Linked list implementation

```
 /*boolean remove(int index)
   * Removes the element at the specified position in the list*/
public boolean remove(int index){
   if (Size==0)return false;//empty linked list
   else if (index==0)return removeFirst(); //remove first element
   else if (index==size-1)return removeLast();//remove last element
   else if (index >0 && index<Size-1){
        Node current=Front;
        for (int i=0;i<index-1;i++)
            current=current.next;
        current.next= current.next.next;
        Size--;
        return true;
}
  else return false; // out of boundary(invalid index)

}
```

Linked list implementation

```
 /*object remove(int index)
   * Removes the element at the specified position in the list*/
public object remove(int index){

                    Write the code here

}
```

Linked list implementation

```
 /*Print linked list == Traversing linked list  recursively*/
public void traverse (Node current){

    if (current!=null){
     System.out.println(current.element);
     traverse (current.next)

}
```

Linked list implementation

```
/** Remove the first node and
   *  return the object that is contained in the removed node. */

public Object removeFirst() {




            // Implementation left as an exercise



}
```

Linked list implementation

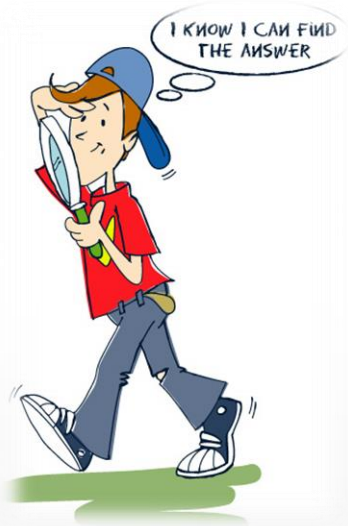
```
/** Remove the last node and
  * return the object that is contained in the removed node. */
 public Object removeLast() {

     // Implementation left as an exercise

     return null;
 }
```

Linked list implementation: H.W

You have one week to do the following:



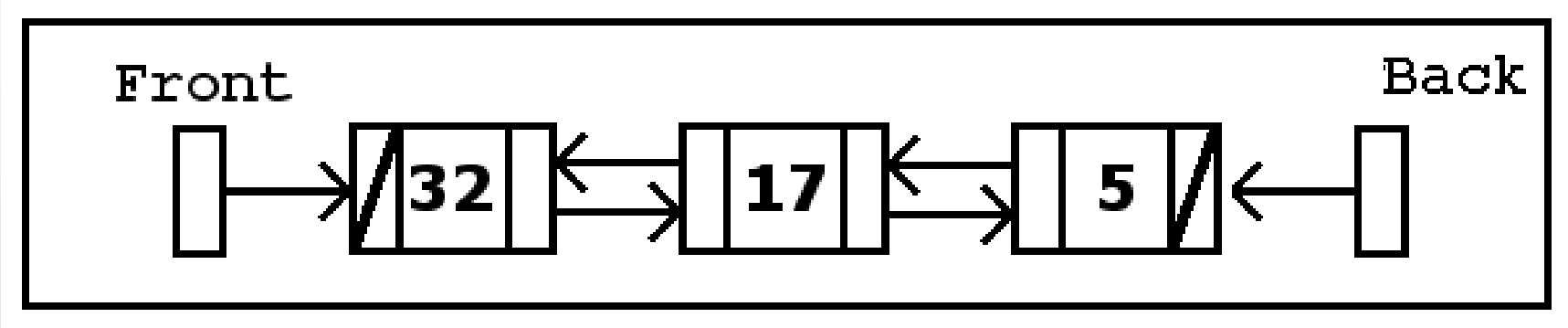
- **/* void clear() Removes all of the elements from the list.*/**
- **int find (Object o); /* return the first index for the specified element in the list*/**
- **/*boolean remove(Object o) Removes the first occurrence of the specified element in the list*/.**

Hint: use the implemented `int find (Object o);`

use the implemented `boolean remove(int index);` Removes the element at the specified position in the list

Double-linked list

- Add a **`prev`** pointer to our **`Node` class**
- Allows backward iteration
- some methods need to be modified
  - when adding or removing a node, we must fix the **`prev`** and **`next`** pointers to have the correct value!
  - can make it easier to implement some methods such as `remove`



Double-linked list

```
/* Stores one element of a linked list. */
Public  class Node
{  public Object element;
   public Node prev, next;

   public Node(Object element) {
         this(element, null, null);
     }

   public Node(Object element, Node prev, Node next){
            this.element = element;
            this.prev = prev;
            this.next = next;
  }
}
```

Double-linked list

```
/* Models an entire linked list. */

 public class DoubleLinkedList {
   private Node Front, Back;
   private int Size;
 public DoublyLinkedList() {

     Front = null;
     Back =  null;
     Size = 0;
  }
}
```

Double-linked list

```
/* void addFirst(Object o)
 * Inserts the given element at the beginning of the list. */
public void addFirst(Object element){
    Node newNode;
    newNode= new Node (element);
    if (Size==0)
      Front=Back=newNode;
    else {
       newNode.next=Front;
       Front.prev=newNode;
       Front=newNode;
   }
   Size++;

}
```

Double-linked list : H.W

You have one week to do the following:

- **/*boolean removeLast()**
  Removes the last element from the list.*/

- **/* void add(int index, Object element) Inserts the specified element at the**
  *** specified position index in this list.**
  ***/**

Double-linked list : H.W

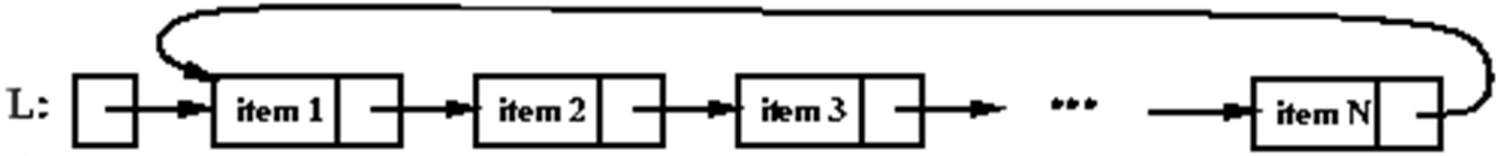
You have one week to do the following:

- **/*boolean removeLast()**
  Removes the last element from the list.*/

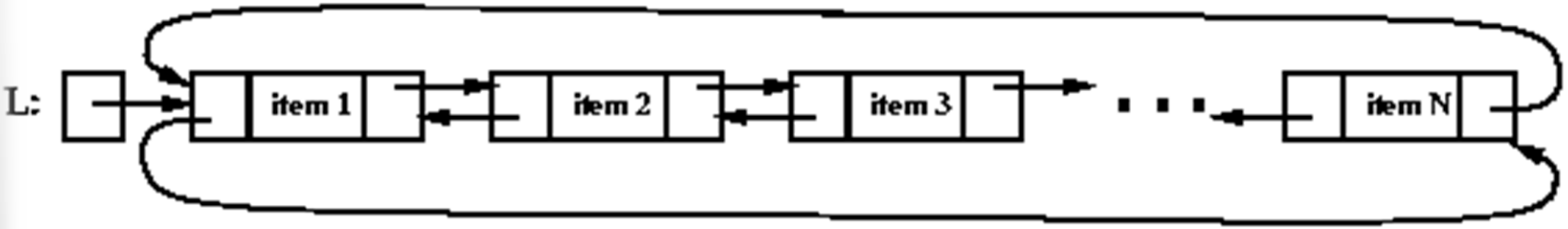
- **/* void add(int index, Object element) Inserts the specified element at the**
  *** specified position index in this list.**
  ***/**

Circular linked lists

Circular, singly linked list:



Circular, doubly linked list:



Extra Exercises

Implement all the non-implemented methods for :

- **Linked List**
- **Double Linked List**

Examples:

boolean contains(Object o) //Returns true if the list contains the specified element.

int lastIndexOf(Object o) //Returns the index in the list of the last occurrence of the specified element, or -1 if the list does not contain this element.

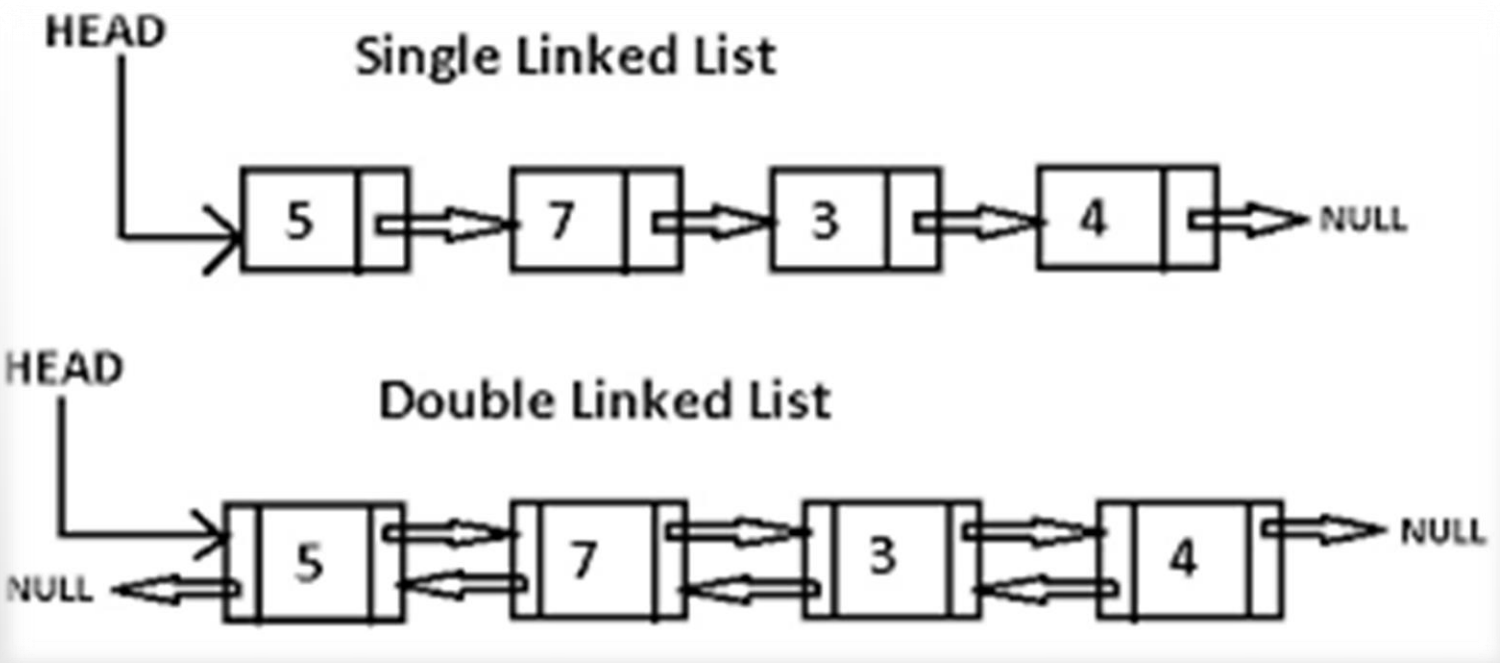
void printList (); // print all the list element

void printList in revers(); // print the list elements in reverse order

……

Extra Exercises

Write the Node Class for the Circular Linked-List (single and double list)



Quiz : In Coming Lecture

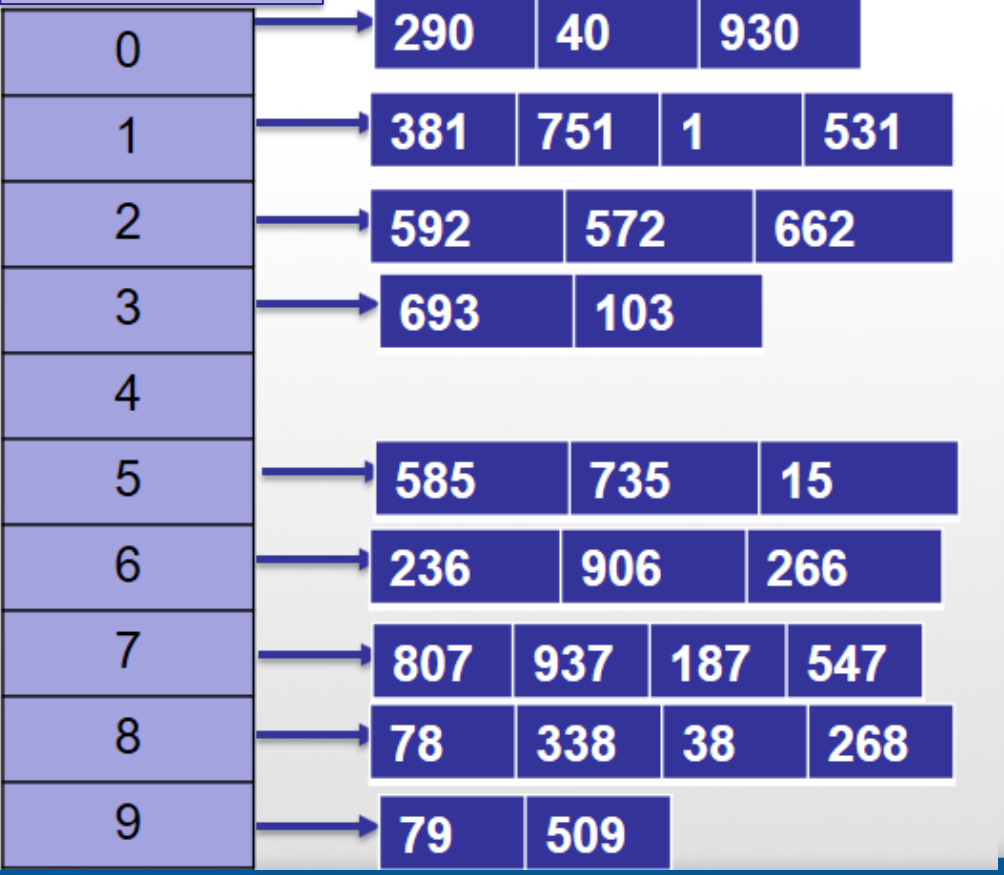
Study the following:

1. **Recursion (coding, and tracing)**
2. **Linked List (single, double, and circular)**

Case Study: Radix Sort Analysis

592, 585, 807, 735, 78, 290, 381, 751, 236, 937, 79, 338, 572, 1, 15, 693, 38, 509, 187, 906, 662, 40, 103, 268, 930, 266, 531,547

After 1st pass

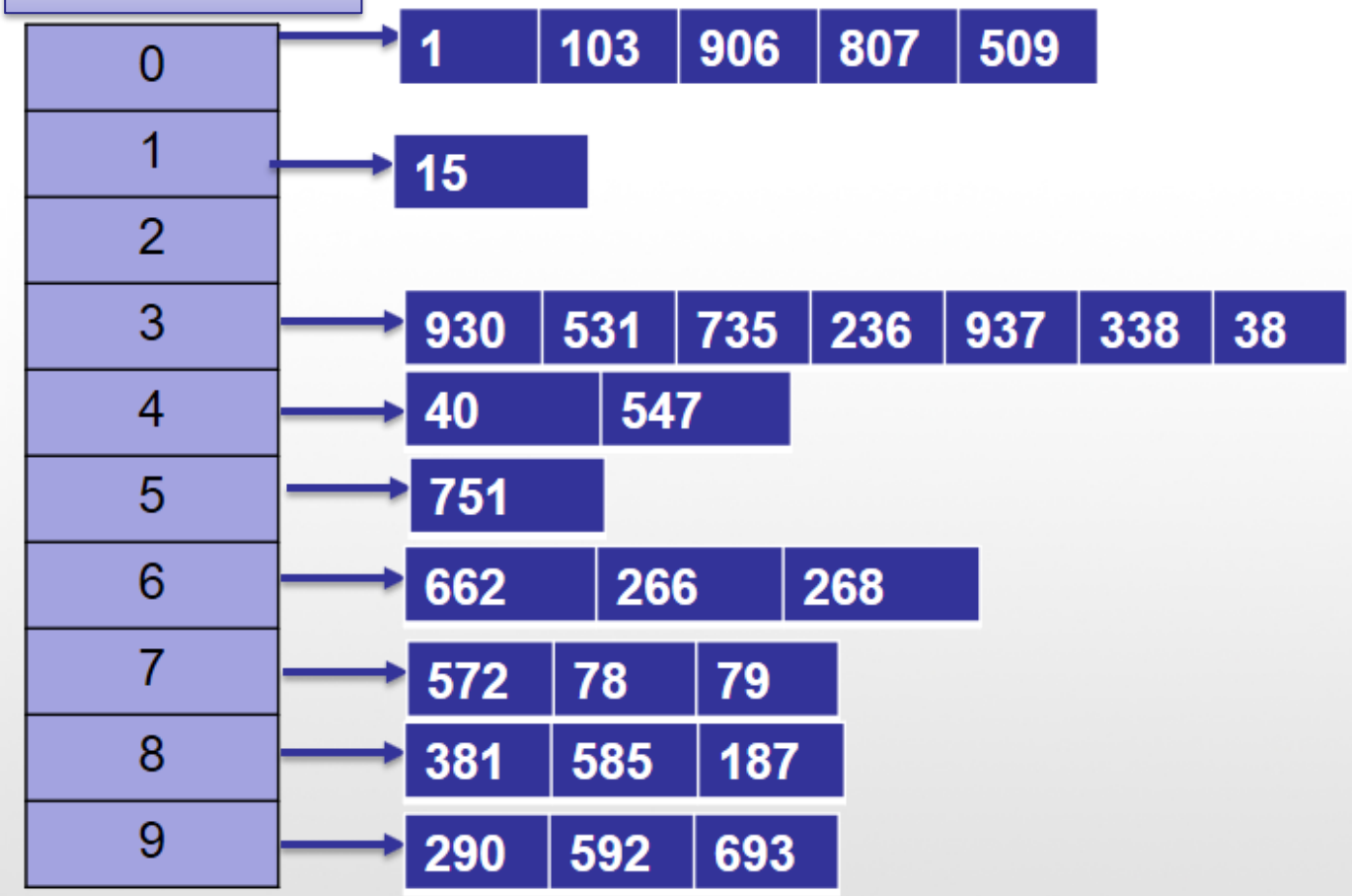


Array Of Linked List

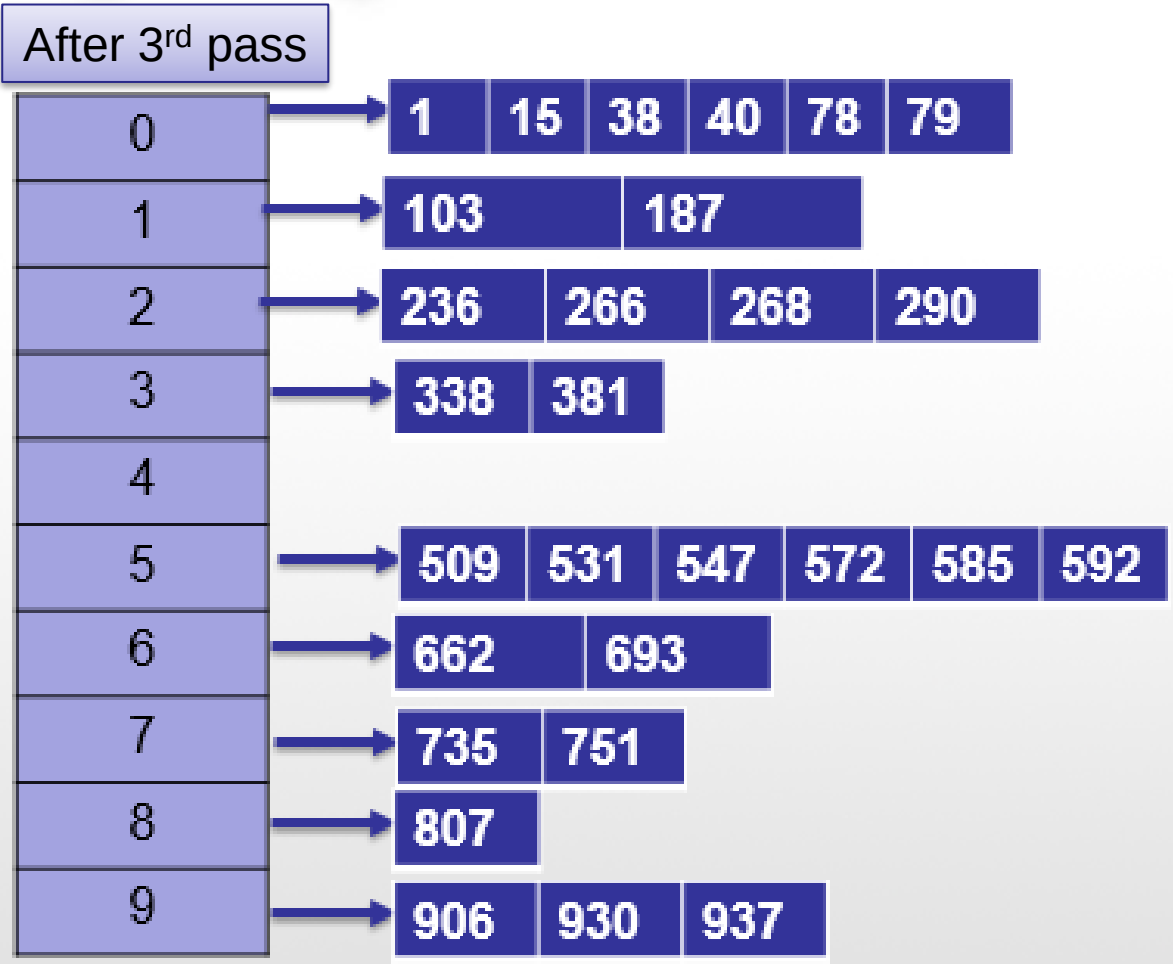
List arrayList [ ]

Case Study: Radix Sort Analysis

After 2nd pass



Case Study: Radix Sort Analysis



Case Study: Radix Sort Analysis

Input size: ***n (number of elements)***

Number of buckets = Radix: ***B***

Number of passes = "Digits": ***P***

Total work is $O(P \cdot (B + n))$

We do p stable sorts that each run O(n) time

Thus in our example, the algorithm will take

T(n) = O (3 * (10 + 28) = 114 execution time.

T(n) = O (n)

Radix Sort : H.W

You have one week to do the following:

- **Sort the following input integers using Radix Sort.**

Input data :478,537,9,721,3,38,123,67

- **If we have a 21 strings of English letters up to the length of 15 character. Find the execution time.**

- **Implement Radix Sort to sort set of strings .**

1. **Read set of Strings**
2. **Print the set before sorting**
3. **Print the set after sorting**

Question?

“Success is the sum of small efforts, repeated day in and day out.”
Robert Collier

References:

1. http://pages.cs.wisc.edu/~vernon/cs367/notes/4.LINKED-LIST.html
2. https://www.tutorialspoint.com/java/java_linkedlist_class.htm
3. Marty Stepp Lecture Notes