recursively */
    private int countNodes(BSTNode r)
    {
      if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }
```

```
/* Functions to search for an element */
public BSTNode search(int val)
{
  return search(root, val);
}

/* Function to search for an element recursively */
private BSTNode search(BSTNode r, int val)
{
    while (r != null)
    {
        if (val < r.getData().code)
              return search(r.getLeft(), val);
        else
              if (val > r.getData().code)
              return search(r.getRight(), val);
            else
                return r;
    }
    return null;
}

/* Function for inorder traversal */
public void inorder()
{
  inorder(root);
}

private void inorder(BSTNode r)
{
  if (r != null)
    {
        inorder(r.getLeft());
        System.out.print(r.getData() +"\n");
        inorder(r.getRight());
    }
}

/* Function for preorder traversal */
public void preorder()
{
  preorder(root);
}

private void preorder(BSTNode r)
{
  if (r != null)
    {
        System.out.print(r.getData() +"\n");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}

/* Function for postorder traversal */
public void postorder()
{
  postorder(root);
}
```

```
private void postorder(BSTNode r)
{
  if (r != null)
    {
        postorder(r.getLeft());
        postorder(r.getRight());
        System.out.print(r.getData() +"\n");
    }
}
public void deleteExpiredDate()
{
  deleteExpiredDate(root);
}

private void deleteExpiredDate(BSTNode r)
{
  if (r != null)
    {
        deleteExpiredDate(r.getLeft());
        if (r.getData().expirationDate.compareTo(new Date()) < 0 )
              delete(r.getData().code);
        deleteExpiredDate(r.getRight());
    }
}

public void createExpiredDateTree(BST temp)
{
  createExpiredDateTree(root,temp);
}

private void createExpiredDateTree(BSTNode r, BST temp)
{
  if (r != null)
    {
        createExpiredDateTree(r.getLeft(),temp);
        if (r.getData().expirationDate.compareTo(new Date()) < 0 )
              temp.insert(r.getData());
        createExpiredDateTree(r.getRight(),temp);
    }
}

public void insertByName(Item data)
{
 root = insertByName(root, data);
}

/* Function to insert data recursively */
private BSTNode insertByName(BSTNode node, Item data)
{
 if (node == null)
       node = new BSTNode(data);
    else
    {
        if (data.name.compareTo(node.getData().name) <= 0)
             node.left = insert(node.left, data);
        else
            node.right = insert(node.right, data);
    }
       return node;
 }
```

```
    public void createValidSortedByNameTree(BST temp)
    {
      createValidSortedByNameTree(root,temp);
    }

    private void createValidSortedByNameTree(BSTNode r, BST temp)
    {
      if (r != null)
        {
            createValidSortedByNameTree(r.getLeft(),temp);
            if (r.getData().expirationDate.compareTo(new Date()) >= 0 )
                  temp.insertByName(r.getData());

            createValidSortedByNameTree(r.getRight(),temp);
        }
    }

    public void createInValidSortedByNameTree(BST temp)
    {
      createInValidSortedByNameTree(root,temp);
    }

    private void createInValidSortedByNameTree(BSTNode r, BST temp)
    {
      if (r != null)
        {
            createInValidSortedByNameTree(r.getLeft(),temp);
            if (r.getData().expirationDate.compareTo(new Date()) < 0 )
                  temp.insertByName(r.getData());

            createInValidSortedByNameTree(r.getRight(),temp);
        }
    }

    public void writeToFile(String fileName) throws IOException
    {
      FileWriter write = new FileWriter(fileName,false);
      PrintWriter printLine = new PrintWriter(write);
      writeToFile(root,printLine);
      printLine.close();
    }

    private void writeToFile(BSTNode r, PrintWriter file)
    {
      if (r != null)
        {
            writeToFile(r.getLeft(),file);
            file.printf("%s"+"%n",r.getData().code +", "+ r.getData().name
+", "+ r.getData().price +", "+
                        r.getData().amount +", "+
r.getData().expirationDate +", "+ r.getData().expirationDate);

            writeToFile(r.getRight(),file);
        }
    }
}


////////////////////////////////

import java.util.Scanner;
```

```
import java.io.IOException;
import java.text.*;

/* Class BinarySearchTree */

public class Lab7_Q2
{
      public static void main(String[] args) throws IOException
    {
            Scanner scan = new Scanner(System.in);

        /* Creating object of BST */
        BST bst = new BST();
        Item element = new Item();
        String next;
        BST bstExpired;
        BST bstName;

        System.out.println("Binary Search Tree Test\n");
        char ch;

        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert Item ");
            System.out.println("2. Find an item by its code ");
            System.out.println("3. List valid items in alphabetical order
of their names ");
            System.out.println("4. List expired items in alphabetical order
of their names ");
            System.out.println("5. List all items ");
            System.out.println("6. Delete an item given by its code ");
            System.out.println("7. Delete all expired items. ");
            System.out.println("8. Create a separate search tree for
expired items");
```

```
            System.out.println("9. Save stock in file stock.data");

            int choice = scan.nextInt();
            switch (choice)
            {
                  case 1 :
                        System.out.println("Enter Item Code");
                        element.code = scan.nextInt();
                        System.out.println("Enter Item Name");
                        element.name = scan.next();
                        System.out.println("Enter Item Price");
                        element.price = scan.nextDouble();
                        System.out.println("Enter Item Amount");
                        element.amount = scan.nextInt();
                        System.out.println("Enter Item received date");

                        next = scan.next("[0-9]{2}/[0-9]{2}/[0-9]{4}");
                        try
                        {
                              element.receivedDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                        } catch (ParseException e) {
                            e.printStackTrace();
                        }
                        System.out.println("Enter Item expiration date");
                  next = scan.next("[0-9]{2}/[0-9]{2}/[0-9]{4}");
                        try
                        {
                              element.expirationDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                        } catch (ParseException e) {
                            e.printStackTrace();
                        }

                    bst.insert(element);
```

```
                    break;
                case 2 :
                    System.out.println("Enter code item to search");
                    BSTNode temp = bst.search( scan.nextInt());
                    if (temp == null)
                        System.out.println("Not found");
                    else
                    {
                        System.out.println("Item Information =
"+temp.getData());
                        System.out.println("Do you want to modify the
information(Y/N)");
                        if (scan.next().charAt(0) == 'Y')
                        {
                              element.code = temp.data.code;
                              System.out.println("Enter Item Name");
                              element.name = scan.next();
                              System.out.println("Enter Item Price");
                              element.price = scan.nextDouble();
                              System.out.println("Enter Item Amount");
                              element.amount = scan.nextInt();
                              System.out.println("Enter Item received
date");

                              next = scan.next("[0-9]{2}/[0-9]{2}/[0-
9]{4}");
                              try
                              {
                                    element.receivedDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                              } catch (ParseException e) {
                                  e.printStackTrace();
                              }
                              System.out.println("Enter Item expiration
date");
                              next = scan.next("[0-9]{2}/[0-9]{2}/[0-
9]{4}");
```

```
                                try
                                {
                                      element.expirationDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                                } catch (ParseException e) {
                                    e.printStackTrace();
                                }
                                temp.data = element;
                            }
                        }
                        break;
                    case 3 :
                      bstName = new BST();
                      bst.createValidSortedByNameTree(bstName);
                      bstName.inorder();
                        break;
                    case 4 :
                      bstName = new BST();
                      bst.createInValidSortedByNameTree(bstName);
                      bstName.inorder();
                        break;
                    case 5 :
                      bst.inorder();
                        break;
                    case 6 :
                        System.out.println("Enter code item to delete");
                        bst.delete( scan.nextInt());
                        break;
                    case 7 :
                      bst.deleteExpiredDate();
                      bst.inorder();
                        break;
                    case 8 :
                      bstExpired = new BST();
                      bst.createExpiredDateTree(bstExpired);
```

```
                  bstExpired.inorder();
                    break;
                case 9 :
                    bst.writeToFile("output.txt");
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
            }

            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');

    }
}
```

Question 3

You have to maintain information for school classes. For each of the students in a class the following information is kept: a unique code, student name, birth date, home address, id of class he is currently in, date of enrollment, status (undergrad, graduate). For keeping track of the students, the school secretary would use a computer program based on a search tree data structure. Write a program to help the secretary, by implementing following the following operations:

- Insert an item with all its associated data.
- Find a student by his/hers unique code, and support updating of the student info if found.
- List students by class in lexicographic order of their names.
- List all students in lexicographic order of their names.
- List all graduated students.
- List all undergrads by their class in lexicographic order of their names.
- Delete a student given by its code.
- Delete all graduates.
- Save all students in file *student.data*.
- Exit.

If file *student.data* exists, your program must automatically load its contents. .

I/O description. This program should be interactive.

Solution Q3

```
import java.util.Date;
/* Class BSTNode */
class Item
{
    int code;
    String name;
    Date birthDate ;
    String address;
    int classId;
    Date enrollmentDate;
    byte status;  // undergrad = 1, graduate = 2

    /* Constructor */
    public Item()
    {
      this.code = 0;
```

```
        this.name = null;
        this.birthDate = null;
        this.address = null;
        this.classId = 0;
        this.enrollmentDate = null;
        this.status = 0;
    }

    /* Constructor */
    public Item(int code, String name, Date birthDate, String address,
            int classId, Date enrollmentDate, byte status)
    {
      this.code = code;
        this.name = name;
        this.birthDate = birthDate;
        this.address = address;
        this.classId = classId;
        this.enrollmentDate = enrollmentDate;
        this.status = status;
    }
    public String toString()
    {
        return "Student information [Code=" + code + ", Name=" + name + ",
Birth Date=" + birthDate
                + ", Address=" + address + ", classId=" + classId +
                ", Enrollment Date=" + enrollmentDate + ", Status=" +
status + "]";
    }

}
////////////////////////////

/*
  Java Program to Implement Binary Search Tree
*/
import java.util.Date;
/* Class BSTNode */
class BSTNode
{
      BSTNode left, right;
      Item data;

    /* Constructor */
    public BSTNode(Item element)
    {
      left = null;
        right = null;
        data = new Item(element.code, element.name, element.birthDate,
                  element.address, element.classId, element.enrollmentDate,
element.status);
    }

    /* Function to set left node */
    public void setLeft(BSTNode n)
    {
        left = n;
    }

    /* Function to set right node */
    public void setRight(BSTNode n)
```

```
    {
        right = n;
    }

    /* Function to get left node */
    public BSTNode getLeft()
    {
      return left;
    }

    /* Function to get right node */
    public BSTNode getRight()
    {
        return right;
    }

    /* Function to set data to node */
    public void setData(Item d)
    {
        data = d;
    }

    /* Function to get data from node */
    public Item getData()
    {
        return data;
    }
}
/////////////////////////////

/* Class BST */
import java.util.Date;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

class BST
{
      private BSTNode root;

    /* Constructor */
    public BST()
    {
      root = null;
    }

    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
      return root == null;
    }

    /* Functions to insert data */
    public void insert(Item data)
    {
      root = insert(root, data);
    }

    /* Function to insert data recursively */
    private BSTNode insert(BSTNode node, Item data)
    {
```

```
 if (node == null)
       node = new BSTNode(data);
   else
   {
       if (data.code <= node.getData().code)
             node.left = insert(node.left, data);
       else
           node.right = insert(node.right, data);
   }
       return node;
}

/* Functions to delete data */
public void delete(int k)
{
  if (isEmpty())
        System.out.println("Tree Empty");
    else
       if (search(k) == null)
             System.out.println("Sorry "+ k +" is not present");
       else
       {
           root = delete(root, k);
           System.out.println(k+ " deleted from the tree");
       }
}

private BSTNode delete(BSTNode root, int k)
{
  BSTNode p, p2, n;
    if (root.getData().code == k)
    {
       BSTNode lt, rt;
       lt = root.getLeft();
       rt = root.getRight();
       if (lt == null && rt == null)
             return null;
       else
             if (lt == null)
           {
               p = rt;
               return p;
           }
           else
             if (rt == null)
             {
                   p = lt;
                   return p;
             }
             else
             {
                   p2 = rt;
                   p = rt;
                   while (p.getLeft() != null)
                         p = p.getLeft();
                   p.setLeft(lt);
                   return p2;
             }
    }
    if (k < root.getData().code)
    {
```

```
            n = delete(root.getLeft(), k);
            root.setLeft(n);
        }
        else
        {
            n = delete(root.getRight(), k);
            root.setRight(n);
        }
        return root;
    }

    /* Functions to count number of nodes */
    public int countNodes()
    {
      return countNodes(root);
    }

    /* Function to count number of nodes recursively */
    private int countNodes(BSTNode r)
    {
      if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }

    /* Functions to search for an element */
    public BSTNode search(int val)
    {
      return search(root, val);
    }

    /* Function to search for an element recursively */
    private BSTNode search(BSTNode r, int val)
    {
        while (r != null)
        {
            if (val < r.getData().code)
                  return search(r.getLeft(), val);
            else
                  if (val > r.getData().code)
                  return search(r.getRight(), val);
                else
                    return r;
        }
        return null;
    }

    /* Function for inorder traversal */
    public void inorder()
    {
      inorder(root);
    }

    private void inorder(BSTNode r)
    {
```

```
  if (r != null)
    {
        inorder(r.getLeft());
        System.out.print(r.getData() +"\n");
        inorder(r.getRight());
    }
}

/* Function for preorder traversal */
public void preorder()
{
  preorder(root);
}

private void preorder(BSTNode r)
{
  if (r != null)
    {
        System.out.print(r.getData() +"\n");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}

/* Function for postorder traversal */
public void postorder()
{
  postorder(root);
}

private void postorder(BSTNode r)
{
  if (r != null)
    {
        postorder(r.getLeft());
        postorder(r.getRight());
        System.out.print(r.getData() +"\n");
    }
}
//////////////////////
public void deleteGraduates()
{
  deleteGraduates(root);
}

private void deleteGraduates(BSTNode r)
{
  if (r != null)
    {
        deleteGraduates(r.getLeft());
        if (r.getData().status == 2 )
              delete(r.getData().code);
        deleteGraduates(r.getRight());
    }
}
//////////////////////////
public void insertByClassAndName(BST temp, int classId)
{
  insertByClassAndName(root,temp,classId);
}
```

```
private void insertByClassAndName(BSTNode r, BST temp, int classId)
{
  if (r != null)
    {
        insertByClassAndName(r.getLeft(),temp, classId);
        if (r.getData().classId == classId )
              temp.insertByName(r.getData());
        insertByClassAndName(r.getRight(),temp, classId);
    }
}
//////////////////////////
public void insertByName(Item data)
{
 root = insertByName(root, data);
}

/* Function to insert data recursively */
private BSTNode insertByName(BSTNode node, Item data)
{
 if (node == null)
       node = new BSTNode(data);
    else
    {
        if (data.name.compareTo(node.getData().name) <= 0)
             node.left = insert(node.left, data);
        else
            node.right = insert(node.right, data);
    }
       return node;
}
//////////////////
public void createTreeByName(BST temp)
{
  createTreeByName(root,temp);
}

private void createTreeByName(BSTNode r, BST temp)
{
  if (r != null)
    {
        createTreeByName(r.getLeft(),temp);
              temp.insertByName(r.getData());

        createTreeByName(r.getRight(),temp);
    }
}
//////////////
public void listGraduate()
{
  listGraduate(root);
}

private void listGraduate(BSTNode r)
{
  if (r != null)
    {
        listGraduate(r.getLeft());
        if (r.getData().status == 2)
              System.out.print(r.getData() +"\n");

        listGraduate(r.getRight());
```

```
        }
    }
    //////////////////
    public void listUndergradsByClassAndName()
    {
      listUndergradsByClassAndName(root);
    }

    private void listUndergradsByClassAndName(BSTNode r)
    {
      if (r != null)
        {
            listUndergradsByClassAndName(r.getLeft());
            if (r.getData().status == 1)
                  System.out.print(r.getData() +"\n");

            listUndergradsByClassAndName(r.getRight());
        }
    }
    //////////////////
    public void writeToFile(String fileName) throws IOException
    {
      FileWriter write = new FileWriter(fileName,false);
      PrintWriter printLine = new PrintWriter(write);
      writeToFile(root,printLine);
      printLine.close();
    }

    private void writeToFile(BSTNode r, PrintWriter file)
    {
      if (r != null)
        {
            writeToFile(r.getLeft(),file);
            file.printf("%s"+"%n",r.getData().code +", "+ r.getData().name
+", "+ r.getData().birthDate +", "+
                        r.getData().address +", "+ r.getData().classId
                        +", "+ r.getData().status);

            writeToFile(r.getRight(),file);
        }
    }
}

//////////////////////////
import java.util.Scanner;
import java.io.IOException;
import java.text.*;

/* Class BinarySearchTree */

public class Lab7_Q3
{
      public static void main(String[] args) throws IOException
    {
            Scanner scan = new Scanner(System.in);

        /* Creating object of BST */
        BST bst = new BST();
        Item element = new Item();
        String next;
        BST bstExpired;
```

```
        BST bstName;
        int classId;

        System.out.println("Binary Search Tree Test\n");
        char ch;

        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert student ");
            System.out.println("2. Find a student by its code ");
            System.out.println("3. List students in alphabetical order of
their names for a specific class ");
            System.out.println("4. List students in alphabetical order of
their names ");
            System.out.println("5. List all graduated students ");
            System.out.println("6. List all undergrads for a specific class
in alphabetical order of their names ");
            System.out.println("7. Delete a student given by its code ");
            System.out.println("8. Delete all graduates");
            System.out.println("9. Save stock in file stock.data");

            int choice = scan.nextInt();
            switch (choice)
            {
                  case 1 :
                        System.out.println("Enter student Code");
                        element.code = scan.nextInt();

                        System.out.println("Enter student Name");
                        element.name = scan.nextLine();

                        System.out.println("Enter student birth date");
                        next = scan.next("[0-9]{2}/[0-9]{2}/[0-9]{4}");
                        try
                        {
                              element.birthDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                        } catch (ParseException e)
                        {
                            e.printStackTrace();
                        }
                        System.out.println("Enter student address");
                        element.address = scan.nextLine();

                        System.out.println("Enter class Id");
                        element.classId = scan.nextInt();
                        System.out.println("Enter student enrollment
date");
                        next = scan.next("[0-9]{2}/[0-9]{2}/[0-9]{4}");
                        try
                        {
                              element.enrollmentDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                        } catch (ParseException e)
                        {
                            e.printStackTrace();
                        }
```

```
                        System.out.println("Enter Item status (undergrad =
1, graduate = 2)");
                        element.classId = scan.nextByte();

                    bst.insert(element);
                    break;
                case 2 :
                    System.out.println("Enter student code to search");
                    BSTNode temp = bst.search( scan.nextInt());
                    if (temp == null)
                        System.out.println("Not found");
                    else
                    {
                        System.out.println("Student Information =
"+temp.getData());
                        System.out.println("Do you want to modify the
student information(Y/N)");
                        if (scan.next().charAt(0) == 'Y')
                        {
                              System.out.println("Enter student Code");
                              element.code = scan.nextInt();

                              System.out.println("Enter student Name");
                              element.name = scan.next();

                              System.out.println("Enter student birth
date");
                              next = scan.next("[0-9]{2}/[0-9]{2}/[0-
9]{4}");
                              try
                              {
                                    element.birthDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                              } catch (ParseException e)
                              {
                                  e.printStackTrace();
                              }
                              System.out.println("Enter student address");
                              element.address = scan.next();

                              System.out.println("Enter class Id");
                              element.classId = scan.nextInt();
                              System.out.println("Enter student enrollment
date");
                              next = scan.next("[0-9]{2}/[0-9]{2}/[0-
9]{4}");
                              try
                              {
                                    element.enrollmentDate = new
SimpleDateFormat("dd/MM/yyyy").parse(next);
                              } catch (ParseException e)
                              {
                                  e.printStackTrace();
                              }

                              System.out.println("Enter Item status
(undergrad = 1, graduate = 2)");
                              element.status = scan.nextByte();

                              temp.data = element;
                        }
```

```
                    }
                    break;
                case 3 :
                        System.out.println("Enter class Id");
                        classId = scan.nextInt();
                  bstName = new BST();
                  bst.insertByClassAndName(bstName, classId);
                  bstName.inorder();
                    break;
                case 4 :
                  bstName = new BST();
                  bst.createTreeByName(bstName);
                  bstName.inorder();
                    break;
                case 5 :
                  bst.listGraduate();
                    break;
                case 6 :
                        System.out.println("Enter class Id");
                        classId = scan.nextInt();
                  bstName = new BST();
                  bst.insertByClassAndName(bstName, classId);
                  bstName.listUndergradsByClassAndName();
                    break;
                case 7 :
                    System.out.println("Enter student code to delete");
                    bst.delete(scan.nextInt());
                    break;
                case 8 :
                  bst.deleteGraduates();
                  bst.inorder();
                    break;
                case 9 :
                    bst.writeToFile("output.txt");
                    break;
                default :
                    System.out.println("Wrong Entry \n ");
                    break;
            }

            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');

    }
}
```

Question 4

A database is a collection of related pieces of information that is organized for easy retrieval. The set of account records shown below, for instance, forms an accounts database.

Record No.	Account ID	First Name	Last Name	Balance
0	6274	James	Johnson	415.56
1	2843	Marcus	Wilson	9217.23
2	4892	Maureen	Albrigh	51462.56
3	8337	Debra	Douglas	27.26
4	9523	Bruce	Gold	719.32
5	3165	John	Carlson	1496.24

Each record in the accounts database is assigned a record number based on that record's relative position within the database file. You can use a record number to retrieve an account record directly, much as you can use an array index to reference an array element directly. The following program from the file AccountRec.java, for example, retrieves a record from the accounts database in the file Accounts.dat. Notice that both keyboard and file input are used in this program.

```
import java.io.*;
import java.util.StringTokenizer;
class AccountRec
{
        // Constants
        private static final long bytesPerRecord = 38; // Number of bytes used to store
                                                       // each record in the accounts
                                                       // database file
                                                       // Data members
        private int acctID;                            // Account identifier
```

```
private String firstName;                    // Name of account holder
private String lastName;
private double balance;                      // Account balance
public static void main (String args[ ]) throws IOException
{
        AccountRec acctRec = new AccountRec( ); // Account record
        long recNum; // User input record number
        String str, // For reading a String
        name;
        // Need random access on the accounts database file; r = read only
        RandomAccessFile inFile =
                new RandomAccessFile("Accounts.dat", "r");
        // Also need tokenized input stream from keyboard
        BufferedReader reader =
                new BufferedReader(new InputStreamReader(System.in));
        StreamTokenizer keybdTokens = new StreamTokenizer(reader);
        // Get the record number to retrieve.
        System.out.println( );
        System.out.print("Enter record number: ");
        keybdTokens.nextToken( );
        recNum = (long)keybdTokens.nval;
        // Move to the corresponding record in the database file using the
        // seek( ) method in RandomAccessFile.
        inFile.seek(recNum * bytesPerRecord);

        str = inFile.readLine( ); // Read the record
        if (str != null) // Is there something in the string?
        {
                // Need to tokenize the String read by readline( )
                StringTokenizer strTokens = new StringTokenizer(str);
                name = strTokens.nextToken( ); // first String token
                acctRec.acctID = Integer.parseInt(name); // Convert String to an int
```

```
                    acctRec.firstName = strTokens.nextToken( ); // 2nd String token -
                                                                // firstName
                    acctRec.lastName = strTokens.nextToken( ); // 3rd String token -
                                                                // lastName
                    name = strTokens.nextToken( ); // 4th String token
                    // Convert the String to a double
                    acctRec.balance = Double.parseDouble(name);
                    // Display the record.
                    System.out.println(recNum + " : " + acctRec.acctID + " "
                            + acctRec.firstName + " " + acctRec.lastName + " "
                            + acctRec.balance);
            }
            else
                    System.out.println("Reached EOF");
            // Close the file streams
            inFile.close( );
    } // main
} // class AccountRec
```

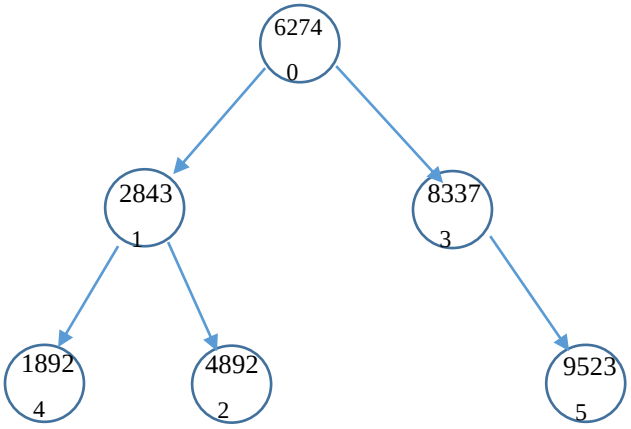
Record numbers are assigned by the database file mechanism and are not part of the account information. As a result, they are not meaningful to database users. These users require a different record retrieval mechanism, one that is based on an account ID (the key for the database) rather than a record number.

Retrievals based on account ID require an index that associates each account ID with the corresponding record number. You can implement this index using a binary search tree in which each element contains the two fields: an account ID (the key) and a record number. Since the binary search tree stores elements of type *TreeElem*, the class IndexEntry given below can be used to implement this index.

```
class IndexEntry implements TreeElem
{
    int acctID; // (Key) Account identifier
```

```
        long recNum; // Record number
        public int key ( )
        {
                return acctID;
        } // Return key field
}
```

You build the index by reading through the database account by account, inserting successive (account ID, record number) pairs into the tree as you progress through the file. The following index tree, for instance, was produced by inserting the account IndexEntry elements from the database records shown above into an (initially) empty tree.



Given an account ID, retrieval of the corresponding account record is a two-step process. First, you retrieve the element from the index tree that has the specified account ID. Then, using the record number stored in the index element, you read the corresponding account record from the database file. The result is an efficient retrieval process that is based on account ID.

Step 1: Create a program that builds an index tree for the accounts database in the file Accounts.dat. Once the index is built, your program should

- Output the account IDs in ascending order.
- Read an account ID from the keyboard and output the corresponding account record.

Step 2: Test your program using the accounts database in the text file Accounts.dat. A copy of this database in given below. Try to retrieve several account IDs, including account IDs that do not occur in the database. A test plan form follows.

Record No.	Account ID	First Name	Last Name	Balance
0	6274	James	Johnson	415.56
1	2843	Marcus	Wilson	9217.23
2	4892	Maureen	Albrigh	51462.56
3	8337	Debra	Douglas	27.26
4	9523	Bruce	Gold	719.32
5	3165	John	Carlson	1496.24
6	1892	Mary	Smith	918.26
7	3924	Simon	Chang	386.85
8	6023	John	Edgar	9.65
9	5290	George	Truman	16110.68
10	8529	Elena	Gomez	86.77
11	1144	Donald	Williams	4114.26

Lab 8: AVL Tree

Exercises:

Question 1

Implement the AVL tree ADT. You must implement the tree class, including *insert()* and *print()* methods, as well as the node class. The tree must maintain the AVL property. The *insert()* method inserts an item into the AVL tree (using the normal binary search tree insert procedure), and then re-balances the trees, using rotations (if necessary). The *print()* method outputs the AVL tree to the console, using an in-order traversal. The output should include the balance for each node (right subtree height – left subtree height). Sample output from *print()* is shown below:

80 (-1)

70 (0)

60 (1)

50 (0)

40 (1)

30 (0)

20 (0)

10 (0)

Test the AVL tree using the driver class provided.

Solution Q1

```
public class AVLNode
{
	int data;	// Data in the node
	AVLNode left;	// Left child
	AVLNode right;	// right child
	int height;	// Height

	// Constructors
	public AVLNode (int d)
	{
		this(d, null, null);
	}
	public AVLNode (int d, AVLNode lt, AVLNode rt)
	{
		data = d;
		left = lt;
		right = rt;
		height = 0;
	}
}
```

```
//////////////////////////////////////

public class AVLTree
{

	private AVLNode root; // The tree root

	public AVLTree( ) // Construct the tree
	{
		root = null;
	}


	public void makeEmpty( ) // Make the tree logically empty.
	{
		root = null;
	}

	/*
	Test if the tree is logically empty.
	return true if empty, false otherwise.
	*/
	public boolean isEmpty( )
	{
		return root == null;
	}

	/*
	Print the tree contents in sorted order.
	*/
	public void printTree( )
	{
		if( isEmpty( ) )
			System.out.println( "Empty tree" );
		else
			printTree( root );
	}

	/*
	method to print the tree in sorted order.
	t the node that roots the tree.
	*/
	private void printTree( AVLNode t ) // inorder traversal
	{
		if( t != null )
		{
			printTree( t.left );
			System.out.println( t.data + " ("+ height(t) +")" );
			printTree( t.right );
		}
	}

	/*
	Insert into the tree; duplicates are ignored.
	x the item to insert.
	*/
	public void insert( int x )
	{
		root = insert( x, root );
	}
```

```
/*
method to insert into a subtree.
x the item to insert.
t the node that roots the subtree.
return the new root of the subtree.
*/

private AVLNode insert( int x, AVLNode t )
{
      if( t == null )
            return new AVLNode( x, null, null );
      if( x < t.data )
      {
            t.left = insert( x, t.left );
            if( height( t.left ) - height( t.right ) == 2 )
                  if( x < t.left.data )
                        t = rotateLeft( t );
                  else
                        t = doubleLeft( t );
      }
      else if( x > t.data )
      {
            t.right = insert( x, t.right );
            if( height( t.right ) - height( t.left ) == 2 )
                  if( x > t.right.data )
                        t = rotateRight( t );
                  else
                        t = doubleRight( t );
      }
      else
            ; // Duplicate; do nothing
      t.height = Math.max( height( t.left ), height( t.right )) + 1;
      return t;
}

/*
Find an item in the tree.
x the item to search for.
return true if x is found.
*/

public boolean search( int x )
{
      return search( x, root );
}

/*
method to find an item in a subtree.
x is item to search for.
t the node that roots the tree.
return true if x is found in subtree.
*/
private boolean search( int x, AVLNode t )
{
      while( t != null )
      {
            if( x < t.data )
                  t = t.left;
            else
                  if( x > t.data )
                        t = t.right;
```

```
                        else
                              return true; // Match
                }
                return false; // No match
        }

        private int height( AVLNode t ) // return height of node t, or -1, if
null.
        {
                if( t == null )
                        return -1;
                else
                        return t.height;
        }

        /*
        Rotate binary tree node with left child.
        For AVL trees, this is a single rotation.
        Update heights, then return new root.
        */
        private AVLNode rotateLeft( AVLNode node2 )
        {
                AVLNode node1 = node2.left;
                node2.left = node1.right;
                node1.right = node2;
                node2.height = Math.max(height(node2.left),
height(node2.right))+1;
                node1.height = Math.max(height(node1.left), node2.height)+1;
                return node1;
        }

        /*
        Rotate binary tree node with right child.
        For AVL trees, this is a single rotation.
        Update heights, then return new root.
        */
        private AVLNode rotateRight( AVLNode node1 )
        {
                AVLNode node2 = node1.right;
                node1.right = node2.left;
                node2.left = node1;
                node1.height = Math.max(height(node1.left),
height(node1.right))+1;
                node2.height = Math.max(height(node2.right), node1.height)+1;
                return node2;
        }

        /*
        Double rotate binary tree node: first left child with its right
child;
        then node node3 with new left child.
        For AVL trees, this is a double rotation.
        Update heights, then return new root.
        */
        private AVLNode doubleLeft( AVLNode node3 )
        {
                node3.left = rotateRight( node3.left );
                return rotateLeft( node3 );
        }

        /*
```

```
      Double rotate binary tree node: first right child with its left
child;
      then node node1 with new right child.
      For AVL trees, this is a double rotation.
      Update heights, then return new root.
      */
      private AVLNode doubleRight( AVLNode node1 )
      {
            node1.right = rotateLeft( node1.right );
            return rotateRight( node1 );
      }
}

////////////////////// AVLTreeDemo.java ////////////////////////////
class Lab8_Q1
{
      public static void main( String [] args )
      {
            AVLTree avl = new AVLTree();
            int[] a = {30,80,50,40,20,60,70,10,90,95};

            // build tree by successive insertions of 30, 80, 50, 40 ...
            for( int i = 0; i < a.length; i++ )
                  avl.insert( a[i] );
            System.out.println( "\nAVL Tree nodes in sorted order:");
            avl.printTree();
            avl.insert( 15 );
            avl.insert( 85 );
            System.out.println( "\n\nAfter insertion of 15 & 85");
            avl.printTree();
            int item = 82; // search for an item
            if( avl.search( item ) )
                  System.out.println( "\n\n" + item + " found");
            else
                  System.out.println( "\n\n" + item + " not found");
      }
}
```

Question 2

Create a method to check if a binary tree is an AVL tree without using the height method.

Solution Q2

```
public class Lab8_Q2
{
    public boolean isBalanced(TreeNode root)
    {
        if(root == null)
            return true;
        if(Math.abs(depth(root.left)-depth(root.right)) > 1)
            return false;
        return isBalanced(root.left)&&isBalanced(root.right);
    }

    public static int depth(TreeNode root)
    {
        if(root==null)
            return 1;
        return 1+Math.max(depth(root.left), depth(root.right));
    }
}
```

Lab 9: Hash Tables

Exercises:

Question 1

Write a program to manage a hash table, using open addressing, where the keys are book ISBNs. Your code should provide create, insert, find and delete operations on that table.
Use a hash function suitable for character strings and motivate your answer in a comment stored in the heading area of your hash table processing routines. Output for find should be yes, followed by the table index if found or no if not found.

Solution Q1

```
public class HashEntry
{
	private String key;
	private String value;
	private byte status;	// insert: 1, delete: 2, empty: 0

	HashEntry(String key, String value, byte status)
	{
		this.key = key;
		this.value = value;
		this.status = status;
	}

	public String getKey()
	{
		return key;
	}

	public String getValue()
	{
		return value;
	}
	public byte getStatus()
	{
		return status;
	}
	public void setDeleteStatus()
	{
		status = 2;
	}
}
```

```
//////////////////////////////////////
public class HashMap
{
	private int tableSize = 128;
	private HashEntry[] table;
	private int currentSize = 0;
```

```
	public HashMap(int size)
	{
		table = new HashEntry[size];
		for (int i = 0; i < size; i++)
			table[i] = null;
		tableSize = size;
		currentSize = 0;
	}

	public void makeEmpty( )
	{
	    for( int i = 0; i < table.length; i++ )
	        table[ i ] = null;

	    currentSize = 0;
	}

	public int getCurrentSize()
	{
	  return currentSize;
	}

	public int getTableSize()
	{
	  return tableSize;
	}

	public boolean contains(String key)
	{
	    return get(key) !=  null;
	}

	public String get(String key)
	{
	  int i = 1;
	    int location = getHash(key);
	    while ((table[location] != null) && (table[location].getStatus()
!= 0))
	    {
	        if (table[location].getKey().equals(key))
	            return table[location].getValue();
	        location = (location + i*i) % tableSize;
	    }
	    return null;
	}

	public void remove(String key)
	{
	  int i = 1;
	    if (!contains(key))
	        return;

	  int hash = getHash(key);
	    while ((table[hash] != null) && (table[hash].getStatus() != 0)
	              && (!table[hash].getKey().equals(key)))
	        hash = (hash + i*i) % tableSize;

	    currentSize--;
	    table[hash].setDeleteStatus();
	}
```

```
      public String find(String key)
      {
        int i =1 ;
          int hash = getHash(key);
          while (((table[hash] != null) && (table[hash].getStatus() != 0)
                    && (!table[hash].getKey().equals(key))) ||
                    ((table[hash].getKey().equals(key)) &&
(table[hash].getStatus() ==2)))
              hash = (hash + i*i) % tableSize;

          if ((table[hash] == null) || (table[hash].getStatus() == 0))
              return null;
          else
              return table[hash].getValue();
      }

      public void insert(String key, String value)
      {
        if ( currentSize >= tableSize / 2)
              rehash();
          int hash = getHash(key);
          int i = 1;
          while ((table[hash] != null) && (table[hash].getStatus() != 0) &&
                    (table[hash].getStatus() != 2))
              hash = (hash + i*i) % tableSize;

          currentSize++;
          table[hash] = new HashEntry(key, value,(byte)1);
      }

      public int getHash( String key)
      {
          int hashVal = 0;

          for( int i = 0; i < key.length( ); i++ )
              hashVal = (2<<5) * hashVal + key.charAt(i);

          hashVal %= tableSize;
          if( hashVal < 0 )
              hashVal += tableSize;

          return hashVal;
      }

      private void rehash( )
      {
        HashMap  newList;

        newList = new HashMap(nextPrime( 2 * table.length ));
          // Copy table over

          for( int i = 0; i < table.length; i++ )
              if ((table[i] != null) && (table[i].getStatus() == 1))
                  newList.insert( table[ i ].getKey(), table[ i
].getValue() );

          table = newList.table;
        tableSize = newList.tableSize;
      }
```

```
    private static int nextPrime( int n )
    {
        if( n % 2 == 0 )
            n++;

        for( ; !isPrime( n ); n += 2 )
            ;

        return n;
    }

    private static boolean isPrime( int n )
    {
        if( n == 2 || n == 3 )
            return true;

        if( n == 1 || n % 2 == 0 )
            return false;

        for( int i = 3; i * i <= n/2; i += 2 )
            if( n % i == 0 )
                return false;

        return true;
    }

    public void printHashTable()
    {
        for( int i = 0; i < table.length; i++ )
            if ((table[ i ] != null)&& (table[i].getStatus() == 1))
                System.out.println(i +" , " +  table[ i ].getKey()+ " ,
"+ table[ i ].getValue());
    }
}
```

////////////////////////////

```
import java.util.Scanner;

public class Lab9_Q1
{
    public static void main(String[] args)
    {
      Scanner scan = new Scanner(System.in);
      System.out.println("Hash Table Test\n\n");
      System.out.println("Enter size");
      int size = scan.nextInt();
      HashMap hashTable = new HashMap(size);
      char ch;
      do
      {
          System.out.println("\nHash Table Operations\n");
          System.out.println("1. insert ");
          System.out.println("2. remove");
          System.out.println("3. get");
          System.out.println("4. clear");
          System.out.println("5. size");

          int choice = scan.nextInt();
          switch (choice)
```

```
            {
            case 1 :
                System.out.println("Enter key and value");
                hashTable.insert(scan.next(), scan.next());
                break;

            case 2 :
                System.out.println("Enter key");
                hashTable.remove(scan.next());
                break;

            case 3 :
                System.out.println("Enter key");
                System.out.println("Value = "+ hashTable.find(
scan.next()));
                break;

            case 4 :
                  hashTable.makeEmpty();
                System.out.println("Hash Table Cleared\n");
                break;

            case 5 :
                System.out.println("Size = "+ hashTable.getTableSize() );
                break;
            default :
                System.out.println("Wrong Entry \n ");
                break;
            }
            /** Display hash table **/
            hashTable.printHashTable();
            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');

    }

}
```

Question 2

Write a program to manage a hash table, using open addressing, where the keys are student full names. Your code should provide create, insert, find and delete operations on that table.

Use a hash function suitable for character strings and motivate your answer in a comment stored in the heading area of your hash table processing routines. Output for find should be yes, followed by the table index if found or no if not found.

Solution Q2

Same as Question #1

Question 3

Write a program to manage a hash table, using open addressing, with numeric long integer keys, where the hash function should be selectable before each run. The methods you should use in building your hash functions are linear, quadratic and double hashing. Your code should provide create, insert, find and delete operations on that table.

Output for find should be yes, followed by the table index if found or no if not found.

Solution Q3

```
public class HashEntry
{
	private long key;
	private String value;
	private byte status;	// insert: 1, delete: 2, empty: 0

	HashEntry(long key, String value, byte status)
	{
		this.key = key;
		this.value = value;
		this.status = status;
	}

	public long getKey()
	{
		return key;
	}

	public String getValue()
	{
		return value;
	}
	public byte getStatus()
	{
		return status;
	}
	public void setDeleteStatus()
	{
		status = 2;
	}
}
//////////////////////////////////////////

public class HashMap
{
	private int tableSize = 128;
	private HashEntry[] table;
	private int currentSize = 0;

	public HashMap(int size)
	{
		table = new HashEntry[size];
		for (int i = 0; i < size; i++)
			table[i] = null;
```

```
            tableSize = size;
            currentSize = 0;
        }

        public void makeEmpty( )
        {
            for( int i = 0; i < table.length; i++ )
                table[ i ] = null;

            currentSize = 0;
        }

        public int getCurrentSize()
        {
          return currentSize;
        }

        public int getTableSize()
        {
          return tableSize;
        }

        public boolean contains(long key)
        {
            return get(key) !=  null;
        }

        public String get(long key)
        {
          int i = 1;
            int location = getHash(key);
            while ((table[location] != null) && (table[location].getStatus()
!= 0))
            {
                if (table[location].getKey() == key)
                    return table[location].getValue();
                location = (location + i*i) % tableSize;             //
Quadratic
             // location = (location + 1) % tableSize;         // Linear
             // location = (location + i * F2) % tableSize;    // Double
            }
            return null;
        }

        public void remove(long key)
        {
          int i = 1;
            if (!contains(key))
                return;

          int hash = getHash(key);
            while ((table[hash] != null) && (table[hash].getStatus() != 0)
                      && (table[hash].getKey() != key))
                hash = (hash + i*i) % tableSize;         // Quadratic
             // hash = (hash + 1) % tableSize;           // Linear
             // hash = (hash + i * F2) % tableSize;      // Double
            currentSize--;
            table[hash].setDeleteStatus();
        }

        public String find(long key)
```

```
      {
        int i =1 ;
          int hash = getHash(key);
          while (((table[hash] != null) && (table[hash].getStatus() != 0)
                    && (table[hash].getKey() != key)) ||
                    ((table[hash].getKey() == key) &&
(table[hash].getStatus() ==2)))
              hash = (hash + i*i) % tableSize;        // Quadratic
           // hash = (hash + 1) % tableSize;          // Linear
           // hash = (hash + i * F2) % tableSize;     // Double

          if ((table[hash] == null) || (table[hash].getStatus() == 0))
              return null;
          else
              return table[hash].getValue();
      }

      public void insert(long key, String value)
      {
        if ( currentSize >= tableSize / 2)
              rehash();
          int hash = getHash(key);
          int i = 1;
          while ((table[hash] != null) && (table[hash].getStatus() != 0) &&
                    (table[hash].getStatus() != 2))
              hash = (hash + i*i) % tableSize;        // Quadratic
           // hash = (hash + 1) % tableSize;          // Linear
           // hash = (hash + i * F2) % tableSize;     // Double

          currentSize++;
          table[hash] = new HashEntry(key, value,(byte)1);
      }

      public int getHash( long key)
      {
          int hashVal = 0;
          hashVal = (int)key % tableSize;
          if( hashVal < 0 )
              hashVal += tableSize;

          return hashVal;
      }

      private void rehash( )
      {
        HashMap  newList;

        newList = new HashMap(nextPrime( 2 * table.length ));
          // Copy table over

          for( int i = 0; i < table.length; i++ )
              if ((table[i] != null) && (table[i].getStatus() == 1))
                  newList.insert( table[ i ].getKey(), table[ i
].getValue() );

          table = newList.table;
        tableSize = newList.tableSize;
      }

      private static int nextPrime( int n )
      {
```

```
            if( n % 2 == 0 )
                n++;

            for( ; !isPrime( n ); n += 2 )
                ;

            return n;
        }

        private static boolean isPrime( int n )
        {
            if( n == 2 || n == 3 )
                return true;

            if( n == 1 || n % 2 == 0 )
                return false;

            for( int i = 3; i * i <= n/2; i += 2 )
                if( n % i == 0 )
                    return false;

            return true;
        }

        public void printHashTable()
        {
            for( int i = 0; i < table.length; i++ )
                if ((table[ i ] != null)&& (table[i].getStatus() == 1))
                    System.out.println(i +" , " +  table[ i ].getKey()+ " ,
"+ table[ i ].getValue());
        }
}
//////////////////////////////////////////////

import java.util.Scanner;

public class Lab9_Q3
{
        public static void main(String[] args)
        {
          Scanner scan = new Scanner(System.in);
          System.out.println("Hash Table Test\n\n");
          System.out.println("Enter size");
          int size = scan.nextInt();
          HashMap hashTable = new HashMap(size);
          char ch;
          do
          {
              System.out.println("\nHash Table Operations\n");
              System.out.println("1. insert ");
              System.out.println("2. remove");
              System.out.println("3. get");
              System.out.println("4. clear");
              System.out.println("5. size");

              int choice = scan.nextInt();
              switch (choice)
              {
              case 1 :
                  System.out.println("Enter key and value");
                  hashTable.insert(scan.nextLong(), scan.next());
```

```
                break;

            case 2 :
                System.out.println("Enter key");
                hashTable.remove(scan.nextLong());
                break;

            case 3 :
                System.out.println("Enter key");
                System.out.println("Value = "+ hashTable.find(
scan.nextLong()));
                break;

            case 4 :
                  hashTable.makeEmpty();
                System.out.println("Hash Table Cleared\n");
                break;

            case 5 :
                System.out.println("Size = "+ hashTable.getTableSize() );
                break;
            default :
                System.out.println("Wrong Entry \n ");
                break;
            }
            /** Display hash table **/
            hashTable.printHashTable();
            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');

      }

}
```

Question 4

Write a program to manage a hash table, using open addressing, with character string keys, where the hash function should be selectable before each run. The methods you should use in building your hash functions are linear, quadratic and double hashing. Your code should provide create, insert, find and delete operations on that table.
Output for find should be yes, followed by the table index if found or no if not found.

Solution Q4

Same as Question #1

Lab 10: Heaps

Exercises:

Question 1

Step 1: Implement the operations in Heap ADT using an array representation of a heap. Heaps can be different sizes; therefore you need to store the actual number of elements in the heap (size), along with the heap elements themselves (element). Remember that in Java the size of the array is held in a constant called length in the array object. Therefore, in Java a separate variable (such as *maxSize*) is not necessary, since the maximum number of elements our heap can hold can be determined by referencing length - more specifically in our case, *element.length.*

You are to fill in the Java code for each of the constructors and methods where only the method headers are given. Each method header appears on a line by itself and does not contain a semicolon. This is not an interface file, so a semicolon should not appear at the end of a method header. Each of these methods needs to be fully implemented by writing the body of code for implementing that particular method and enclosing the body of that method in braces.

```
public class Heap
{
        // Constant
        private static final int DEF_MAX_HEAP_SIZE = 10; // Default maximum heap size
        // Data members
        private int size;                   // Actual number of elements in the heap
        private HeapData [ ] element; // Array containing the heap elements
        // ___The following are Method Headers ONLY ___ //
        // each of these methods needs to be fully implemented
        // Constructors and helper method setup
        public Heap ( )                     // Constructor: default size
        public Heap ( int maxNumber ) // Constructor: specific size
        // Class methods
        private void setup ( int maxNumber ) // Called by constructors only
```

```
        // Heap manipulation methods
        public void insert ( HeapData newElement ) // Insert element
        public HeapData removeMax ( ) // Remove max pty element
        public void clear ( ) // Clear heap
        // Heap status methods
        public boolean isEmpty ( )      // Heap is empty
        public boolean isFull ( )       // Heap is full
        // Output the heap structure - used in testing/debugging
        public void showStructure ( )
        // Recursive partner of the showStructure() method
        private void showSubtree ( int index, int level )
}       // class Heap
```

Step 2: Save your implementation of the Heap ADT in the file Heap.java. Be sure to document your code.

Solution Q1

```
public class Heap
{
      // Constant
      private static final int DEF_MAX_HEAP_SIZE = 10; // Default maximum
heap size
    private static final int FRONT = 1;
      // Data members
      private int size;              // Actual number of elements in the
heap
      private int [ ] element; // Array containing the heap elements

      public Heap ( )          // Constructor: default size
      {
            setup(DEF_MAX_HEAP_SIZE);
      }

      public Heap ( int maxNumber ) // Constructor: specific size
      {
            setup(maxNumber);
      }

      // Class methods
      private void setup ( int maxNumber ) // Called by constructors only
      {
        size = 0;
        element = new int[maxNumber + 1];
        element[0] = Integer.MAX_VALUE;
      }
```

```
    private int parent(int pos)
    {
        return pos / 2;
    }

    private int leftChild(int pos)
    {
        return (2 * pos);
    }

    private int rightChild(int pos)
    {
        return (2 * pos) + 1;
    }

    private boolean isLeaf(int pos)
    {
        if (pos >=  (size / 2)  &&  pos <= size)
        {
            return true;
        }
        return false;
    }

    private void swap(int fpos,int spos)
    {
        int tmp;
        tmp = element[fpos];
        element[fpos] = element[spos];
        element[spos] = tmp;
    }

    private void maxHeapify(int pos)
    {
        if (!isLeaf(pos))
        {
            if ( element[pos] < element[leftChild(pos)]  || element[pos] <
element[rightChild(pos)])
            {
                if (element[leftChild(pos)] > element[rightChild(pos)])
                {
                    swap(pos, leftChild(pos));
                    maxHeapify(leftChild(pos));
                }
                else
                {
                    swap(pos, rightChild(pos));
                    maxHeapify(rightChild(pos));
                }
            }
        }
    }

    public void maxHeap()
    {
        for (int pos = (size / 2); pos >= 1; pos--)
        {
            maxHeapify(pos);
        }
    }
```

```
	// Heap manipulation methods
	public void insert (int newElement ) // Insert element
	{
		element[++size] = newElement;
	  int current = size;

	  while(element[current] > element[parent(current)])
	  {
		swap(current,parent(current));
		current = parent(current);
	  }
	}

	public int removeMax ( ) // Remove max pty element
	{
	  int popped = element[FRONT];
	  element[FRONT] = element[size--];
	  maxHeapify(FRONT);
	  return popped;
	}

	public void clear ( ) // Clear heap
	{
		size = 0;
	}

	static int log(int x, int base)
	{
	    return (int) (Math.log(x) / Math.log(base));
	}

	public int getLevel() // return number of level
	{
		int level = (int) log(size,2);
		if (size == (int )Math.pow(2,level))
			return level;
		else
			return level +1;
	}

	// Heap status methods
	public boolean isEmpty ( )    // Heap is empty
	{
		return size == 0;
	}

	public boolean isFull ()      // Heap is full
	{
		return size == element.length;
	}

	// Output the heap structure - used in testing/debugging
	public void showStructure ( )
	{
	  for (int i = 1; i <= size / 2; i++ )
	  {
		System.out.print(" PARENT : " + element[i] + " LEFT CHILD : " +
element[2*i]
			+ " RIGHT CHILD :" + element[2 * i  + 1]);
		System.out.println();
	  }
```

```
        }

        // Recursive partner of the showStructure() method
        private void showSubtree(int index, int level)
        {
        }

}


//////////////////////////////////////


public class Lab10Q1
{
    public static void main(String...arg)
    {
        System.out.println("The Max Heap is ");
        Heap maxHeap = new Heap(15);
        maxHeap.insert(5);
        maxHeap.insert(3);
        maxHeap.insert(17);
        maxHeap.insert(10);
        maxHeap.insert(84);
        maxHeap.insert(19);
        maxHeap.insert(6);
        maxHeap.insert(22);
        maxHeap.insert(9);
        maxHeap.maxHeap();
        maxHeap.showStructure();
     //   System.out.println("The max val is " + maxHeap.removeMax());
        System.out.println();
      //  maxHeap.showSubtree(1, 40);
    }
}
```

Question 2

After removing the root element, the *removeMax* operation inserts a new element at the root and moves this element downward until a heap is produced. The following method performs a similar task, except that the heap it is building is rooted at array entry root and occupies only a portion of the array.

void moveDown (HeapData [ ] element, int root, int size)

Precondition:

The left and right subtrees of the binary tree rooted at root are heaps.

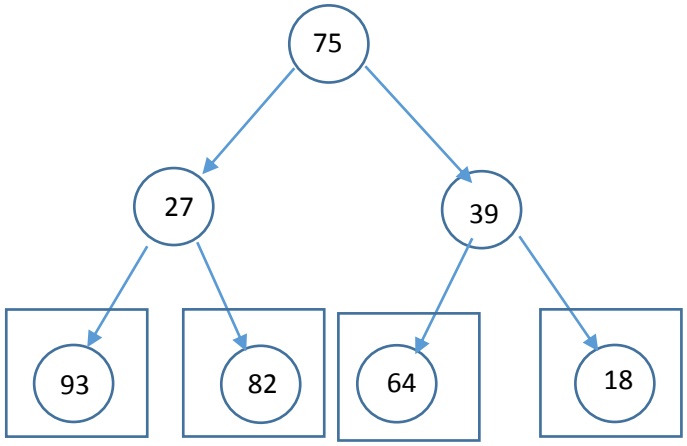
Post condition:

Restores the binary tree rooted at root to a heap by moving element [root] downward until the tree satisfies the heap property. Parameter size is the number of elements in the array.

Remember that repeatedly swapping array elements is not an efficient method for restoring the heap.

In this exercise, you implement an efficient sorting algorithm called heap sort using the *moveDown( )* method. You first use this method to transform an array into a heap. You then remove elements one-by-one from the heap (from the highest priority element to the lowest) until you produce a sorted array.

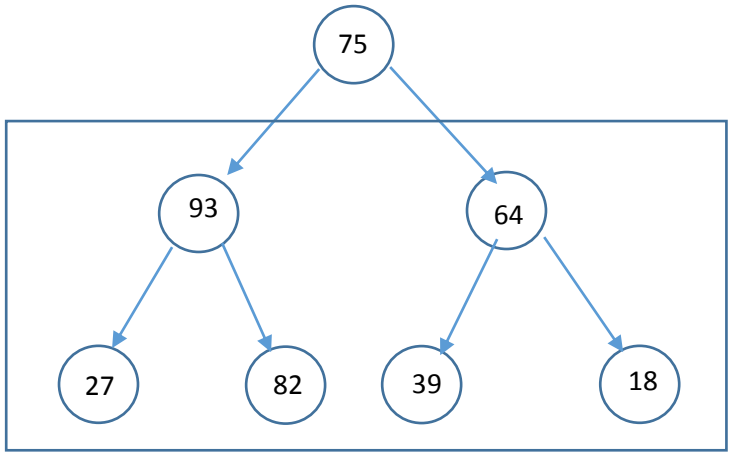
Let's begin by examining how you transform an unsorted array into a heap. Each leaf of any binary tree is a one-element heap. You can build a heap containing three elements from a pair of sibling leaves by applying the *moveDown( )* method to that pair's parent. The four *singleelement* heaps (leaf nodes) in the following tree



Index	Entry
0	75
1	27
2	39

3	93
4	82
5	64
6	18

Are transformed by the calls *moveDown*(sample, 1, 7) and *moveDown*(sample, 2, 7) into a pair of three-element heaps:

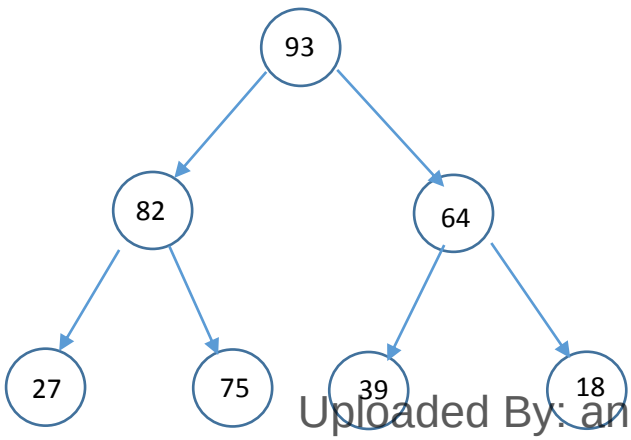


Index	Entry
0	75
1	93
2	64
3	27
4	82
5	39
6	18

By repeating this process, you build larger and larger heaps, until you transform the entire tree (array) into a heap.

```
// Build successively larger heaps within the array until the
// entire array is a heap.
for ( j = (size - 1) / 2; j >= 0; j-- )
        moveDown( element, j, size );
```

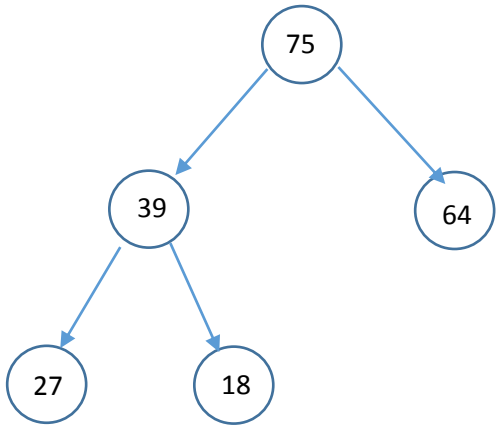
Combining the pair of three-element heaps shown previously using the call moveDown(sample, 0, 7), for instance, produces the following heap.



Index	Entry
0	93
1	82
2	64
3	27
4	75
5	39
6	18

Now that you have a heap, you remove elements of decreasing priority from the heap and gradually construct an array that is sorted in ascending order. The root of the heap contains the highest priority element. If you swap the root with the element at the end of the array and use moveDown( ) to form a new heap, you end up with a heap containing six elements and a sorted array containing one element. Performing this process a second time yields a heap containing five elements and a sorted array containing two elements.

Heap



Sorted array

Index	Entry
0	75
1	39
2	64
3	27
4	18
5	82

6	93

You repeat this process until the heap is gone and a sorted array remains.

```
// Swap the root element from each successively smaller heap with
// the last unsorted element in the array. Restore the heap after
// each exchange.
for ( j = size - 1; j > 0; j-- )
{
        temp = element[ j ];
        element[ j ] = element[ 0 ];
        element[ 0 ] = temp;
        moveDown( element, 0, j );
}
```

A shell containing a heapSort( ) method comprised of the two loops shown above is given in the file TestHeapSort.jshl.

Step 1: Using your implementation of the removeMax operation as a basis, create an implementation of the moveDown( ) method.

Step 2: Before testing the resulting heapSort( ) method using the test program in the file TestHeapSort.java, prepare a test plan for the heapSort( ) method that covers arrays of different lengths containing a variety of priority values. Be sure to include arrays that have multiple elements with the same priority. A test plan form follows.

Solution Q2

```
public class HeapSort
{
      // Constant
    private static final int FRONT = 1;
      // Data members
      private int size;              // Actual number of elements in the
heap
      private int [ ] element; // Array containing the heap elements

      public HeapSort(int [] element)           // Constructor: default
size
      {
        size = element.length;
```

```
        this.element = new int[element.length+1];

        for (int i=1; i<= element.length; i++)
            this.element[i] = element[i-1];
      }

    private int parent(int pos)
    {
        return pos / 2;
    }

    private int leftChild(int pos)
    {
        return (2 * pos);
    }

    private int rightChild(int pos)
    {
        return (2 * pos) + 1;
    }

    private boolean isLeaf(int pos)
    {
        if (pos >=  (size / 2)  &&  pos <= size)
        {
            return true;
        }
        return false;
    }

    private void swap(int fpos,int spos)
    {
        int tmp;
        tmp = element[fpos];
        element[fpos] = element[spos];
        element[spos] = tmp;
    }

    private void maxHeapify(int pos)
    {
      int left = leftChild(pos);
      int right = rightChild(pos);
      int max = pos;
      if ((left <= size) && (element[pos] < element[left]))
            max = left;
      if ((right <= size) && (element[max] < element[right]))
            max = right;
      if ( max != pos)
      {
            swap(pos, max);
            maxHeapify(max);
      }
    }

    public void maxHeap()
    {
        for (int pos = (size / 2); pos >= 1; pos--)
        {
            maxHeapify(pos);
        }
    }
```

```
    public void sort()
    {
      maxHeap();
      for(int i= size; i> 1 ; i--)
      {
            swap(FRONT, size);
            size--;
            maxHeapify(FRONT);
      }
    }

    public int [] getSortedData()
    {
      return element;
    }

}
//////////////////////////////////////

public class Lab10Q2
{
    public static void main(String []arg)
    {
      int element[] = {5, 3, 17, 10, 84, 19, 6, 22, 9};
        System.out.println("The Max Heap is ");
        HeapSort maxHeap = new HeapSort(element);
        maxHeap.sort();
        element = maxHeap.getSortedData();
        for(int i = 0; i< element.length; i++)
            System.out.println(element[i]);
    }
}
```

Question 3

A priority queue is a linear data structure in which the elements are maintained in descending order based on priority. You can only access the element at the front of the queue-that is, the element with the highest priority-and examining this element entails removing (dequeuing) it from the queue.

You can easily and efficiently implement a priority queue as a heap by using the Heap ADT insert operation to enqueue elements and the removeMax operation to dequeue elements. The following incomplete definitions derive a class called PtyQueue from the Heap class. In Java the keyword extends is used to specify inheritance (class PtyQueue extends Heap means PtyQueue inherits from Heap). Thus, PtyQueue is the subclass and Heap is the superclass. The subclass inherits all of the public and protected instance variables and methods defined by the superclass and adds its own, unique elements as needed.

```
class PtyQueue extends Heap
{
        // Constructor
        public PtyQueue ( ) // Constructor: default size
        { }
        public PtyQueue ( int size ) // Constructor: specific size
        { }
        // Queue manipulation methods
        public void enqueue ( HeapData newElement ) // Enqueue element
        { }
        public HeapData dequeue ( ) // Dequeue element
        { }
} // class PtyQueue
```

Implementations of the Priority Queue ADT constructor, enqueue, and dequeue operations are given in the file PtyQueue.java. These implementations are very short, reflecting the close relationship between the Heap ADT and the Priority Queue ADT. Note that you inherit the remaining operations in the Priority Queue ADT from the Heap class. You may use the file TestPtyQueue.java to test the Priority Queue implementation.

Operating systems commonly use priority queues to regulate access to system resources such as printers, memory, disks, software, and so forth. Each time a task requests access to a system resource, the task is placed on the priority queue associated with that resource. When the task is dequeued, it is granted access to the resource to print, store data, and so on.

Suppose you wish to model the flow of tasks through a priority queue having the following properties:

- One task is dequeued every minute (assuming that there is at least one task waiting to be dequeued during that minute).
- From zero to two tasks are enqueued every minute, where there is a 50% chance that no tasks are enqueued, a 25% percent chance that one task is enqueued, and a 25% chance that two tasks are enqueued.
- Each task has a priority value of zero (low) or one (high), where there is an equal chance of a task having either of these values.

You can simulate the flow of tasks through the queue during a time period n minutes long using the following algorithm.

```
Initialize the queue to empty.
for ( minute = 0 ; minute < n ; ++minute )
{
        If the queue is not empty, then remove the task at the front of the queue.
        Compute a random integer k between 0 and 3.
        If k is 1, then add one task to the queue. If k is 2, then add two tasks.
        Otherwise (if k is 0 or 3), do not add any tasks to the queue.
        Compute the priority of each task by generating a random value of 0 or 1
        (assuming here are only 2 priority levels).
}
```

Solution Q3

Lab 11: Sorting 1

Exercises:

Question 1

Running an actual program, count the number of compares needed to sort n values using insertion sort, where n varies (e.g., powers of 2).

Solution Q1

```
public class InsertionSort
{
	public static int sort(Comparable [] data)
	{
		int compares = 0;
		for (int i = 1; i < data.length; ++i)
		{
			Comparable temp = data[i];
			int pos = i;
			// Shuffle up all sorted items > data[i]
			while (pos > 0 &&
			  data[pos-1].compareTo(temp) > 0)
			{
				data[pos] = data[pos-1];
				pos--;
				++compares ;
			} // end while
			// Insert the current item
			data[pos] = temp;
		}
		return compares;
	}
}

 ////////////////////////////////////////////////////////////

import java.lang.Integer;

public class Lab11Q1
{
	public static void main(String[] args)
	{
		// test for integers
		Integer [] arrInt;
		arrInt = new Integer [50];
		int compares = 0;
		for (int i = 0; i< arrInt.length ; i++)
		{
			arrInt[i] = new Integer((int )(Math.random() * 50 + 1));

		}

		compares = InsertionSort.sort(arrInt);
```

```
		System.out.println("Sorted Data: ");
		for (int i = 0; i< arrInt.length  ; i++)
		{
			System.out.println(arrInt[i]);
		}

		System.out.println("# of Compares = " + compares);

		///////////////

		// test for Strings
		String [] arrSt;
		arrSt = new String [10];
		arrSt[0] = "Graphics";
		arrSt[1] = "Computer Science";
		arrSt[2] = "Book";
		arrSt[3] = "Algorithm";
		arrSt[4] = "Data Structure";
		arrSt[5] = "Introduction";
		arrSt[6] = "Computer Lab";
		arrSt[7] = "Compiler";
		arrSt[8] = "Operating System";
		arrSt[9] = "Database";

		compares = InsertionSort.sort(arrSt);
		System.out.println();
		System.out.println("Sorted Data: ");
		for (int i = 0; i< arrSt.length ; i++)
		{
			System.out.println(arrSt[i]);
		}

		System.out.println("# of Compares = " + compares);
	}
}
```

Question 2

Running an actual program, and using the millisecond timer, System.currentTimeMillis, measure the length of time needed to sort arrays of data of various sizes using a sort of your choice.

Solution Q2

```
public class RadixSort
{
    public static void sort( int[] a)
    {
        int i, m = a[0], exp = 1, n = a.length;
        int[] b = new int[n];
        for (i = 1; i < n; i++)
            if (a[i] > m)
                m = a[i];

        while (m / exp > 0)
        {
            int[] bucket = new int[10];

            for (i = 0; i < n; i++)
                bucket[(a[i] / exp) % 10]++;

            for (i = 1; i < 10; i++)
                bucket[i] += bucket[i - 1];

            for (i = n - 1; i >= 0; i--)
                b[--bucket[(a[i] / exp) % 10]] = a[i];

            for (i = 0; i < n; i++)
                a[i] = b[i];

            exp *= 10;
        }
    }
}

////////////////////////////////////
public class Lab11Q2
{
      public static void main(String[] args)
      {
            // test for integers
            int [] arrInt;
            arrInt = new int [5000];
            for (int i = 0; i< arrInt.length ; i++)
            {
                  arrInt[i] = (int )(Math.random() * 5000 + 1);
            }

            long timeBefore = System.currentTimeMillis();

            RadixSort.sort(arrInt);

            long timeAfter = System.currentTimeMillis();
```

```
            System.out.println("Sorting Time in milliseconds = " +
                               (timeAfter - timeBefore));

            System.out.println("Sorted Data = ");
            for (int i = 0; i< arrInt.length  ; i++)
            {
                  System.out.println(arrInt[i]);
            }
      }
}
```

Question 3

Write a Java program for sorting a given list of names in ascending order.

Solution Q3

```
public class MergeSort
{
	public static void sort(Comparable [] data, int low, int high)
	{
		int mid = (low + high)/2;
		if (low < high)
		{
			System.out.println(low+"  "+mid+"   "+high);
			sort(data, low, mid);
			sort(data, mid+1, high);
			merge(data, low, mid, high);
		}
	}

	public static void merge(Comparable [] data,int low,int mid,int high)
	{
		int n = high - low+1;
		Comparable[] temp = new Comparable[n];
		int i = low, j = mid+1, k=0;

		while ((i <= mid) && (j <= high))
		{
			if (data[j].compareTo(data[i]) <= 0)
				temp[k++] = data[j++];
			else
				temp[k++] = data[i++];
		}

		if (i <= mid)
			for (; i<= mid;)
				temp[k++] = data[i++];
		else
			for (; j<= high; )
				temp[k++] = data[j++];

		for (k = 0; k < n; k++)
			data[low + k] = temp[k];
	}
}

//////////////////////////////////////////
```

```
public class Lab11Q3
{
	public static void main(String[] args)
	{
		String [] arrSt;
		arrSt = new String [10];
		arrSt[0] = "Graphics";
		arrSt[1] = "Computer Science";
		arrSt[2] = "Book";
		arrSt[3] = "Algorithm";
		arrSt[4] = "Data Structure";
		arrSt[5] = "Introduction";
		arrSt[6] = "Computer Lab";
		arrSt[7] = "Compiler";
		arrSt[8] = "Operating System";
		arrSt[9] = "Database";

		MergeSort.sort(arrSt, 0, arrSt.length-1);

		System.out.println("Sorted Data = ");
		for (int i = 0; i< arrSt.length ; i++)
			System.out.println(arrSt[i]);
	}
}
```

Lab 12: Sorting 2

Exercises:

Question 1

Modify the partition method used by quicksort so that the pivot is randomly selected.

Solution Q1

```
import java.util.Random;

public class QuickSort
{
	public static void sort(Comparable [] data)
	{
		quickSort(data, 0, data.length - 1);
	}

	public static void quickSort(Comparable [] data, int low, int high)
	{
	  int i, j, pivot;
	  Comparable temp;
	  Random rand = new Random();
	  /** partition **/
	  if (low < high)
	  {
	      i = low;
	      j = high-1;
	      //pivot = low;
	      pivot = low + rand.nextInt(high-low+1);
	      swap(data, pivot, high);
	      pivot = high;
	      while (i <= j)
	      {
			while (data[i].compareTo(data[pivot]) <0)
				i++;
			while (data[j].compareTo(data[pivot]) > 0)
				j--;
			if (i < j)
				swap(data,i,j);
	      }
	      swap(data, i, high);
	      quickSort(data, low,i-1);
	      quickSort(data, i+1, high);
	  }
	}

	public static void swap(Comparable [] data, int i, int j)
	{
		Comparable temp;
	  temp = data[i];
	  data[i] = data[j];
	  data[j] = temp;
	}
}
//////////////////////////////////////////
```

```
public class Lab12Q1
{
	public static void main(String[] args)
	{
		String [] arrSt;
		arrSt = new String [10];
		arrSt[0] = "Graphics";
		arrSt[1] = "Computer Science";
		arrSt[2] = "Book";
		arrSt[3] = "Algorithm";
		arrSt[4] = "Data Structure";
		arrSt[5] = "Introduction";
		arrSt[6] = "Computer Lab";
		arrSt[7] = "Compiler";
		arrSt[8] = "Operating System";
		arrSt[9] = "Database";

		QuickSort.sort(arrSt);

		System.out.println("Sorted Data = ");
		for (int i = 0; i< arrSt.length ; i++)
			System.out.println(arrSt[i]);
	}
}
```

Question 2

Generate a 100-element list containing integers in the range 0-10. Sort the list with shell sort and with quick sort.

1. Which is faster? Why?
2. Try this again, but with 100,000 elements. Note the relative difference> Why does it exists?

Solution Q2

```
public class QuickSort
{
	public static void sort(int [] data)
	{
		quickSort(data, 0, data.length - 1);
	}

	public static void quickSort(int [] data, int low, int high)
	{
	  int i, j, pivot;

	  if (low < high)
	  {
	      i = low;
	      j = high-1;
	      pivot = getPivot(data, low, high);

	      while (i < j)
	      {
			while (data[++i] < pivot)
				;
			while (data[--j] > pivot)
				;
			if (i < j)
				swap(data,i,j);
	      }
	      swap(data, i, high-1);
	      quickSort(data, low,i-1);
	      quickSort(data, i+1, high);
	  }
    }

	public static void swap(int [] data, int i, int j)
	{
		int temp;
	  temp = data[i];
	  data[i] = data[j];
	  data[j] = temp;
	}

	public static int getPivot(int [] data, int low, int high)
	{
		int mid = (low + high) /2;

		if (data[low] > data[mid])
			swap(data, low, mid);
		if (data[low] > data[high])
			swap(data, low, high);
```

```
            if (data[mid] > data[high])
                  swap(data, mid, high);

            swap(data, mid, high-1);

            return data[high-1];
      }
}
////////////////////////////////////

public class ShellSort
{
      public static void sort(int [] data)
      {
             int i, j, increment;
             int temp;
             for(increment = data.length/2; increment > 0; increment /=2)
                   for( i = increment; i < data.length ; i++)
                   {
                         temp = data[i];
                         for ( j= i; j>= increment; j-=increment)
                               if ( data[j-increment] > temp)
                                     data[j] = data[j-increment];
                               else
                                     break;
                         data[j] = temp;
                   }
      }
}
////////////////////////////////////////////

public class Lab12Q2
{
      public static void main(String[] args)
      {
            int [] arrInt;
            // A) Test for 10 integer
/*          arrInt = new int [10];

            for (int i = 0; i< arrInt.length; i++)
                  arrInt[i] = (int )(Math.random() * 100 );

            QuickSort.sort(arrInt);

            System.out.println("Sorted Data (Quick Sort = ");
            for (int i = 0; i< arrInt.length ; i++)
                  System.out.println(arrInt[i]);


            for (int i = 0; i< arrInt.length; i++)
                  arrInt[i] = (int )(Math.random() * 100 );

            ShellSort.sort(arrInt);

            System.out.println("Sorted data (Shell Sort) : ");
            for (int i = 0; i< arrInt.length ; i++)
                  System.out.println(arrInt[i]);      */

            // B) Test for 100,000 integer
            arrInt = new int [100000];

            for (int i = 0; i< arrInt.length; i++)
```

```
                arrInt[i] = (int )(Math.random() * 100000 );

            long timeBefore = System.currentTimeMillis();

            QuickSort.sort(arrInt);

            long timeAfter = System.currentTimeMillis();
            System.out.println("Quick sort Time in milliseconds = " +
                               (timeAfter - timeBefore));

            for (int i = 0; i< arrInt.length; i++)
                arrInt[i] = (int )(Math.random() * 100000 );

            timeBefore = System.currentTimeMillis();

            ShellSort.sort(arrInt);

            timeAfter = System.currentTimeMillis();
            System.out.println("Shell sort Time in milliseconds = " +
                               (timeAfter - timeBefore));
      }
}
```