Contents :-

- Ordered Dictionaries
- Implementation
- Binary Search Tree
- Searching in BST
- Minimum in BST
- Maximum in BST
- Insertion in BST

Ordered Dictionaries :-

- In addition to the dictionary functionality, We want to support following Operations :-

- Min()
- Max()
- Predecessor(S,k)
- Successor(S,k)

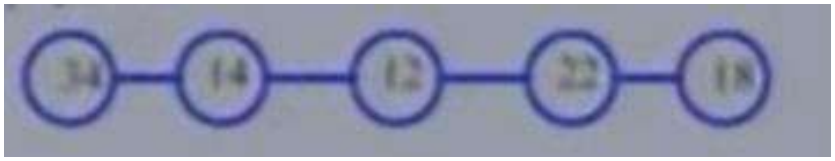
A List-Based Implementation :-

- Unordered List

  Searching takes O(n) times.

  Inserting takes O(1) times.

  Successor takes O(n) times.



- Ordered List

  Searching takes O(n) times.

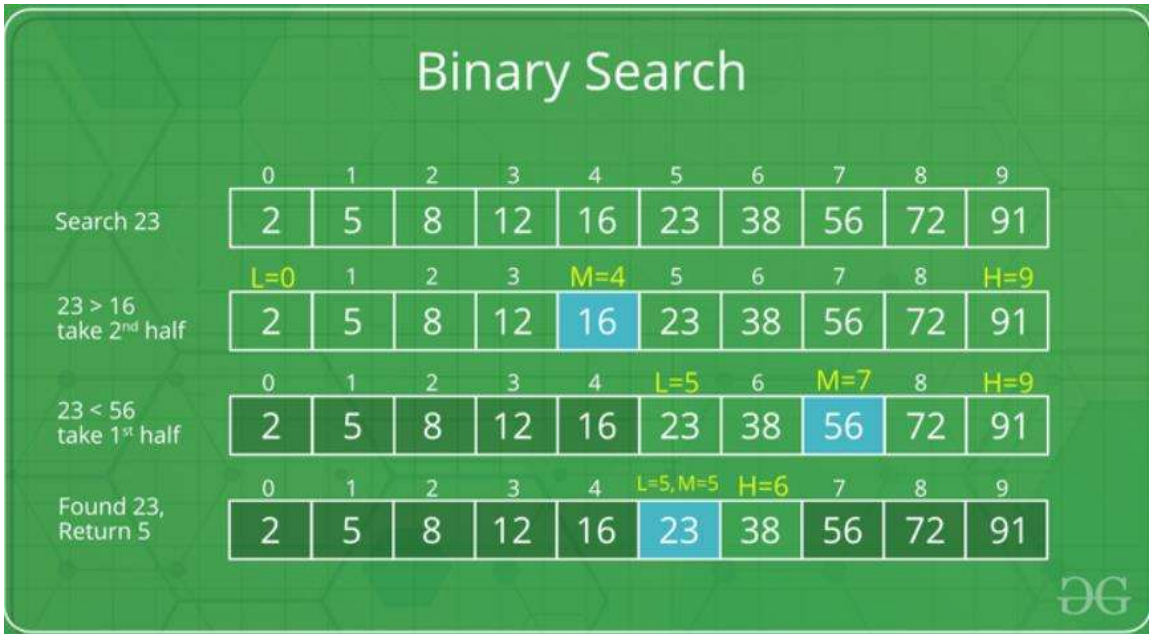
  Inserting takes O(n) times.

  min, max takes O(1) times.

- Using Array we can improve the Searching time.

Binary Search :-

- findElement(23)



- What about insertion and deletion ?

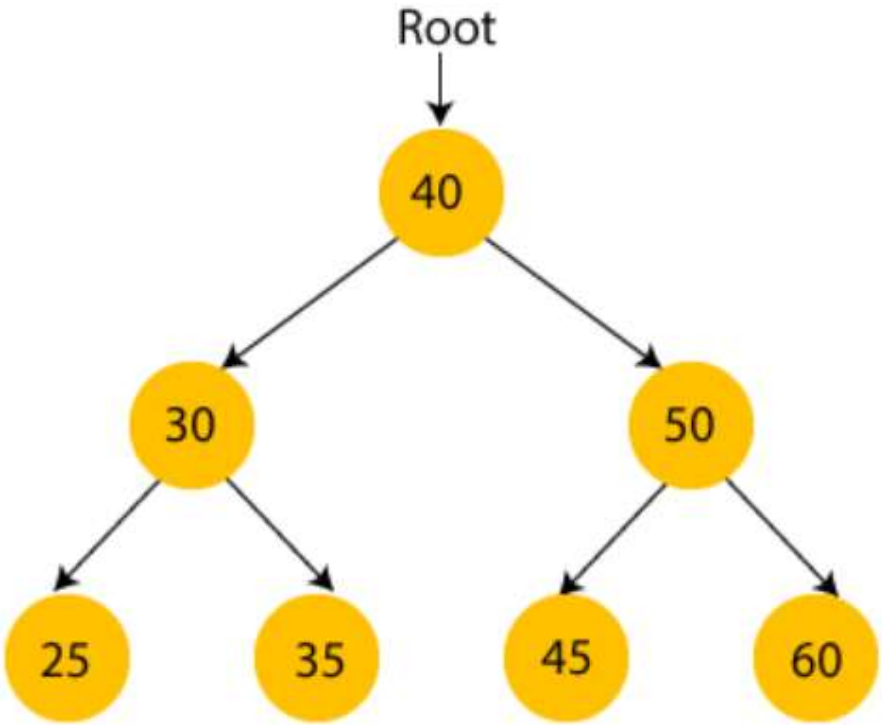
Binary Search Tree :-

- A Binary Search tree is a binary tree with searching ability.
- A binary search tree follows some order to arrange the elements.
- The value of left node must be smaller than the parent node.
- The value of right node must be greater than the parent node.
- This rule is applied recursively to the left and right subtrees of the root.

Binary Search Tree:-

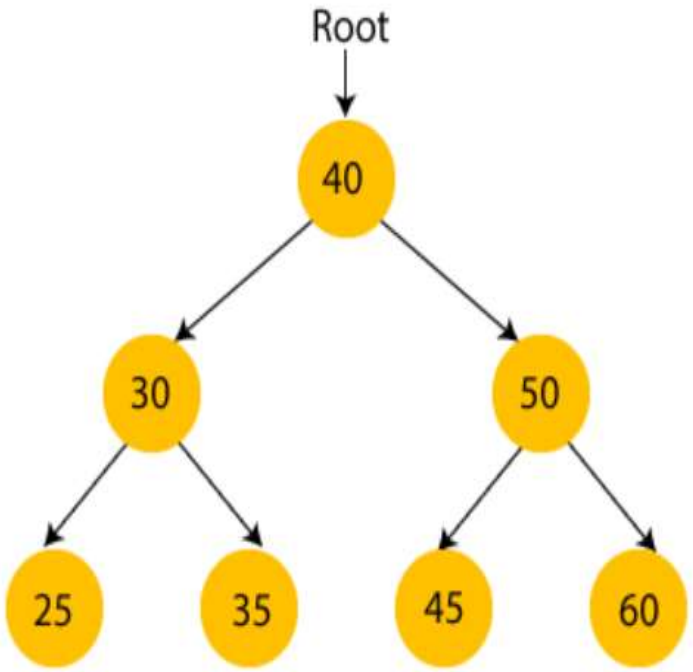
- Example sequence

  40,30,50,25,35,45,60



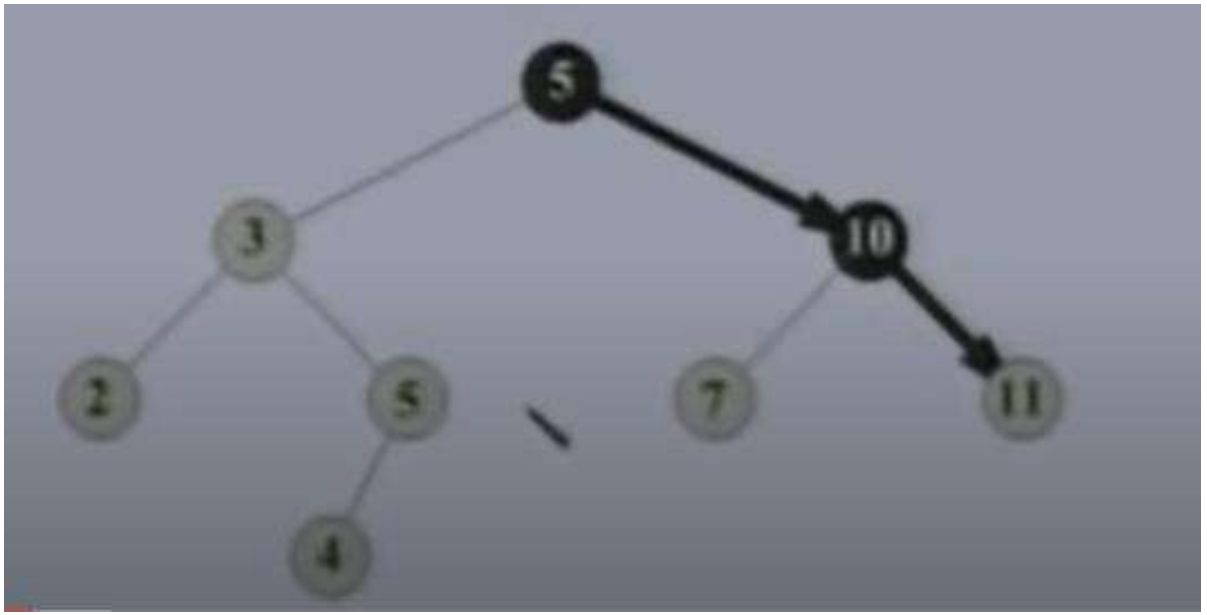
Searching in BST :-

- To find element with key k in a Tree T
- Compare k with key[root[T]]
- If k< key[root[T]], Search for k in left[root[T]]
- Otherwise , Search for k in right[root[T]]
- Searching takes O(h), Where h is the height of the binary tree.



Search Example :-

- Search(T,11)



Algorithm for BST Search:-

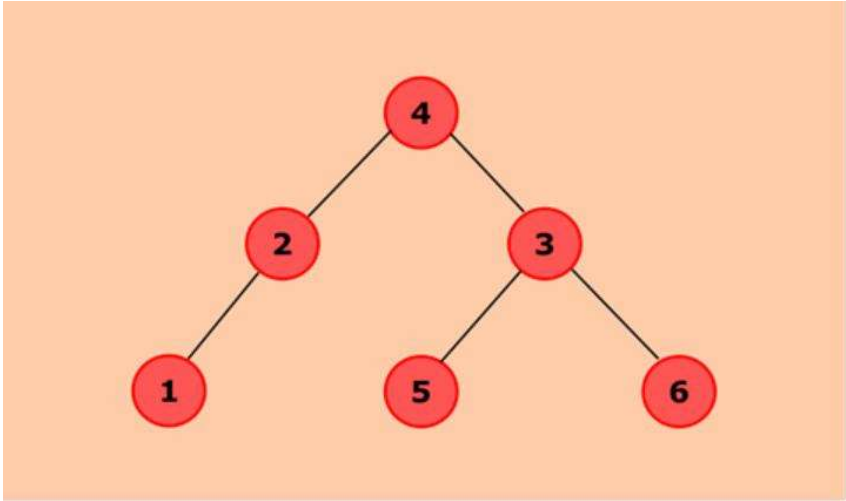
```
Search (root, item)
Step 1 - if (item = root → data) or (root = NULL)
return root
else if (item < root → data)
return Search(root → left, item)
else
return Search(root → right, item)
END if
Step 2 - END
```

BST Minimum:-

- How to find min element in BST ?

```
TreeMinimum(x)
 while left[x] != NULL
 do x <- left[x]
 return x ;
```

- Running Time O(h),i.e. proportional to the height of tree.

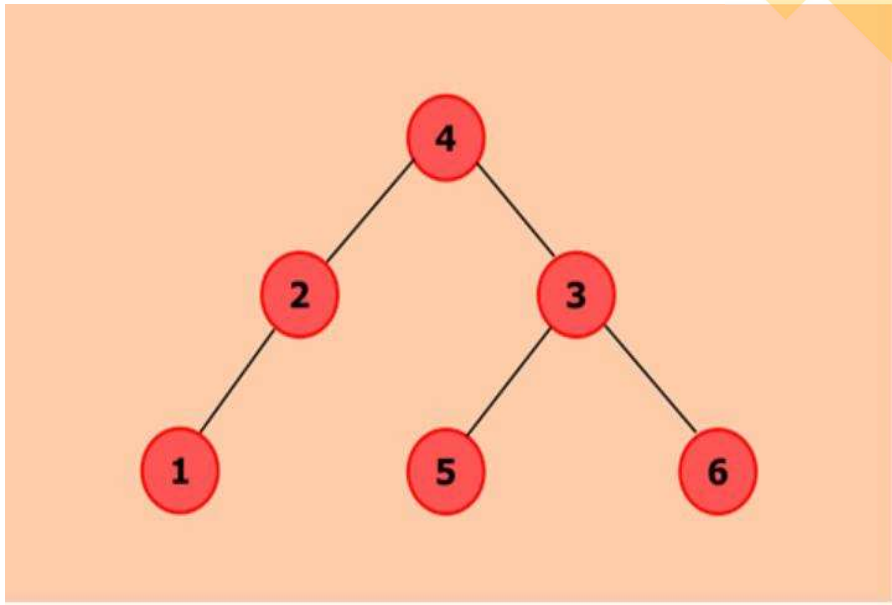


BST maximum :-

- How to find min element in BST ?

```
TreeMaximum(x)
 while right[x] != NULL
 do x <- right[x]
 return x ;
```

- Running Time O(h),i.e. proportional to the height of tree.



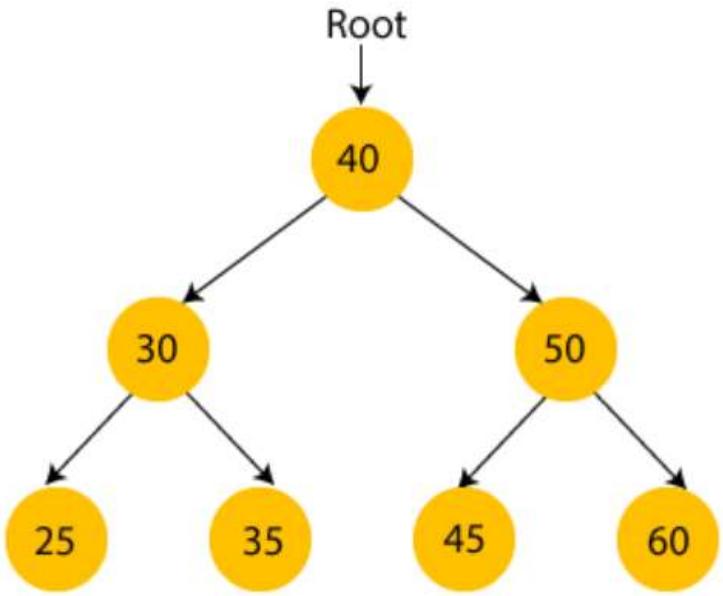
Successor:-

- Given x, find the node with the smallest key greater than key[x]
- We can distinguish two cases, depending on the right subtree of x
- Case -1

  right subtree of x is non empty.

  Successor will be left most node of the

  right subtree.



Successor :-

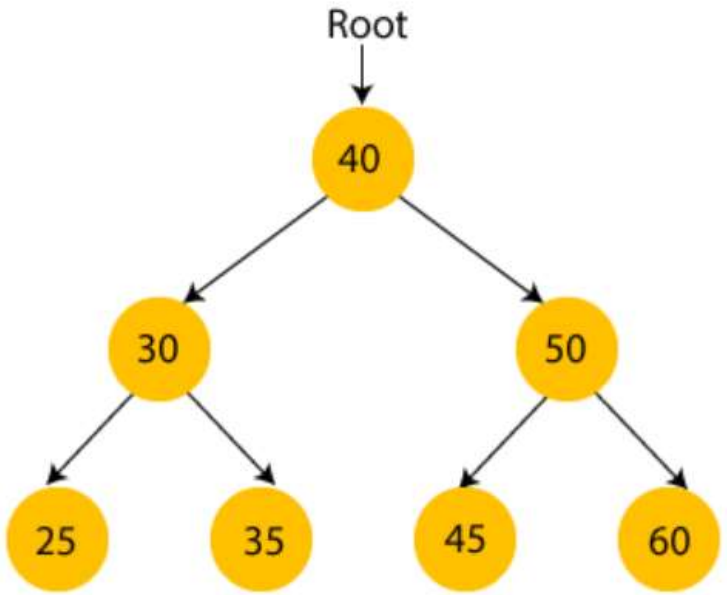
- Case -2

  The right sub tree of x is empty.

  Successor is the lowest ancestor of x

  whose left child is also an ancestor of x

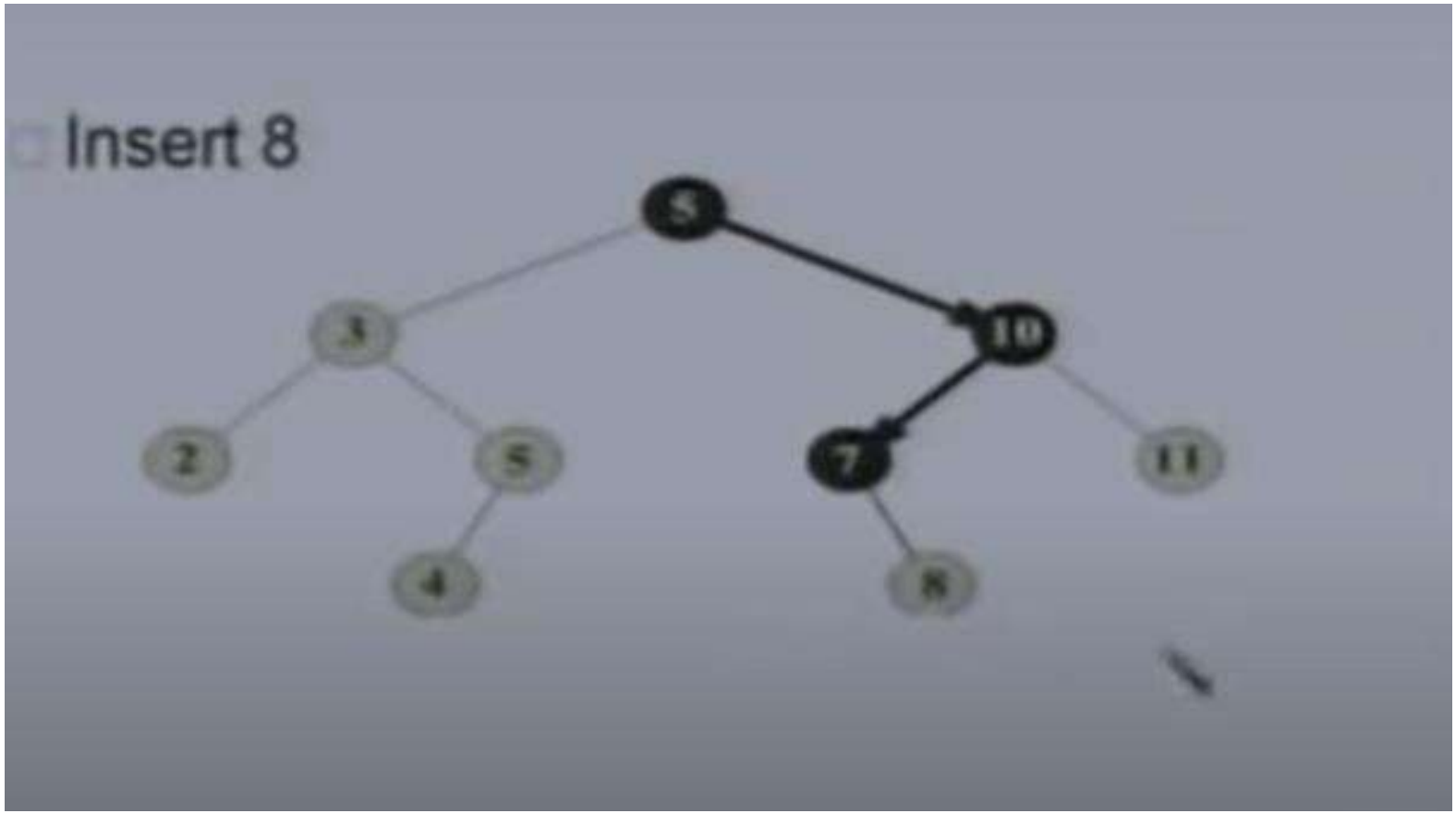
  (Why ?)



BST Insertion:-

- The basic idea is similar to searching.
- take an element k (whose left and right child are null).
- Find place in T where k belongs to just like searching for k.
- And k there.
- The Running on a tree of height h is O(h).
- It is proportional to the height of tree.

BST Insert :-



BST Insertion Algorithm:-

```
insert (Node n, int x)
{
     if (n==null)
     {
          n = new Node(x)
     }
     if(n.data < x)
          insert(n.right, x)
     else if(n.data > x)
          insert(n.left, x)
}
```

Thank you