Binary Trees, Expression Trees, and Binary Search Trees

Abdallah Karakra

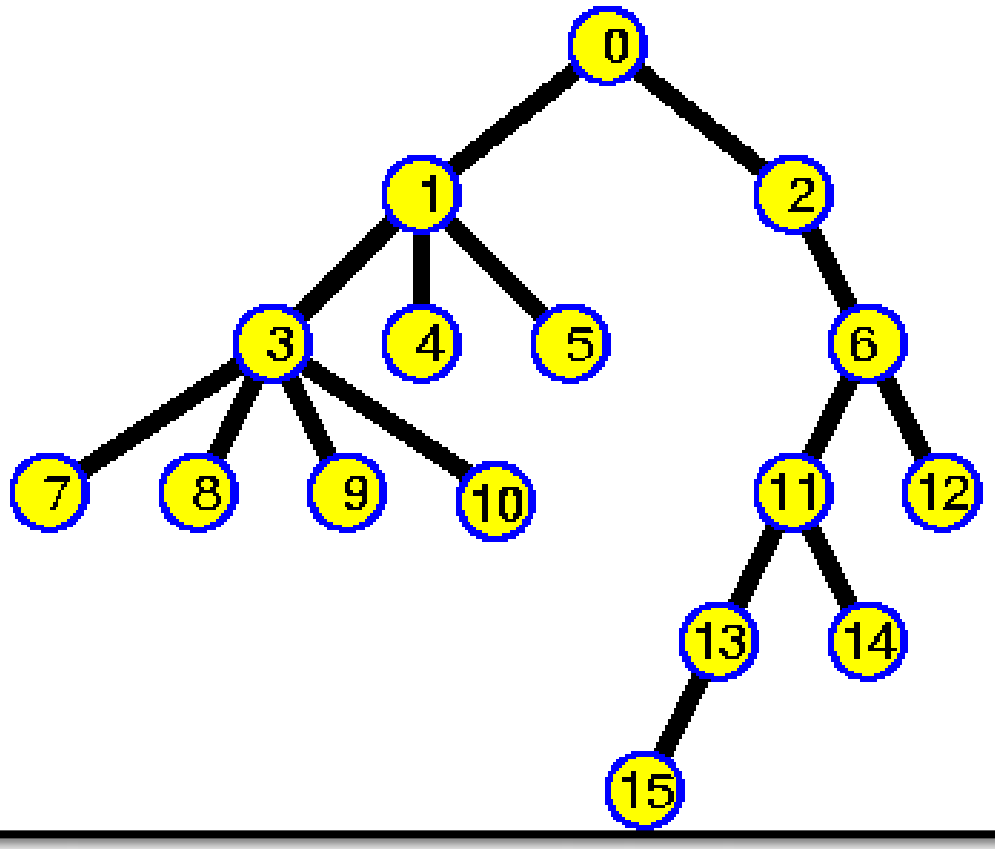
Computer Science Department

COMP242

Wednesday, May 24, 2023

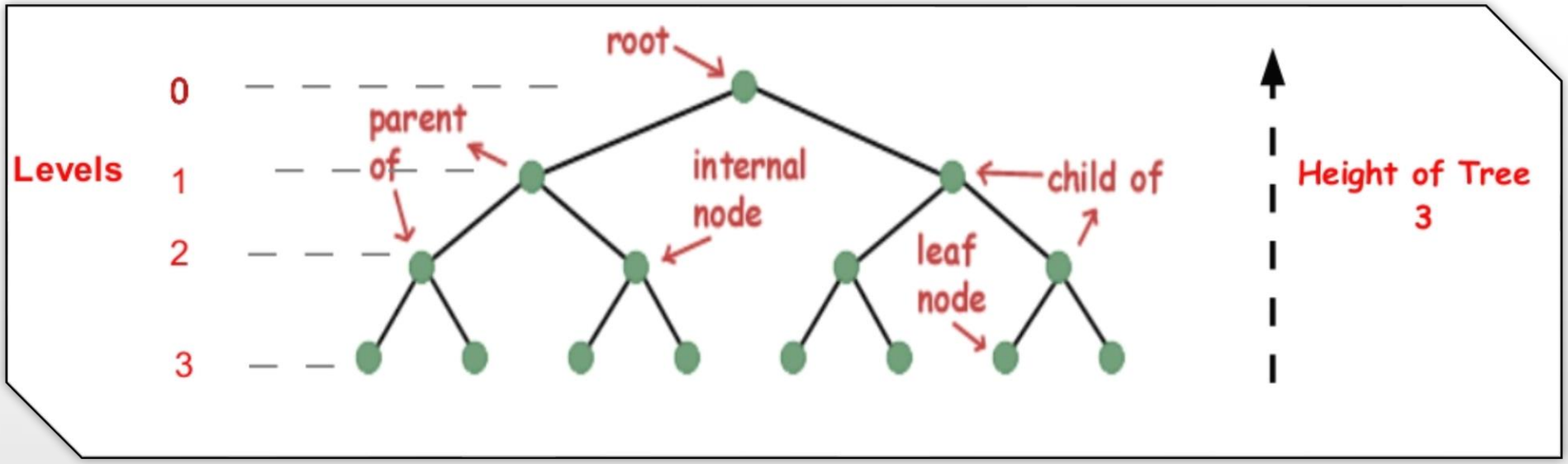
Recall

Tree Definitions



Tree has 16 nodes
Tree has degree 4
Tree has depth 5
Node 0 is the root
Node 1 is internal
Node 4 is a leaf
4 is a child of 1
1 is the parent of 4
0 is grandparent of 4
3, 4 and 5 are siblings

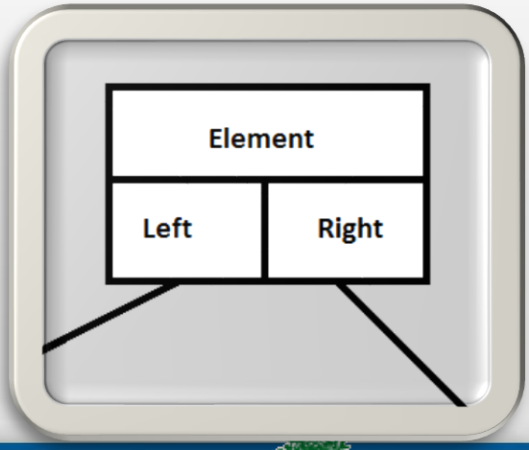
Recall



Binary Trees

- A binary tree is a tree in which no node can have more than two children(every node has at most 2 children).

- Each node has an element, a reference to a left child and a reference to a right child.



Full and Complete Binary Tree

There are two forms of binary tree:

1. **Full binary tree**: a binary tree T is full if each node is either **a leaf** or possesses **exactly two child nodes**.

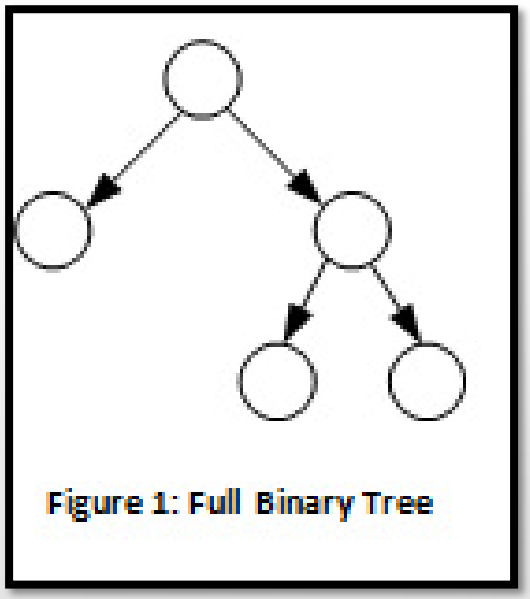
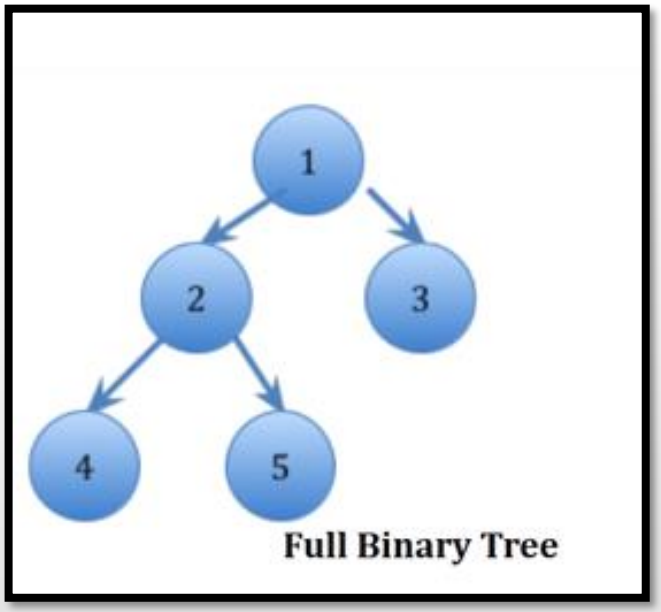
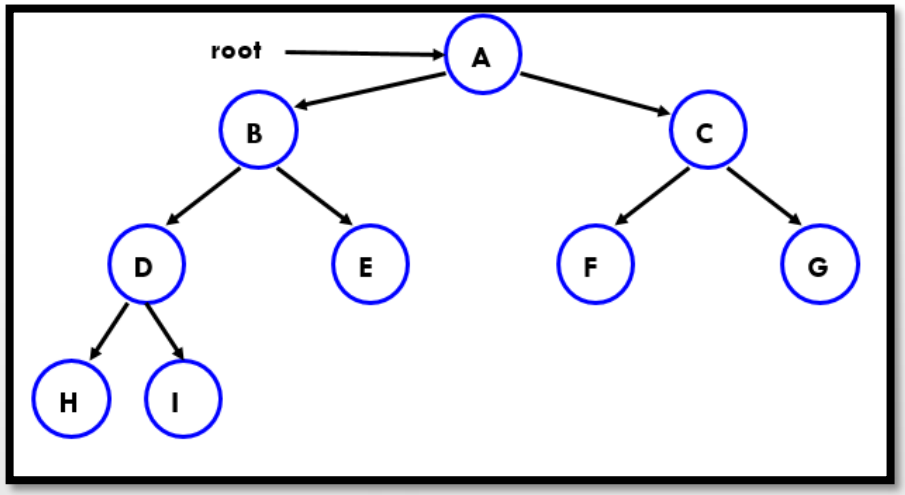


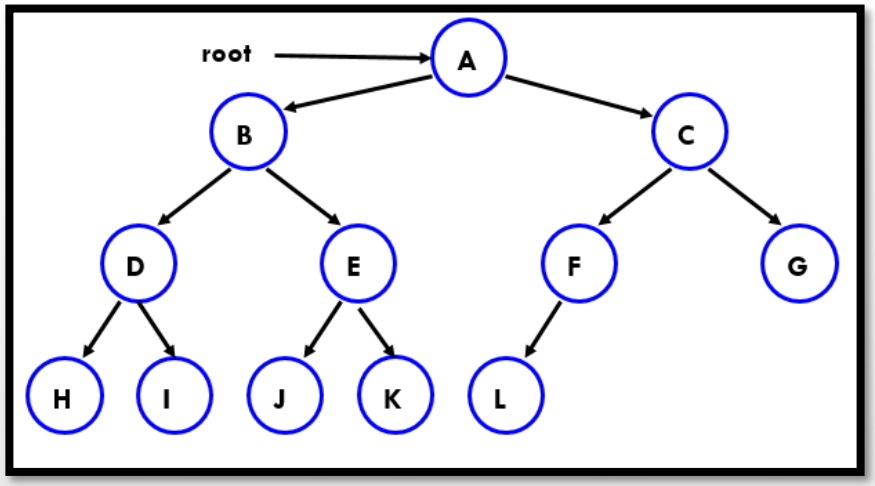
Figure 1: Full Binary Tree



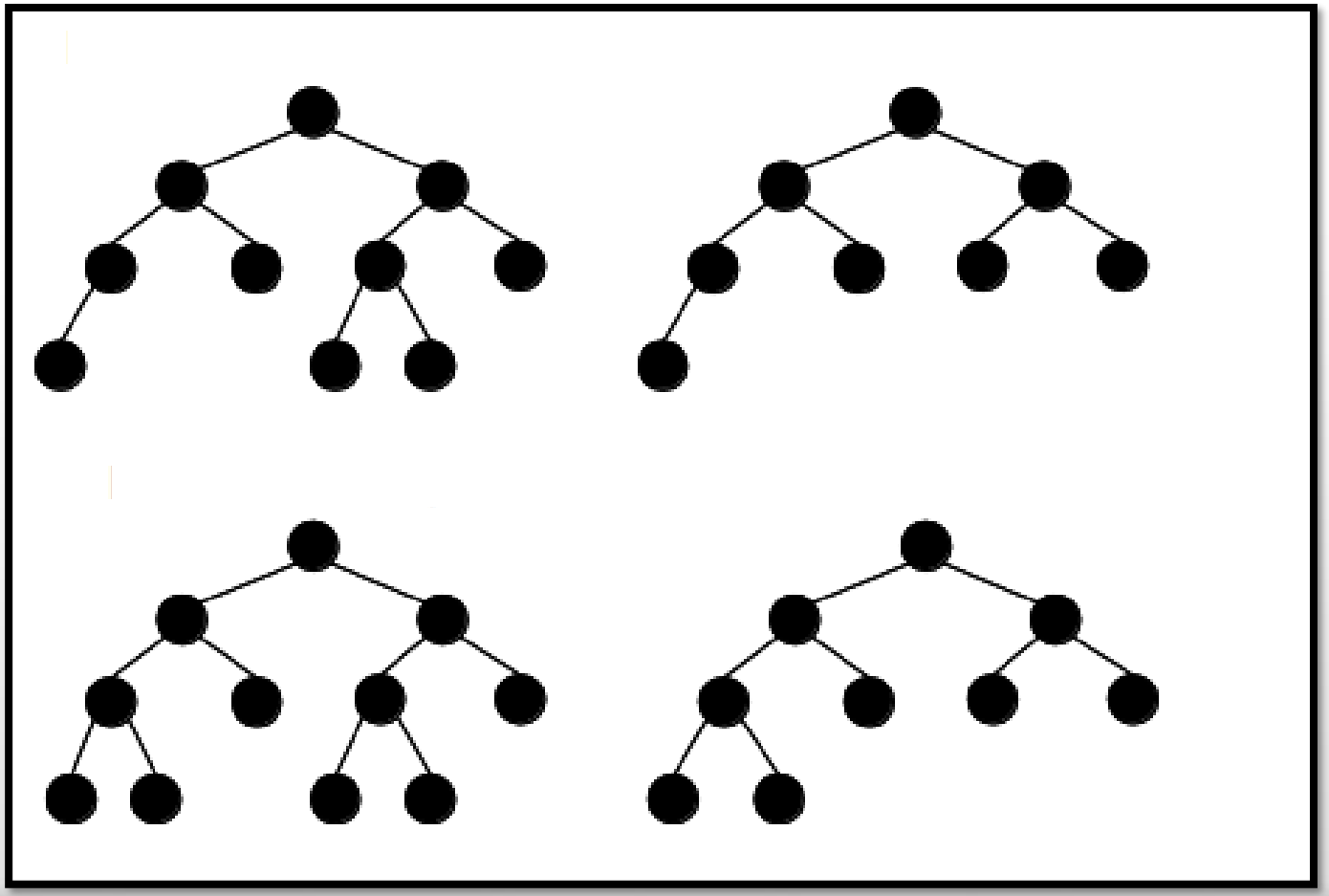
Full and Complete Binary Tree

2. A complete binary tree: is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right.

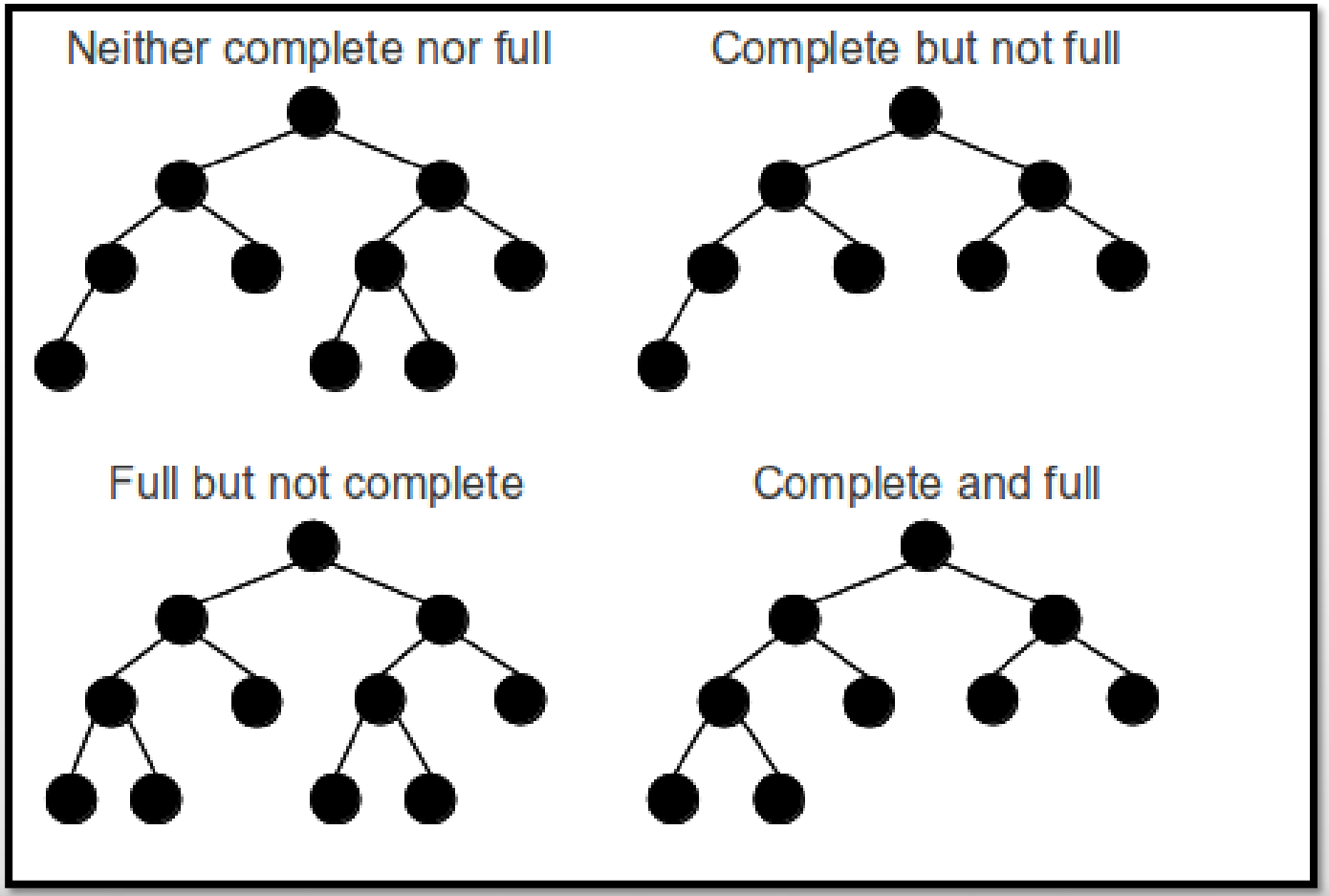




Full and Complete Binary Tree

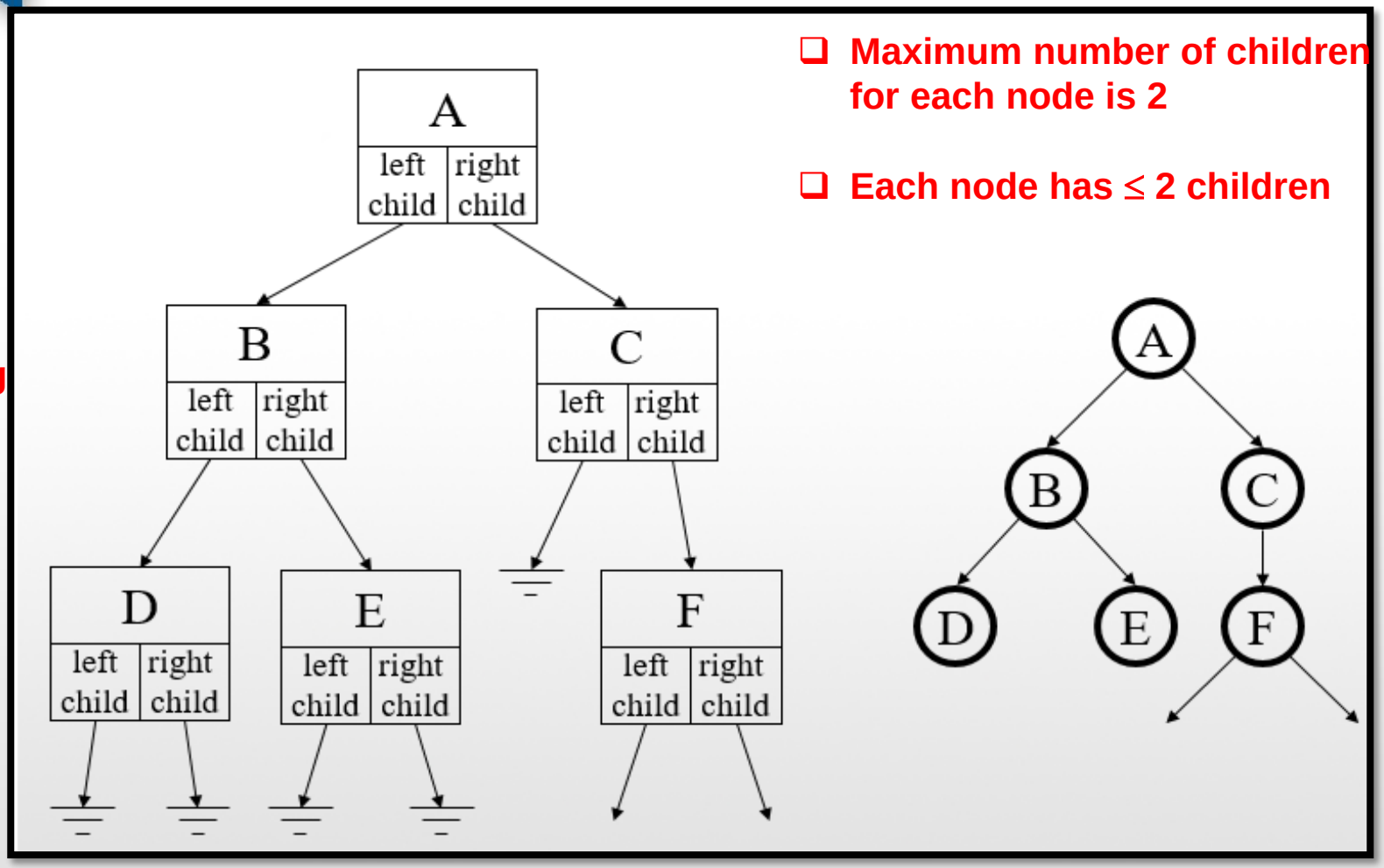


Full and Complete Binary Tree



SELF STUDY

Binary Tree Representation

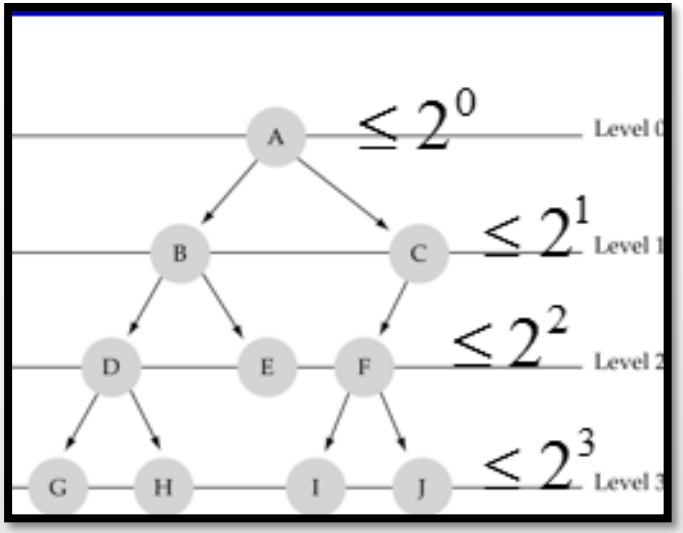


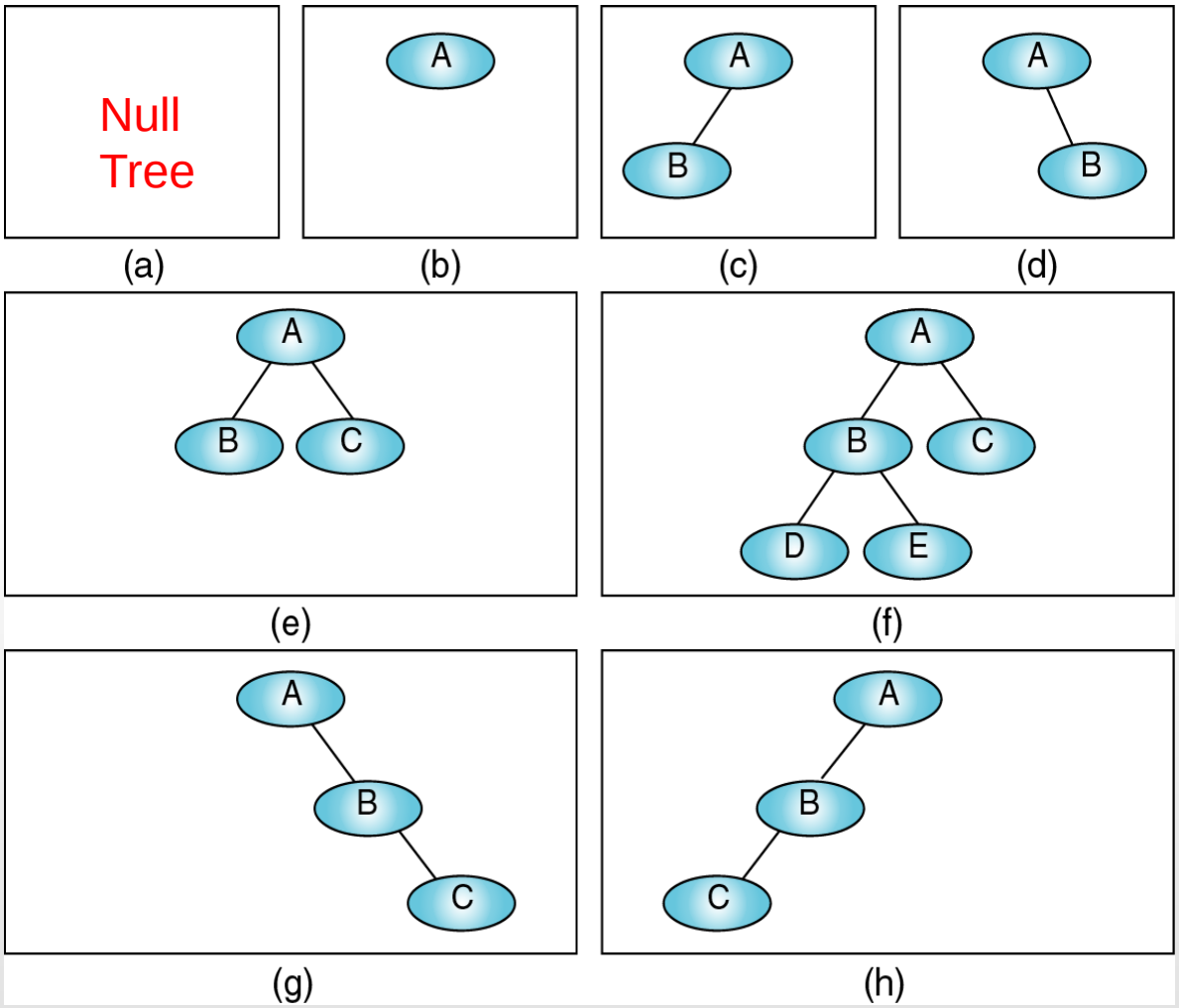
- **Maximum number of children for each node is 2**
- **Each node has ≤ 2 children**

Each node is labeled as being either a left child or a right child

Binary Tree Representation

Max number of node in each level <= 2^L where L=0,1,2,…,L-1





Implementation : Binary Tree

```
//Class Node for the Binary Tree
public class BinaryTreeNode {
   Object element;          //store data
   BinaryTreeNode left;    // left child
   BinaryTreeNode right; //right child

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

  public BinaryTreeNode(Object element,BinaryTreeNode left,BinaryTreeNode right)
   {
      this.element=element;
      this.left=left;
      this.right=right;
    }
}
```

Expression Trees

- A Binary Expression Tree is **A special kind of binary tree in which**

1.Each leaf is an operand (ex: constants, variables names,.).

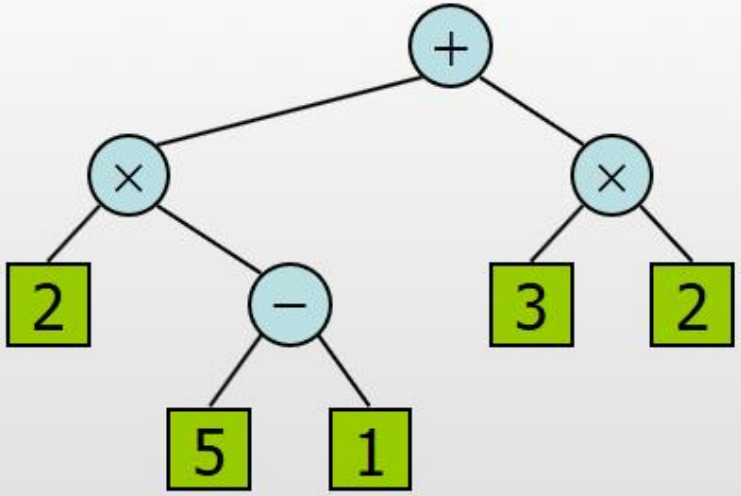
2.The root and internal nodes are operators (ex: +, - ,*, ..etc).

3.Subtrees are subexpressions with the root being an operator**.**

Example:

expression tree for the expression

((2 × (5 − 1)) + (3 × 2))



Constructing an Expression Tree

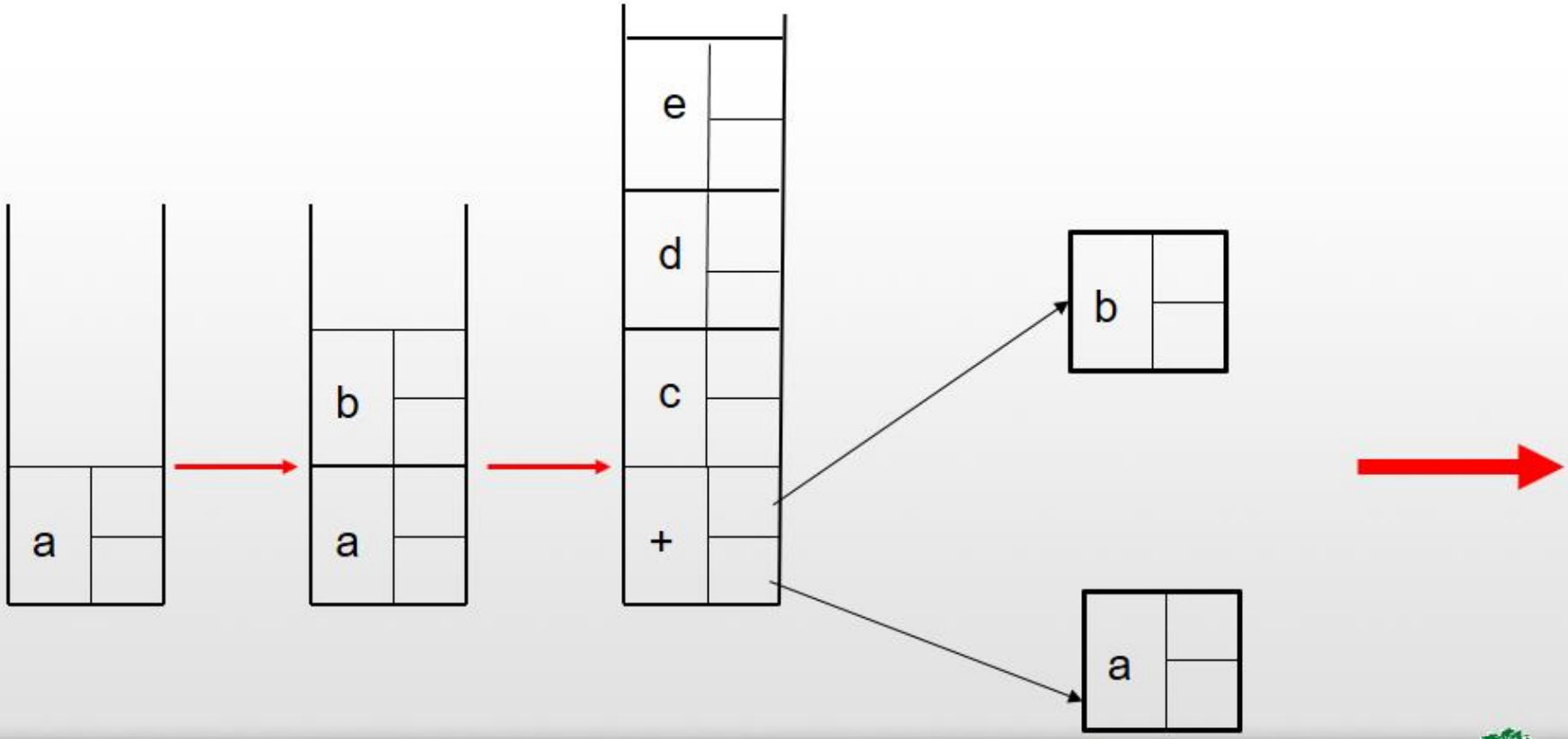
- The pseudo code algorithm to convert a valid postfix expression, containing binary operators, to an expression tree: (**We will use a stack to build an expression tree from postfix**)

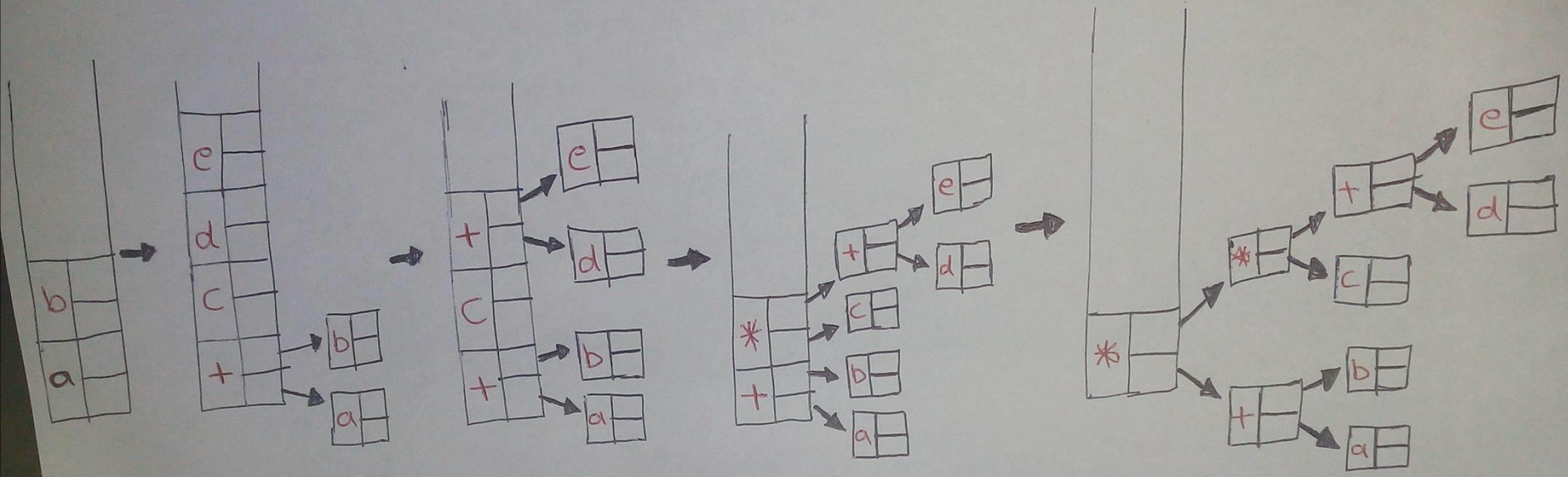
```
1  while(not the end of the expression)
2  {
3     if(the next symbol in the expression is an operand)
4     {
5       create a node for the operand ;
6       push the reference to the created node onto the stack ;
7     }
8    if(the next symbol in the expression is a binary operator)
9    {
10     create a node for the operator ;
11     pop from the stack a reference to an operand ;
12     make the operand the right subtree of the operator node ;
13     pop from the stack a reference to an operand ;
14     make the operand the left subtree of the operator node ;
15     push the reference to the operator node onto the stack ;
16   }
17 }
```

Constructing an Expression Tree

Example: Consider the expression (a + b) * (c * (d + e)).

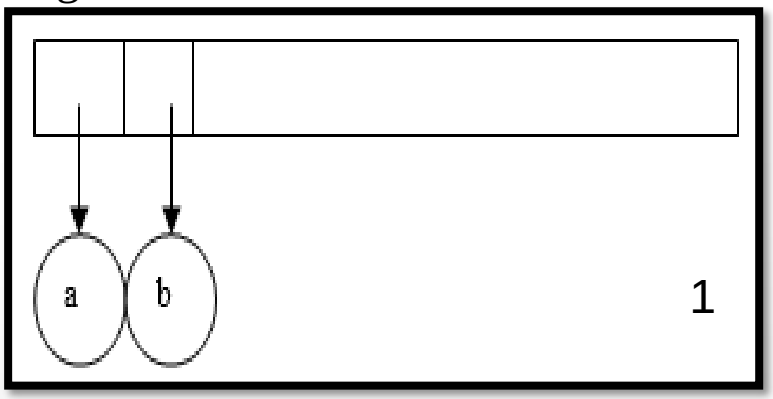
The postfix expression is: a b + c d e + * *

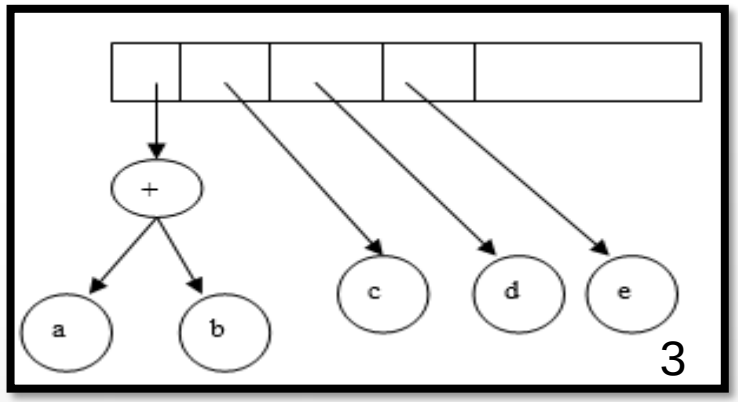


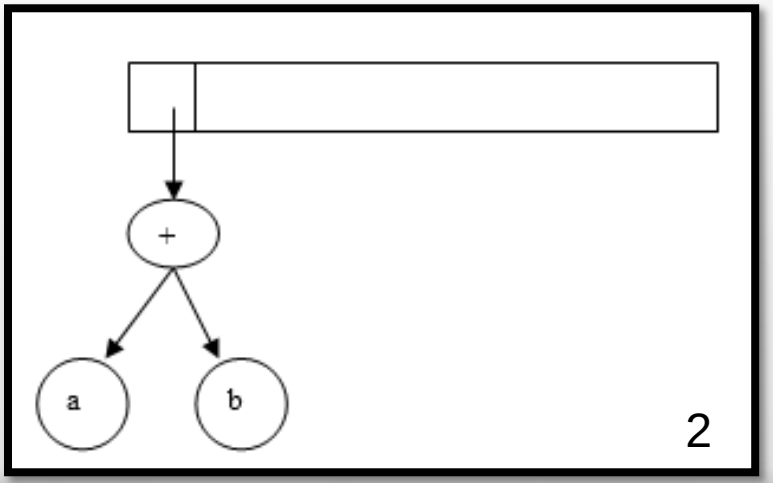


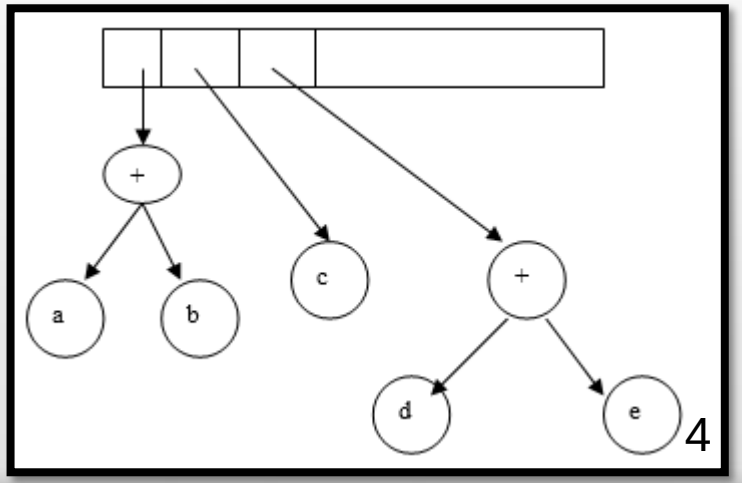
Constructing an Expression Tree

(I repeat the previous diagram. For convenience, the stack grow from left to right in the diagrams

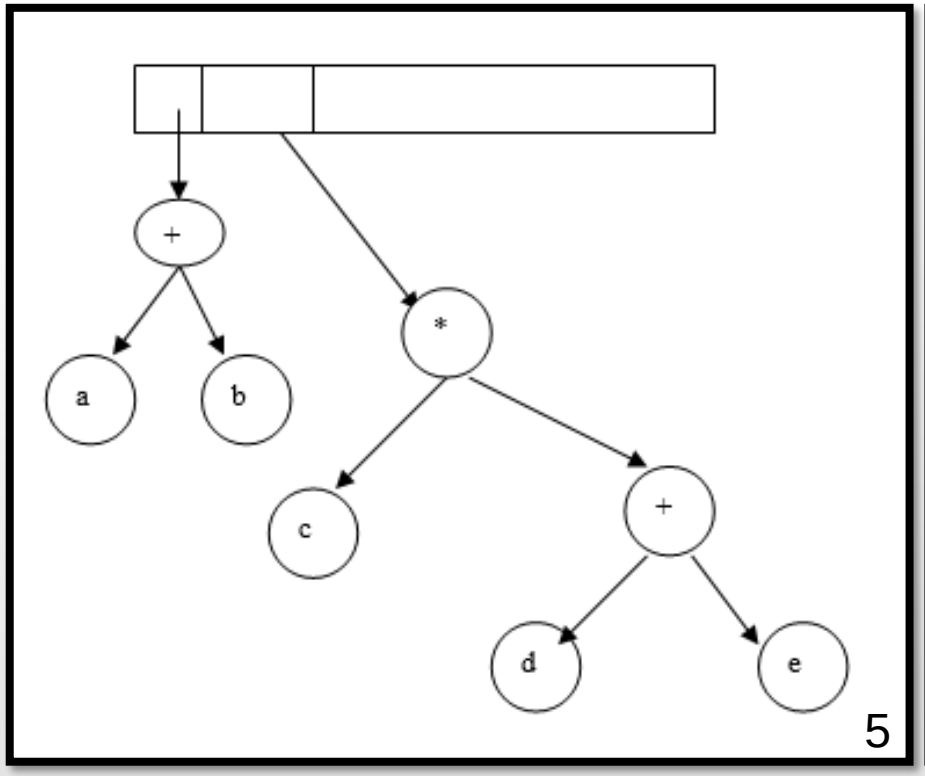


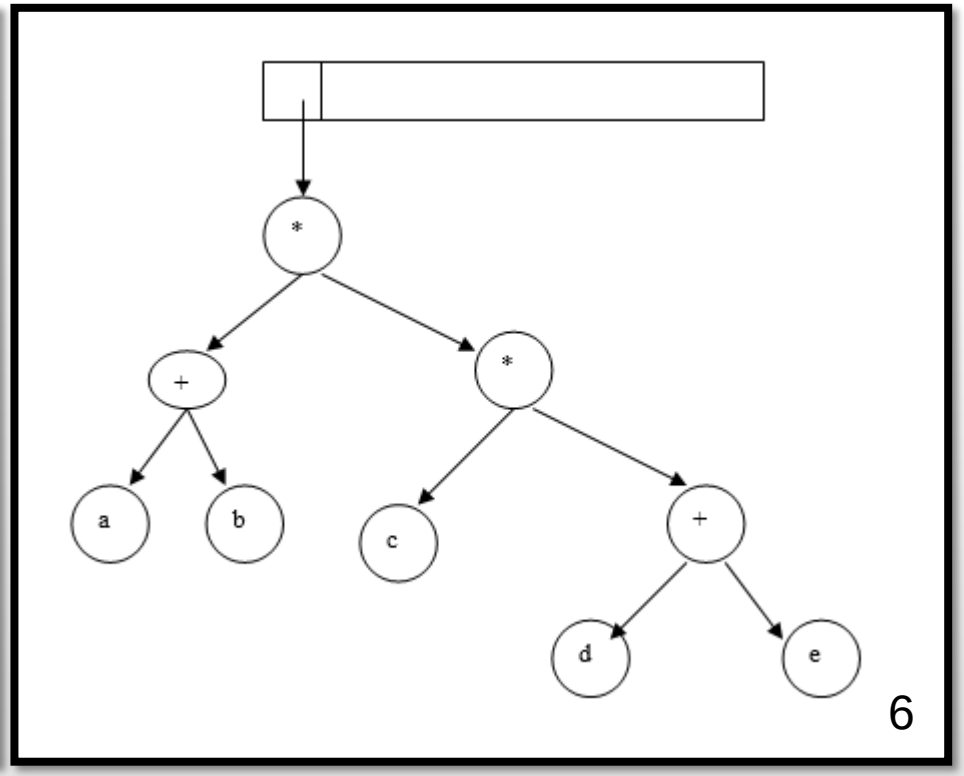






Constructing an Expression Tree





Examples

Give the prefix, infix and postfix expressions corresponding to the tree in the figure below.

Hint : prefix is preorder , infix is inorder, and postfix is postorder

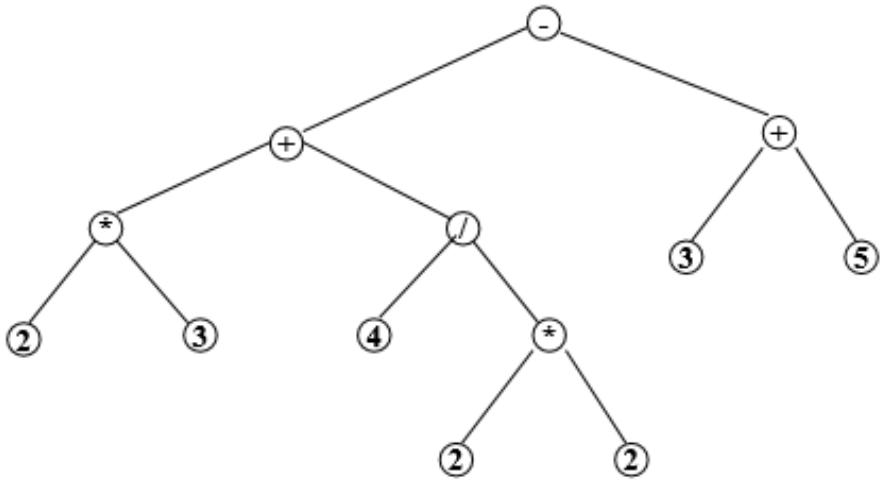


Figure 1: Expression Tree.

Solution:

prefix: - + * 2 3 / 4 * 2 2 + 3 5

infix: 2 * 3 + 4 / 2 * 2 - 3 + 5

postfix: 2 3 * 4 2 2 * / + 3 5 + -

A binary search tree

- **A binary search tree is a binary tree with a special property called the BST-property, which is given as follows:**

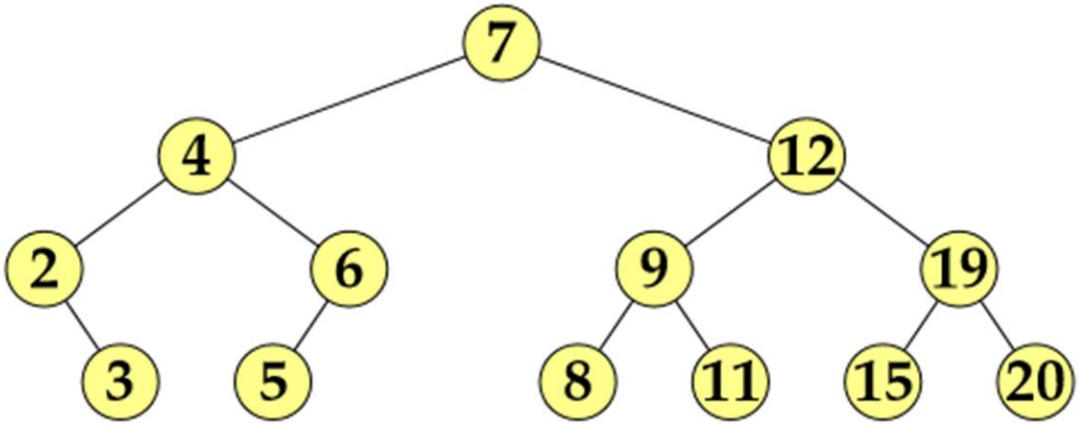
For all nodes x and y, if y belongs to the left subtree of x, then the key at y is less than or equal the key at x, and if y belongs to the right subtree of x, then the key at y is greater than or equal the key at x

Value (Left) <= Value (Root) < Value (Right)

Any one of them if duplication is allowed

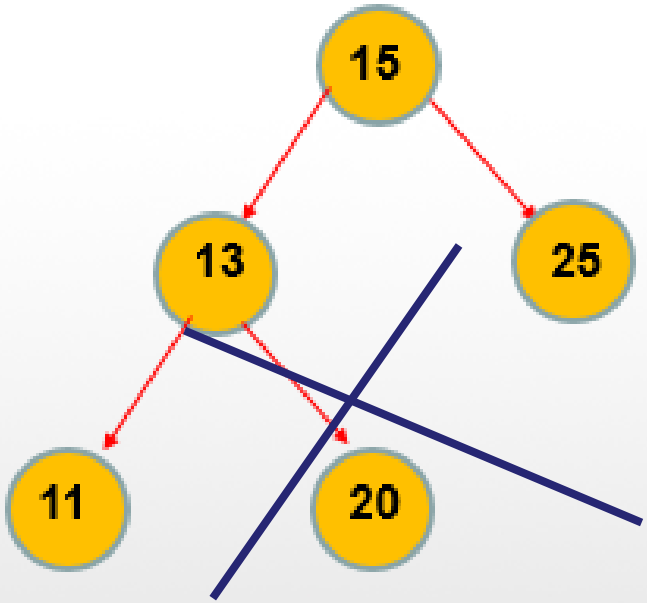
Value (Left) < Value (Root) <= Value (Right)

A binary search tree

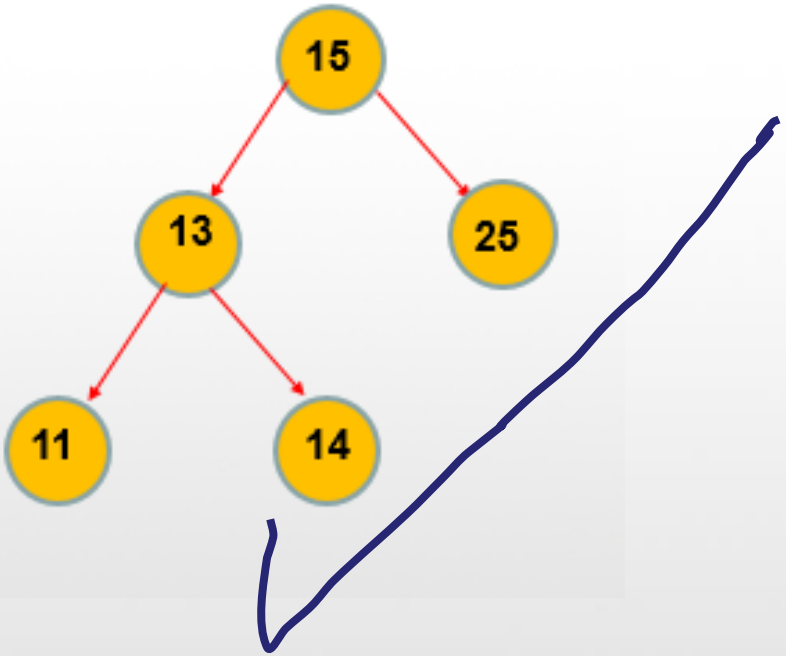


A binary search tree

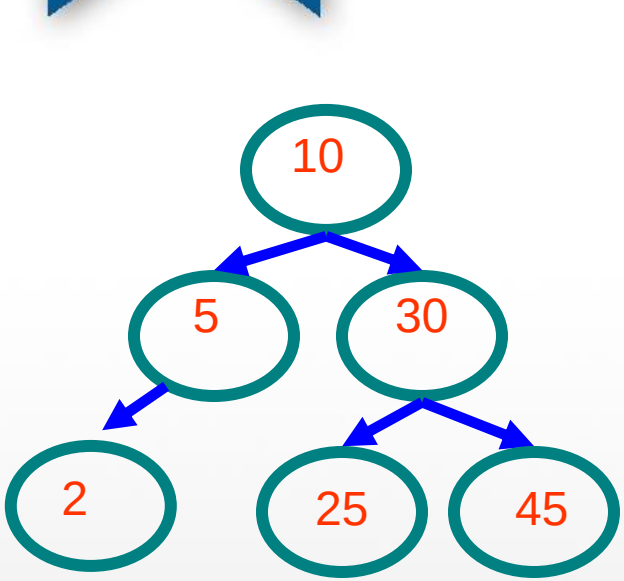
- **Which of the following is a binary search tree ?**

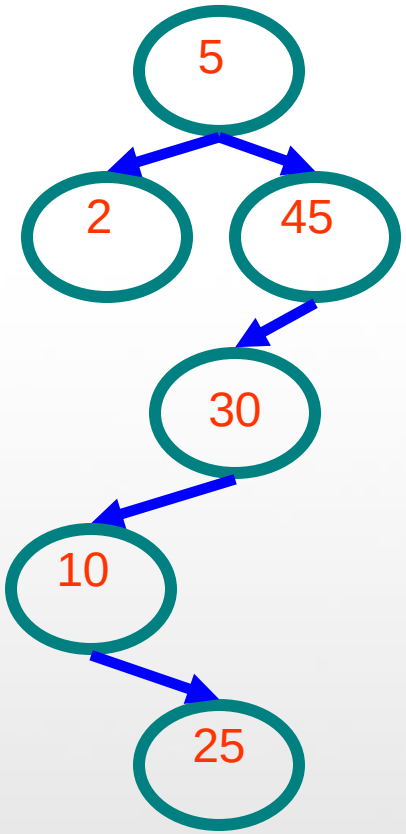


20 ✗ 15

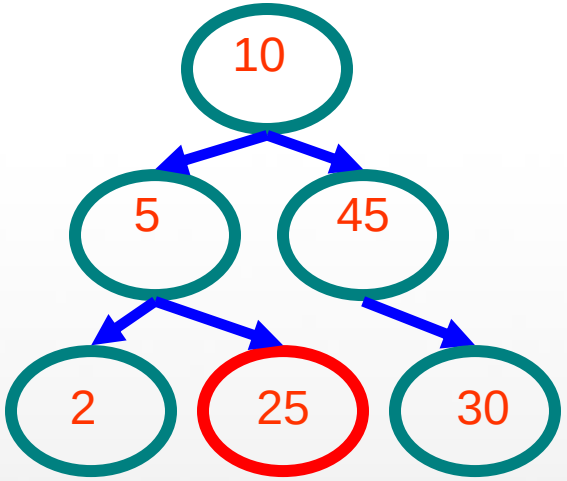


A binary search tree



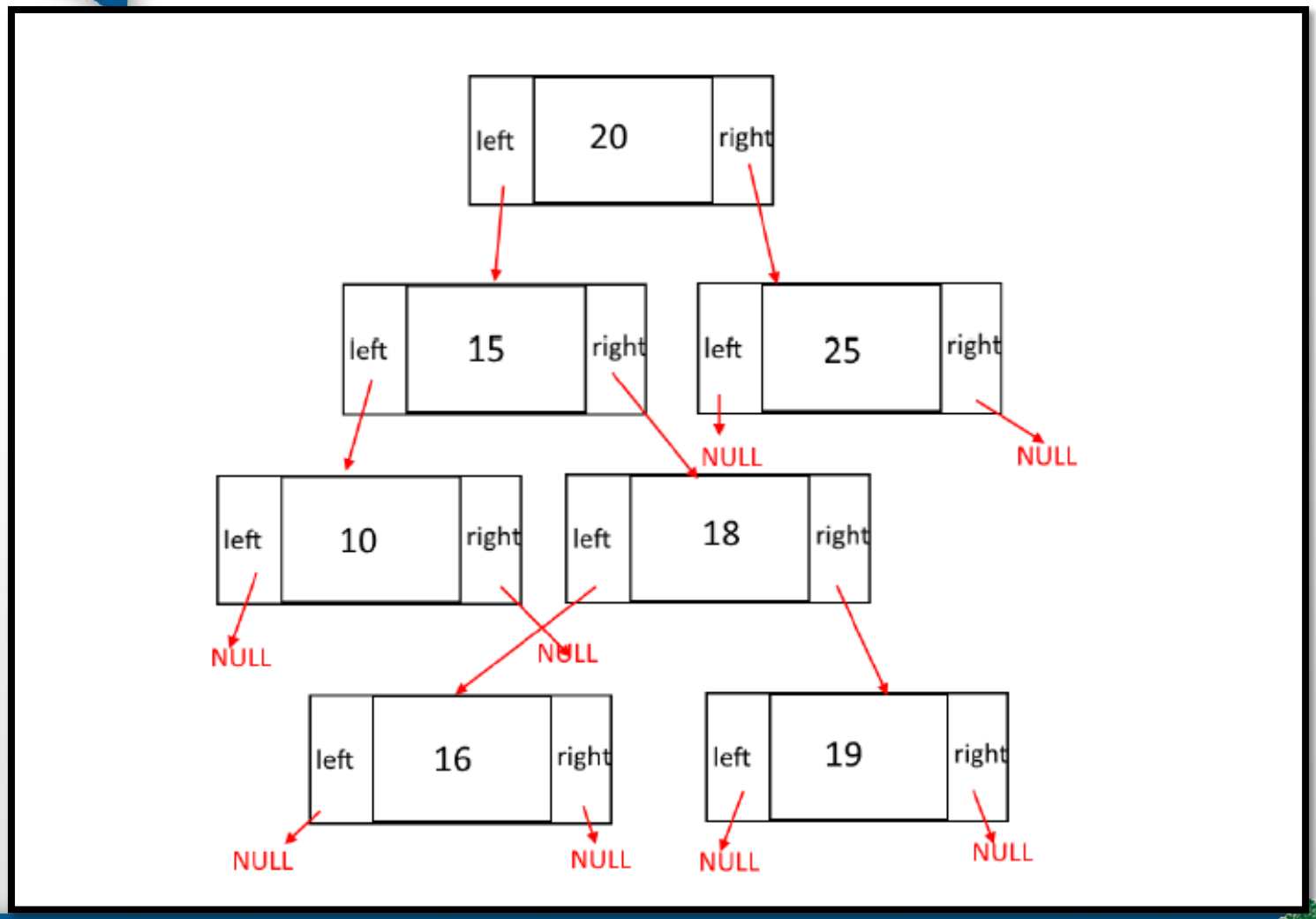


Binary search trees



Non-binary search tree

A binary search tree: Coding



Implementation : Binary Search Tree

```
//Class Node for the Binary Search Tree
public class BSTNode {
//for objects replace int to Object and modify the code
   int element;
   BSTNode left;
   BSTNode right;

   public BSTNode (int element){
      this (element,null,null);
   }
   public BSTNode (int element,BSTNode left,BSTNode right){
      this.element=element;
      this.left=left;
      this.right=right;
   }

}
```

Implementation : Binary Search Tree

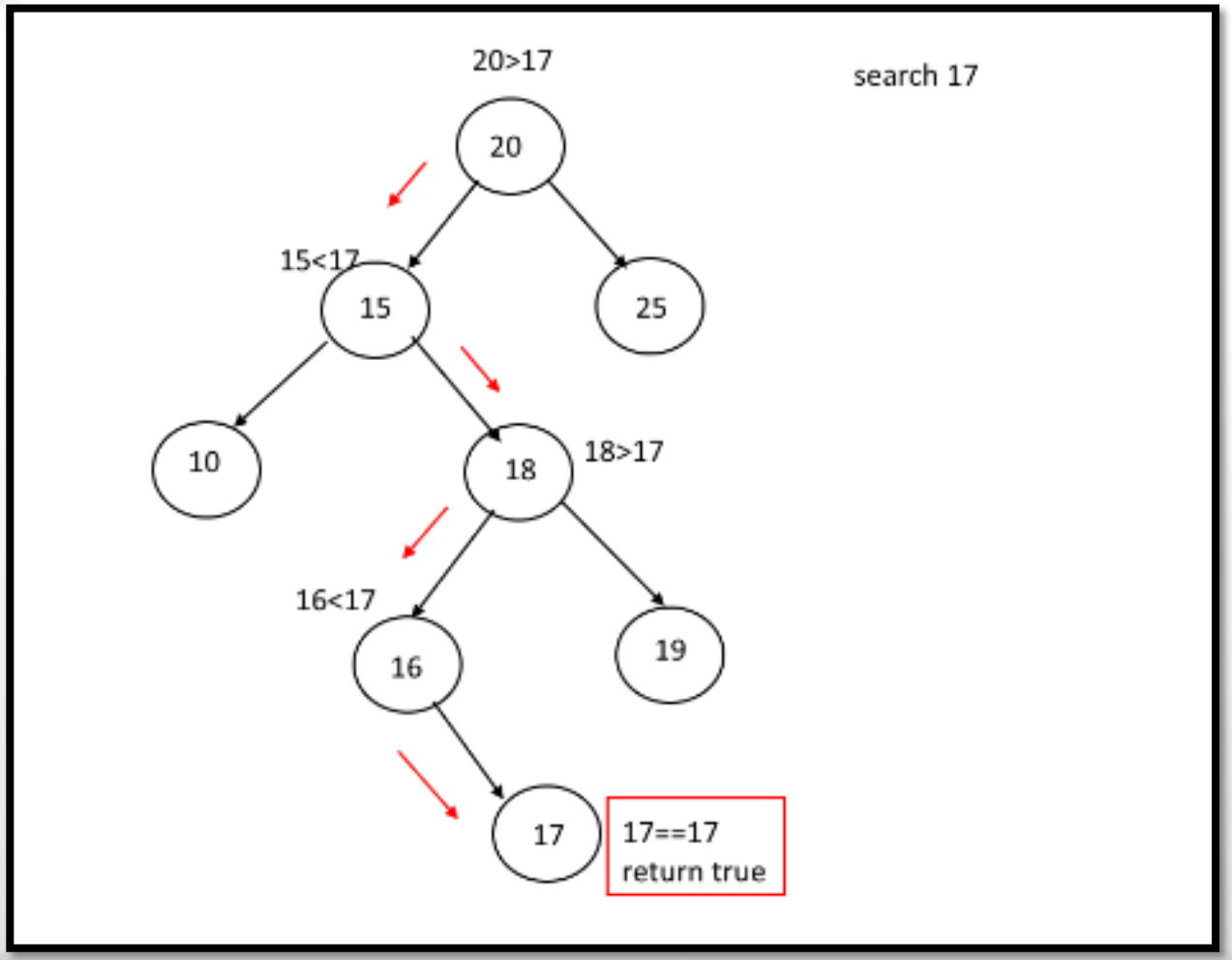
```
// Binary Search Tree Class

public class BST {

   private BSTNode root;

   public BST(){
     root=null;
    }

   /* Methods go here */

 }
```

Binary Search Tree: Contains

Contains(int n):

- start from the root and compare root.element with n
- if root.element is greater than n that means we need to go to the left of the root.
- if root.element is smaller than n that means we need to go to the right of the root
- if any point of time root.element is equal to the n then we have found the node, return true.
- if we reach to the leaves (end of the tree) return false, we didn't find the element

Binary Search Tree: Contains



Binary Search Tree: Conatins

```
//Check whether the element is in a tree or not
private  boolean contains (int e,BSTNode current)
{
   if (current==null)
       return false; // Not found, empty tree.

    else if (e<current.element) // if smaller than root.
       return contains (e,current.left); // Search left subtree
    else if (e>current.element) // if larger than root.
        return contains (e,current.right); // Search right subtree

    return true; // found .

}
```

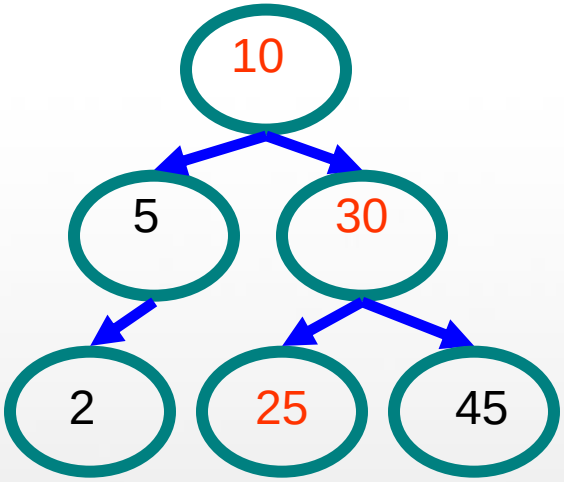
Binary Search Tree: Find

```
//Returns the node contains the given element
private BSTNode find(int element, BSTNode  current)
{
    if (current == null)
       return null;
    if (element < current.element)
        return find(element, current.left);
    else if (element > current.element)
       return find(element, current.right);
    else
       return current;
}
```

Rewrite the above code, using an Iterative way (Loop)?

Binary Search Tree: Find

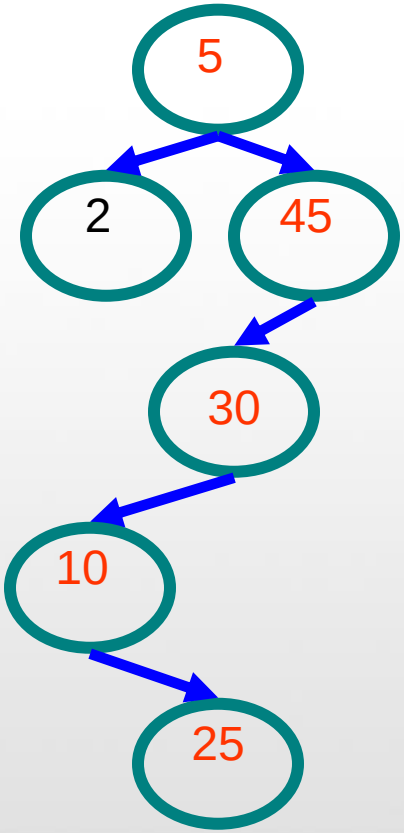
- find (25)



10 < 25, right

30 > 25, left

25 = 25, found



5 < 25, right

45 > 25, left

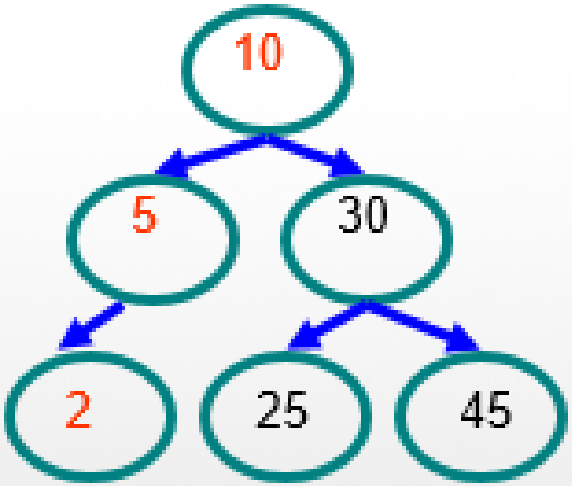
30 > 25, left

10 < 25, right

25 = 25, found

Binary Search Tree: Find

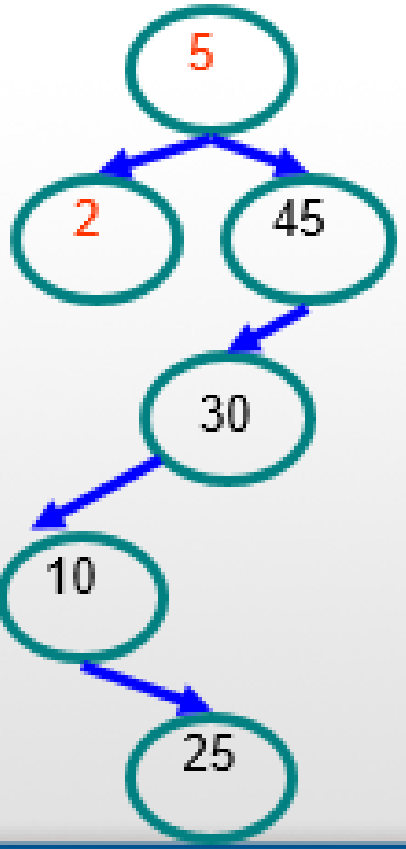
- find (2)



10 > 2, left

5 > 2, left

2 = 2, found



5 > 2, left

2 = 2, found

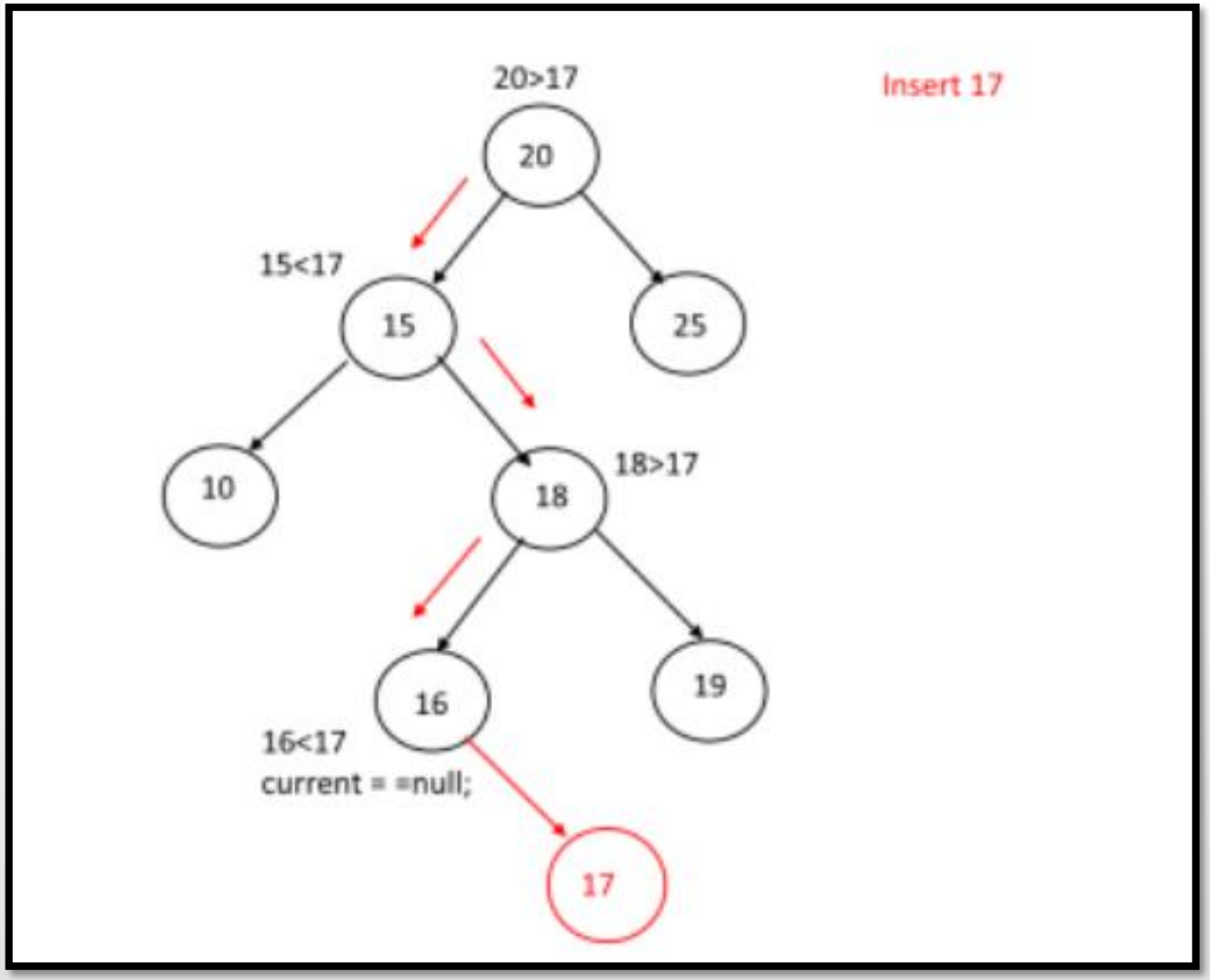
SELF STUDY

Binary Search Tree: Insert

Insert(int n)

- find the **place(location) to insert the node**.
- Let the **current = root** .
- start from the current and compare **current.element with n**
- **if current.element is greater than n** that means we need to **go to the left of the root**.
- **if current.element is smaller than n** that means we need to **go to the right of the root**.
- if any point of **time current is null that means we have reached to the leaf node**, insert your node here with the help of parent node.

Binary Search Tree: Insert

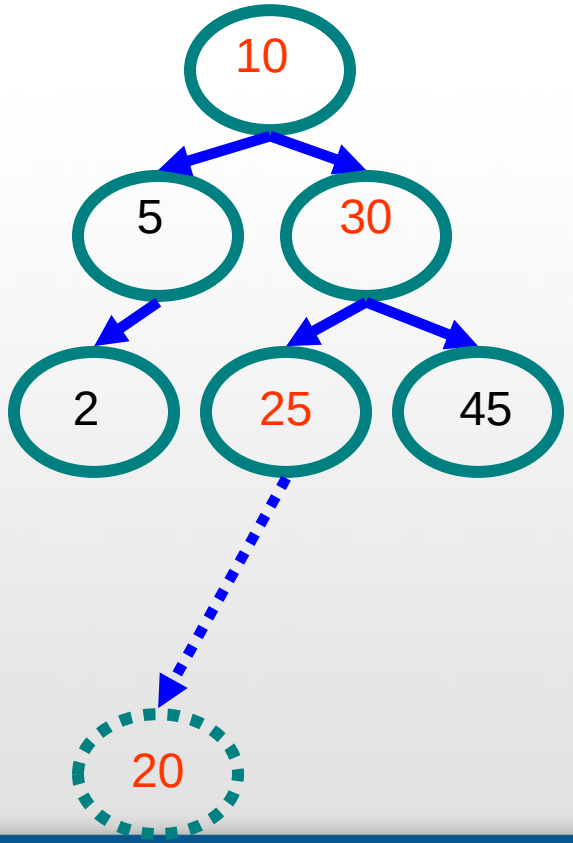


Binary Search Tree: Insert

```
//Insert element function
private  BSTNode insert (int element, BSTNode current){
    if (current==null)
        current= new BSTNode(element); //create one node tree
    else
    {
          if (element<current.element)
             current.left=insert(element,current.left);
          else
             current.right=insert(element,current.right);

    }

    return current;
}
```

Binary Search Tree: Insert

- Insert (20)



10 < 20, right

30 > 20, left

25 > 20, left

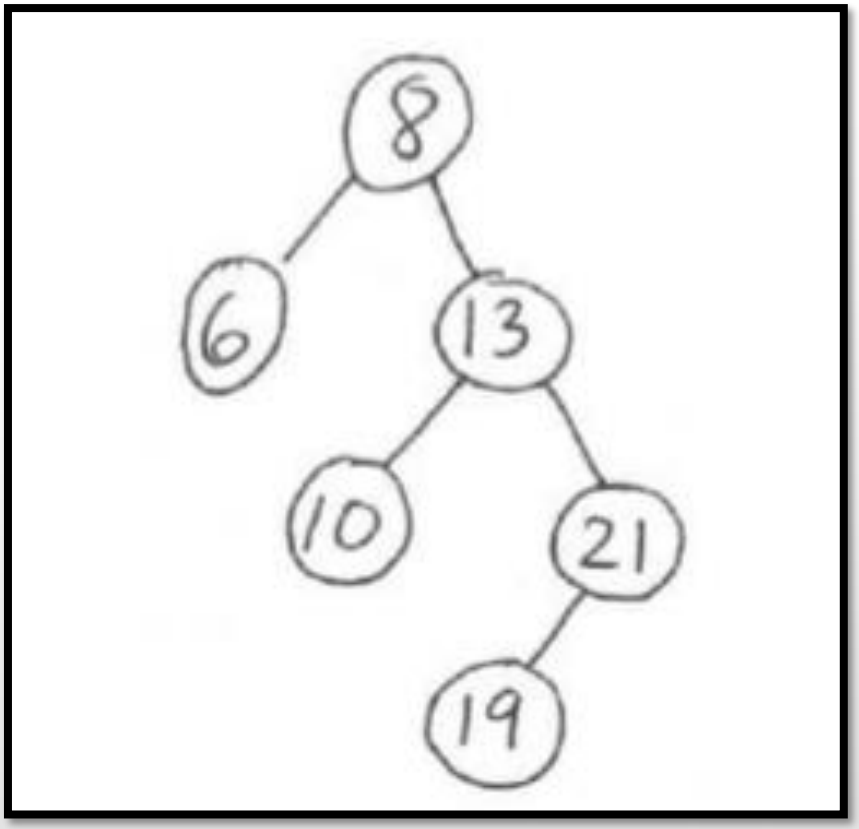
Insert 20 on left

Binary Search Tree: Insert

Beginning with an empty binary search tree, what binary search tree is formed when the following data is inserted in the order given? 8,13,6,10,21,19.

Binary Search Tree: Insert

Beginning with an empty binary search tree, what binary search tree is formed when the following data is inserted in the order given? 8,13,6,10,21,19.

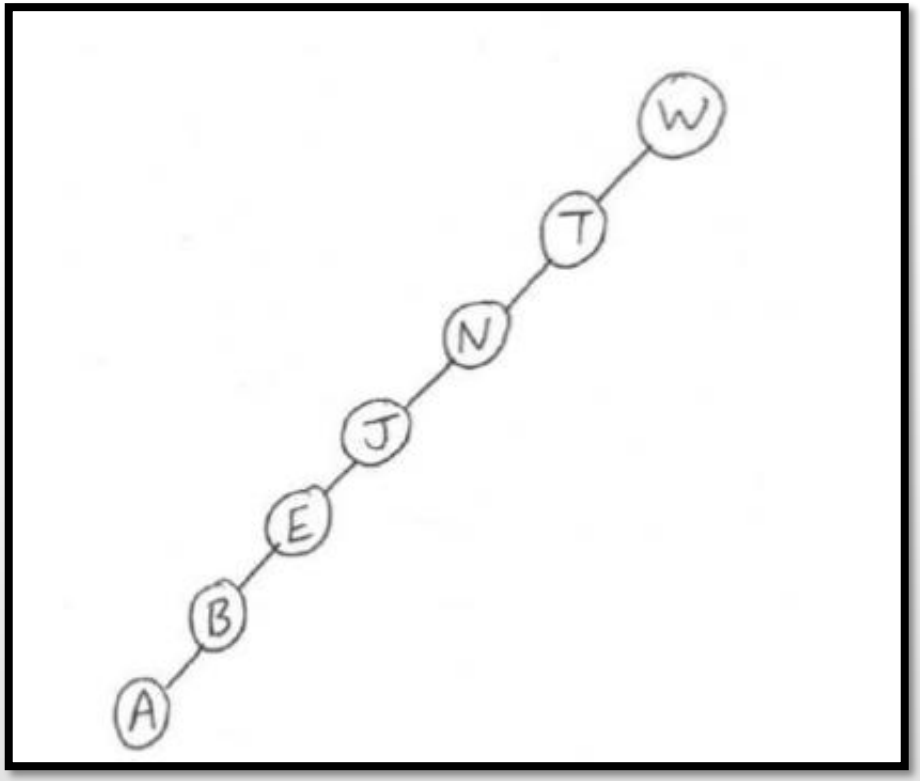


What is the time of search in the formed tree?

Binary Search Tree: Insert

Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given?

W, T, N, J, E, B, A



What is the time of search in the formed tree?

Binary Search Tree : findMin &&findMax

```
private BSTNode findMin (BSTNode current){
 if (current==null)
    return null;
 else if (current.left==null)
     return current;
 else
     return findMin(current.left); //keep going to the left
}
```

```
private BSTNode findMax (BSTNode current){
 if (current==null)
    return null;
 else if (current.right==null)
    return current;
 else
    return findMax(current.right); //keep going to the right
}
```

Binary Search Tree : Delete

Delete(int n):

- **Complicated than Find() and Insert() operations. Here we have to deal with 3 cases.**
- **Node to be deleted is a leaf node (No Children).**
- **Node to be deleted has only one child.**
- **Node to be deleted has two children**

Binary Search Tree : Delete

Delete(int n):

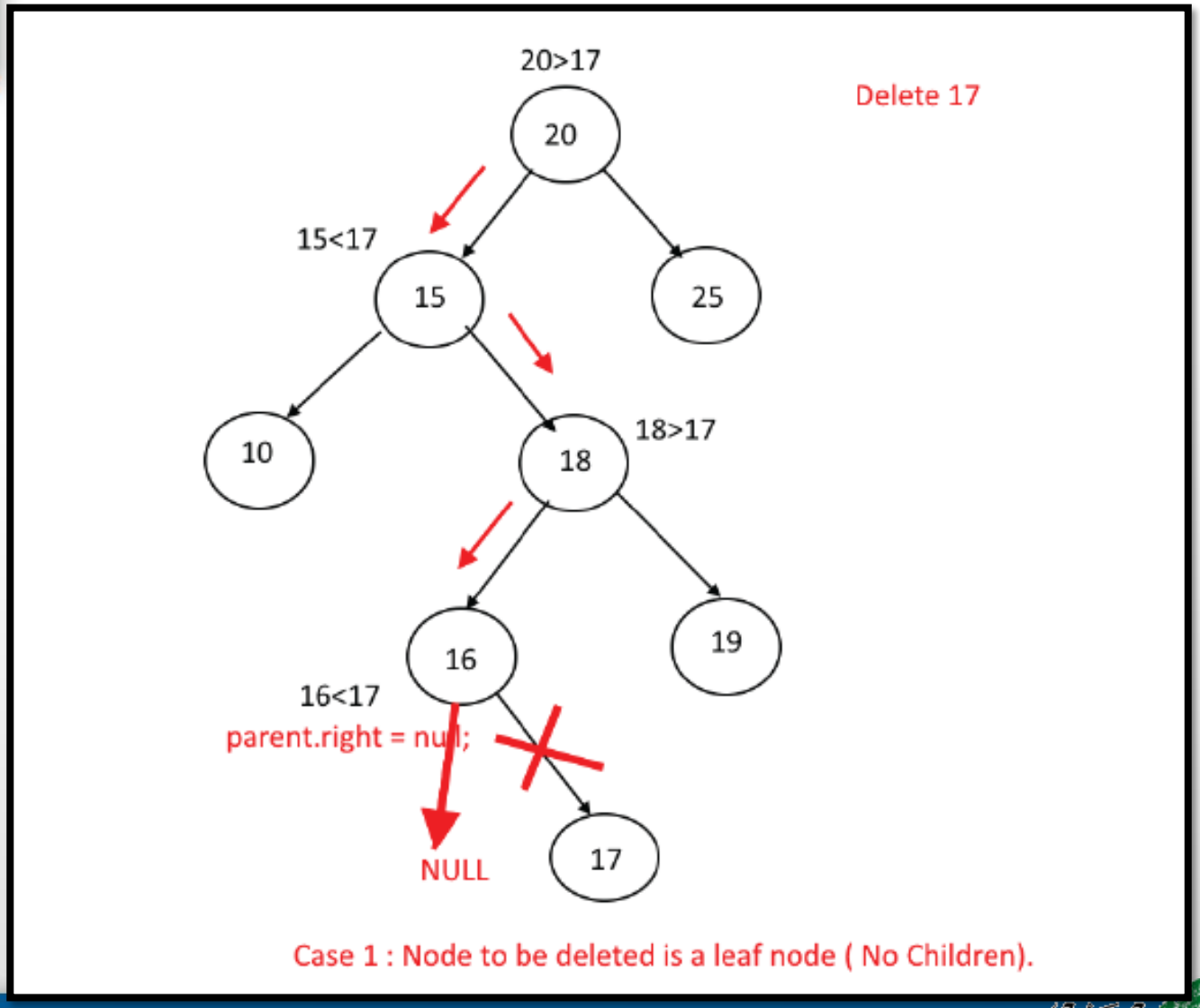
- **Complicated than Find() and Insert() operations. Here we have to deal with 3 cases.**

- **Node to be deleted is a leaf node (No Children).**
  - **Reset its parent link to null**

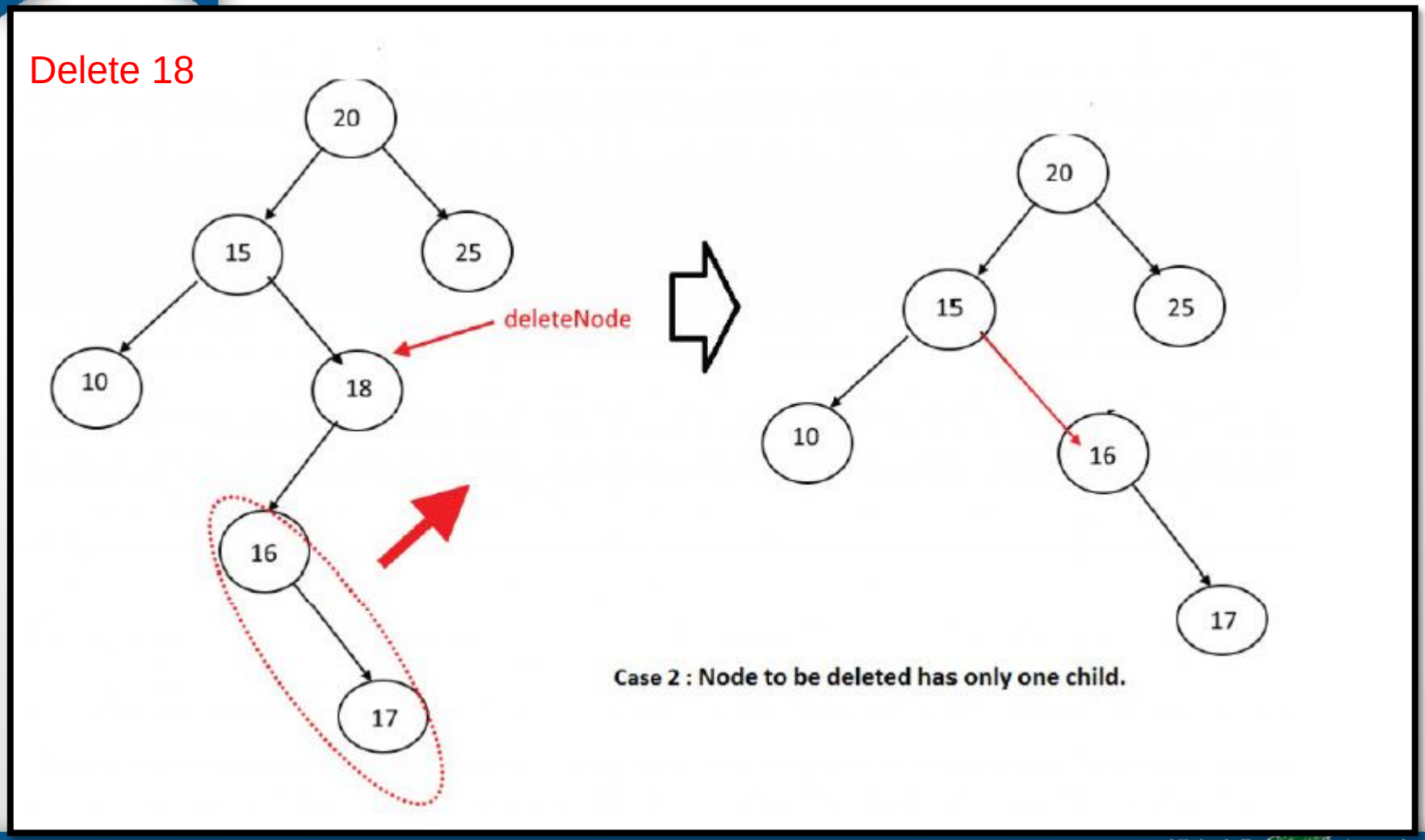
- **Node to be deleted has only one child.**
  - **Replace the node by its single child**

- **Node to be deleted has two children**
  - **Replace the node by the largest one in its left subtree or the smallest one in its right subtree.**
  - **Delete the replacing node from the sub tree.**

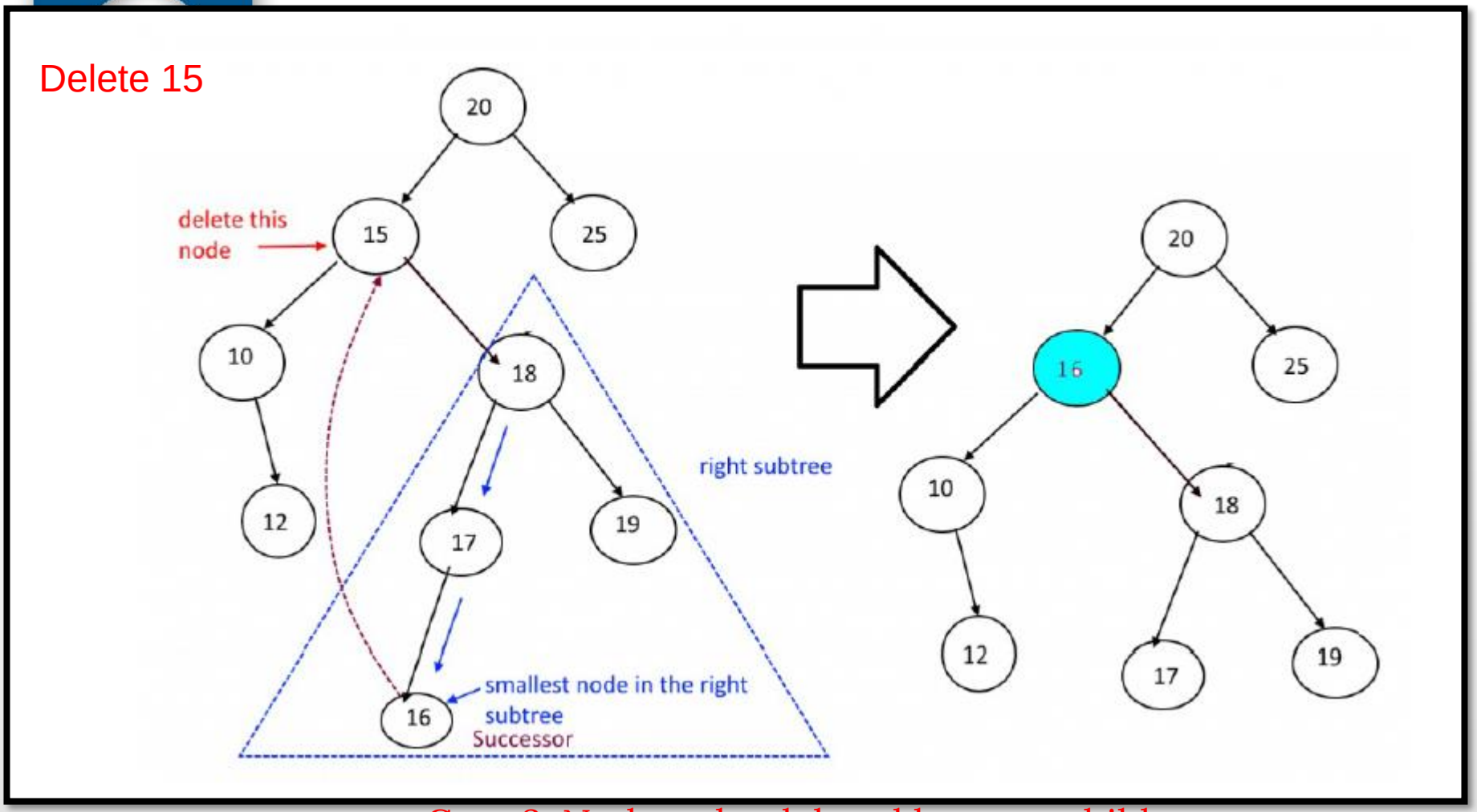
Binary Search Tree : Delete



Binary Search Tree : Delete



Binary Search Tree : Delete



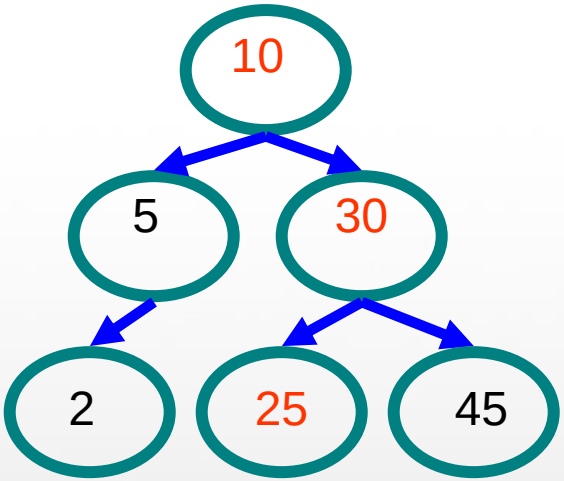
Case 3: Node to be deleted has two children.

Binary Search Tree : Delete

```
private BSTNode remove (int e,BSTNode current){
  if (current==null) return null; // Item not found,Empty tree
  if (e<current.element)
     current.left=remove(e,current.left);
  else if (e>current.element)
     current.right=remove(e,current.right);
  else // found element to be deleted
  if (current.left!=null && current.right!=null)// two children
   {
         /*Replace with smallest in right subtree */
         current.element=findMin(current.right).element;
         current.right=remove(current.element,current.right);
   }
  else// one or zero child
   current= (current.left!=null)?current.left:current.right;

  return current;
}
```

Binary Search Tree : Delete

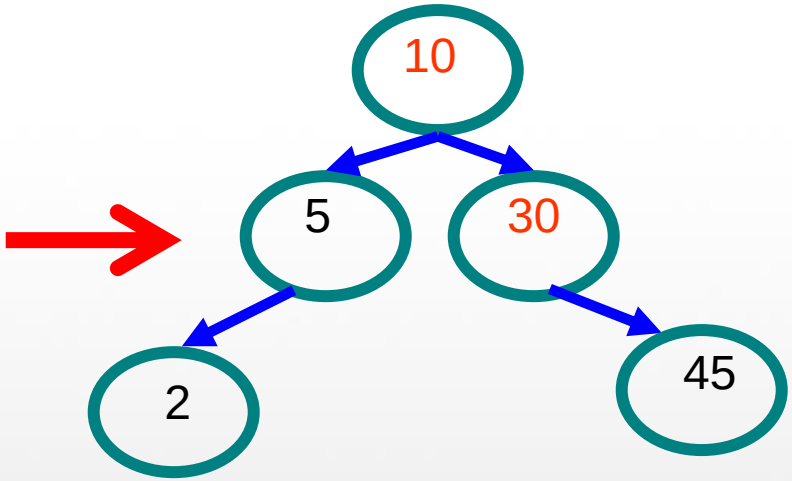
- Delete (25)



10 < 25, right

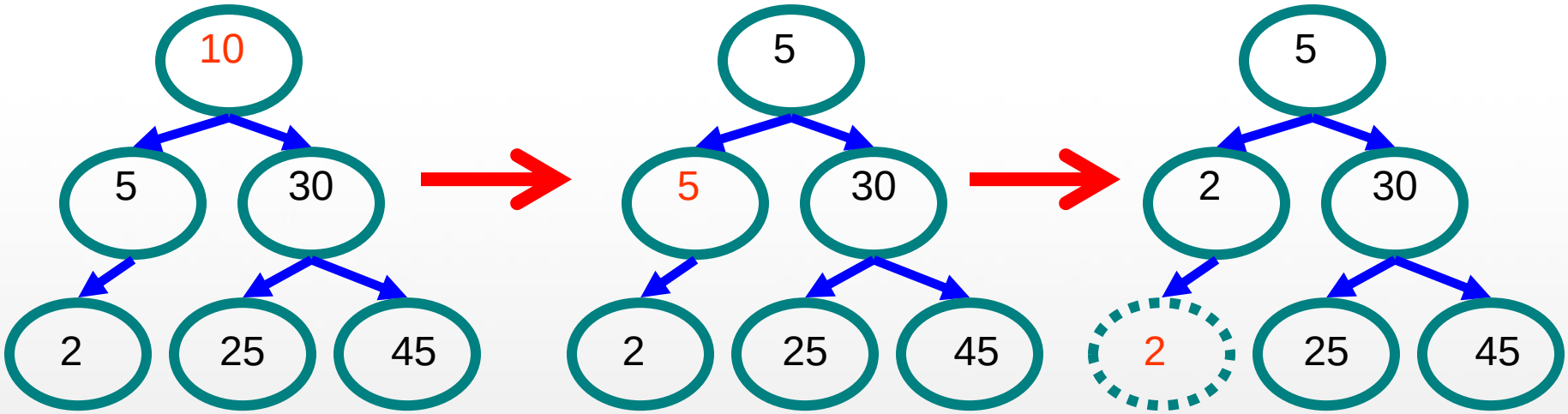
30 > 25, left

25 = 25, delete



Binary Search Tree : Delete

- Delete (10)



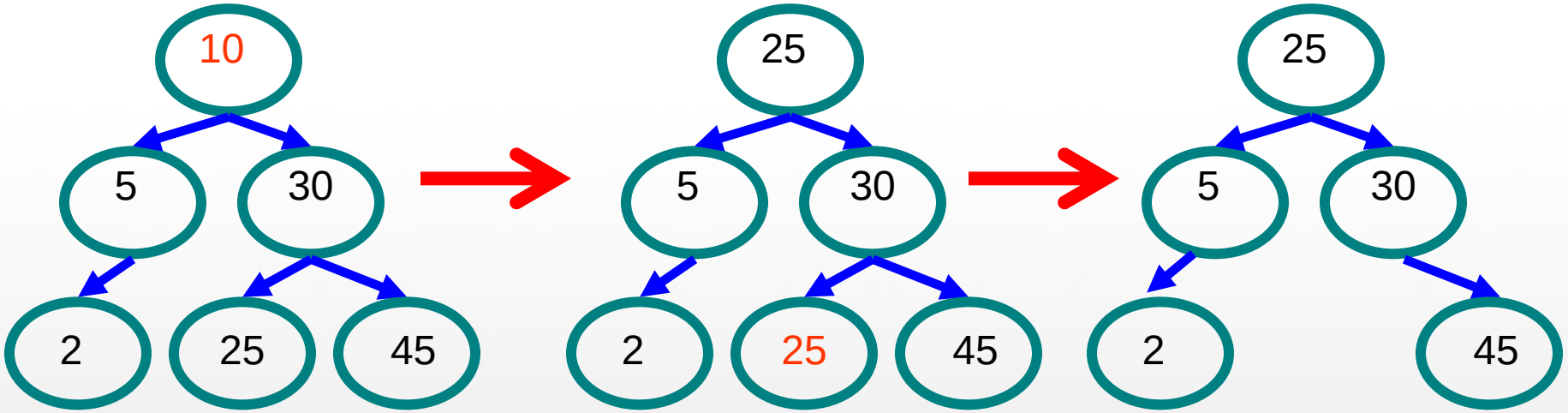
Replacing 10 with largest value in left subtree

Replacing 5 with largest value in left subtree

Deleting leaf

Binary Search Tree : Delete

- Delete (10)

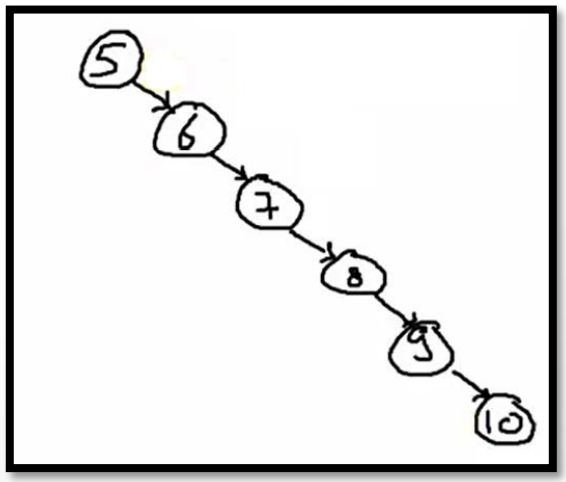


Replacing 10 with smallest value in right subtree

Deleting leaf

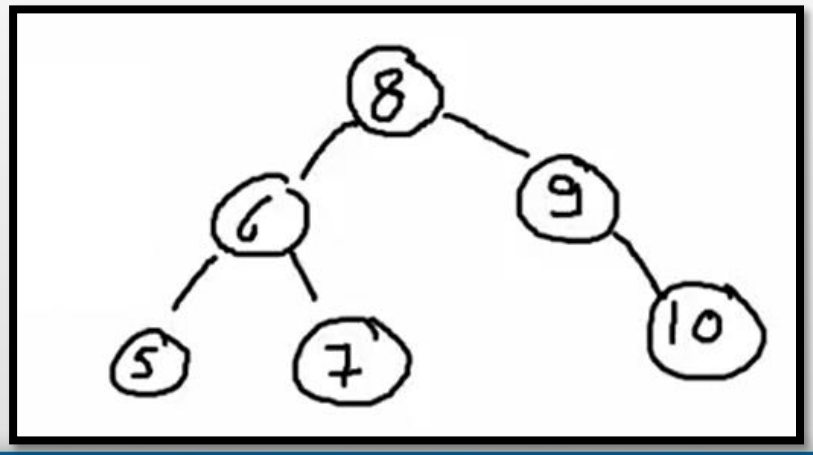
Resulting tree

Time complexity of the binary search tree.



T(n) = O (n) for Search, Insert, and Delete

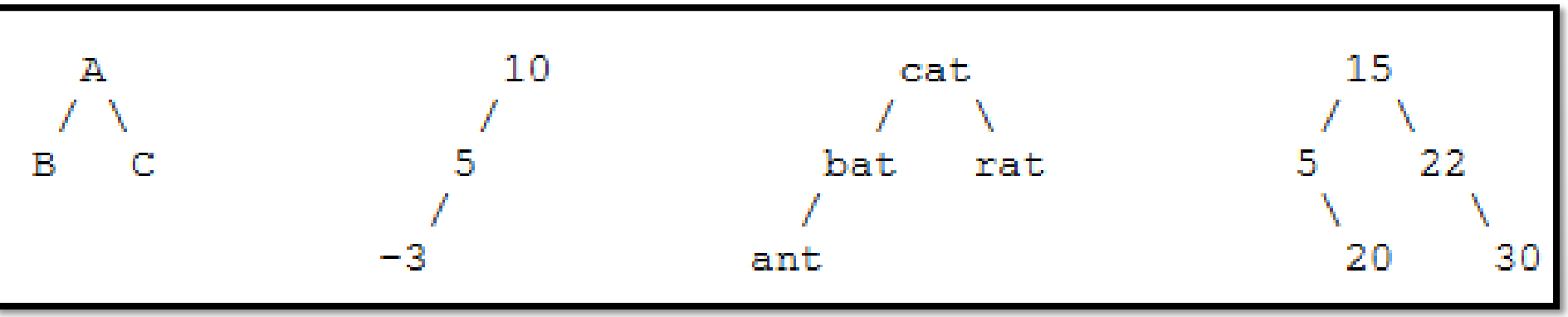
Like linked list



T(n) = O (log n) for Search, Insert, and Delete

Extra Exercises

- Which of the following binary trees are BSTs? If a tree is **not** a BST, say why.



- Using which kind of traversal (preorder, postorder, inorder, or level-order) visits the nodes of a BST in sorted order?

Question?

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