CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap: Properties of Binary Tree

- Minimum height *h* of binary tree with n nodes?
- Maximum height *h* of binary tree with *n* nodes? General Case: $\log (n+1) -1 \leq h \leq n-1$
- Minimum number of external nodes in a tree of height *h*?
- Maximum number of external nodes in a tree of height *h*? General case: $1 \leq n_{ext} \leq 2^h$
- Minimum number of internal nodes in a tree of height *h*?
- Maximum number of internal nodes in a tree of height *h*? General case: $1 \leq n_{int} \leq 2^h - 1$

Exercise: Binary Tree (1)

In the binary tree, the number of internal nodes of degree 1 is 1, and the number of internal nodes of degree 2 is 2. The number of leaf nodes in the binary tree is:

Exercise: Binary Tree (2)

In the binary tree, the number of internal nodes of degree 2 is 15, and the number of internal nodes of degree 1 is 7. The number of leaf nodes in the binary tree is:

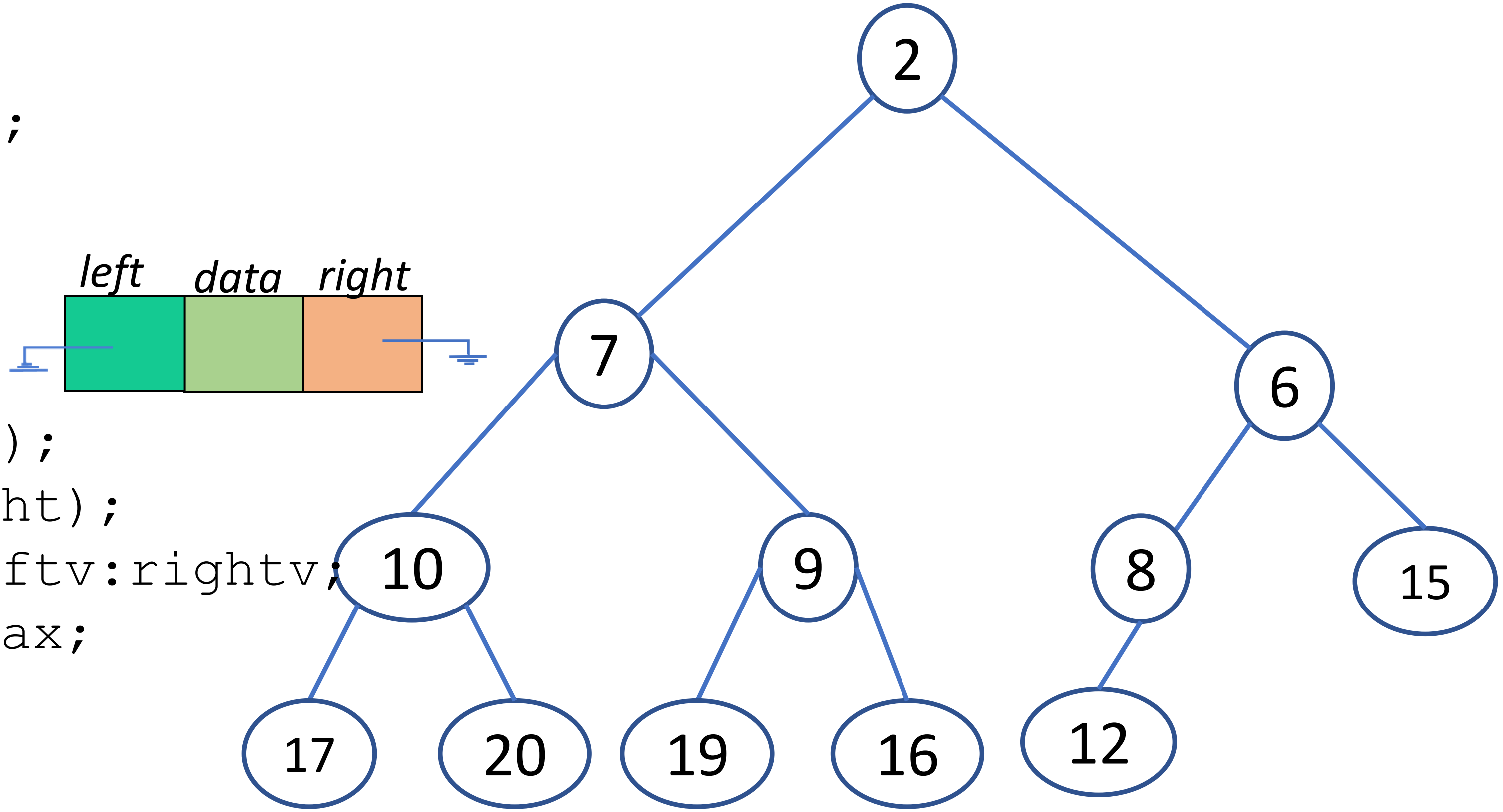
(A) 11
(B) 14
(C) 15
(D) 16

Motivation → Binary Search Tree

- *Auxiliary Operations: Find a <u>minimum</u> and <u>maximum</u> number from the given binary tree*

```
struct btree {
      struct btree *left;
      int data;
    struct btree *right; }
```

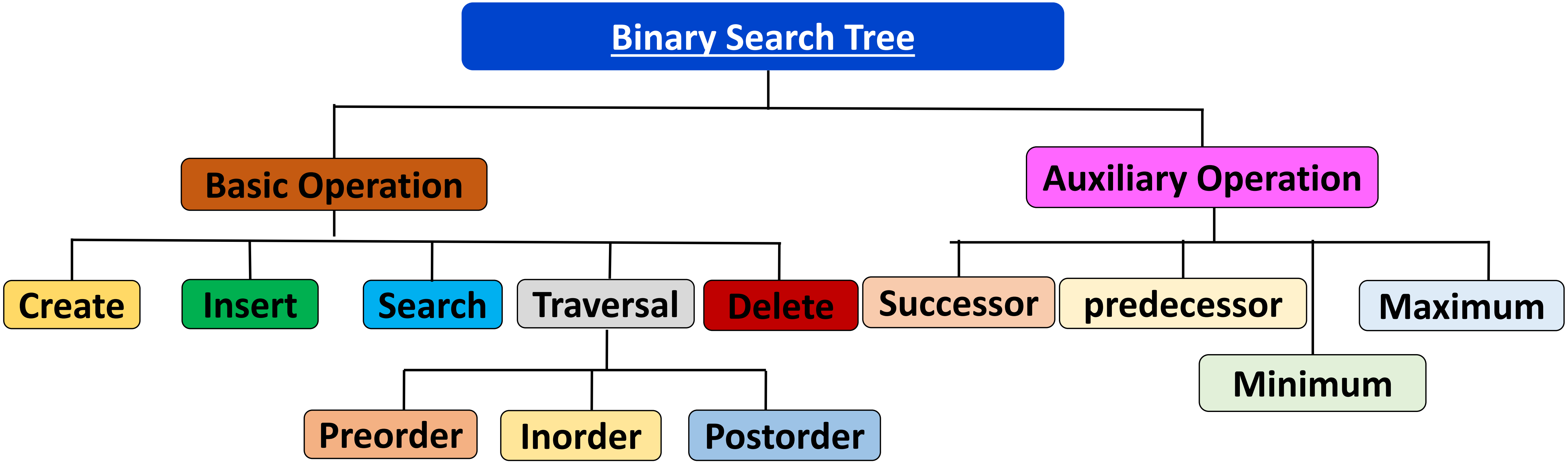
```
int FindMax(struct btree *root){
    int rval, leftv, rightv, max;
    struct btree *fmax;
    fmax=root;
    if (fmax!=NULL){
        rval=fmax->data;
        leftv=FindMax(fmax->left);
        rightv=FindMax(fmax->right);
        max= (leftv > rightv)?leftv:rightv;
        max= (rval > max)?rval:max;
    }
   return max;
}
```



btree → <u>binary tree</u>

Time Complexity O (*n*) n: # of nodes in a binary tree

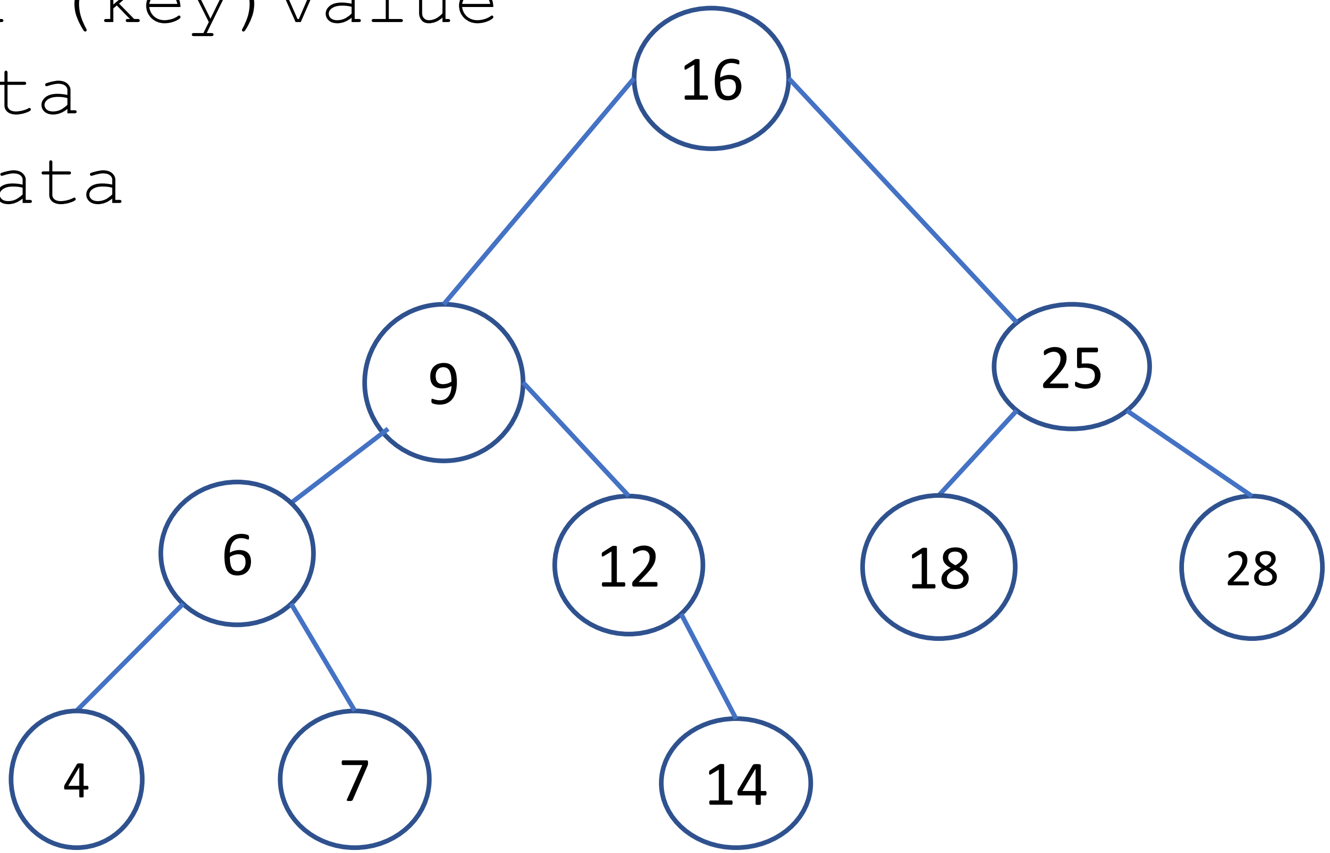
Binary Search Tree



Binary Search Tree: Properties

```
struct BSTree {
              struct BSTree *left;
              int data;
            struct BSTree *right;}
```

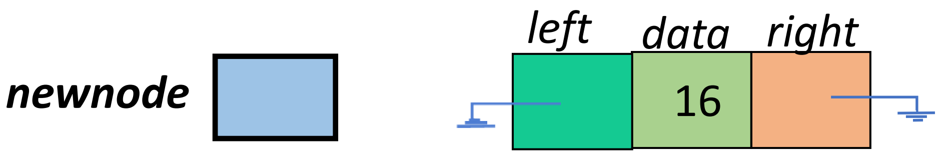
- Each node contains a data (key)value
- Left Subtree < root → data
- Right Subtree > root → data

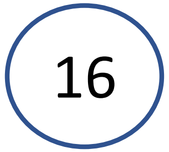


Binary Search Tree: Create

Steps:

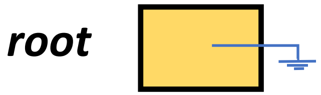
(i) Creating a node with data





```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
}
```

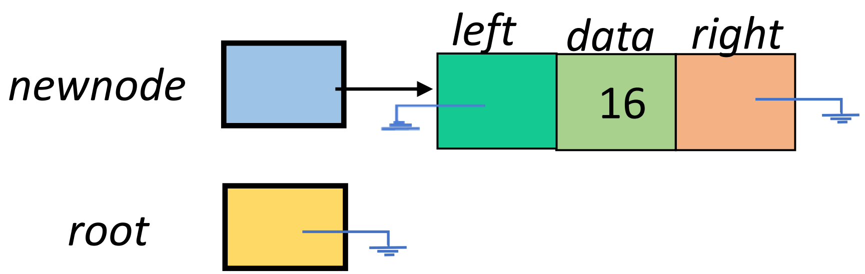
```
struct BSTree *newnode = malloc (sizeof (struct BSTree));
newnode → left = NULL;
newnode → data = 16;   //Entering data
newnode → right = NULL; //making node next to NULL
```



*struct node *root = NULL*

(ii) Adding a new node to an empty BST

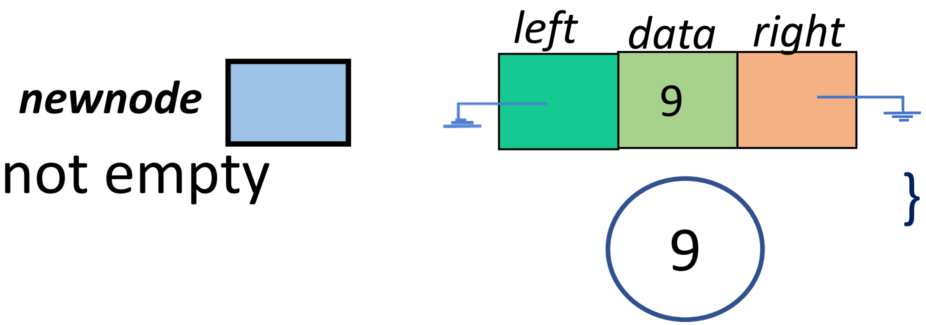
```
if (root == NULL)
    root = newnode
```



Binary Search Tree: Insert

```
struct BSTree {
          struct BSTree *left;
          int data;
        struct BSTree *right;
}
```

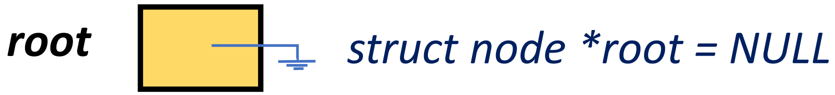
Steps:

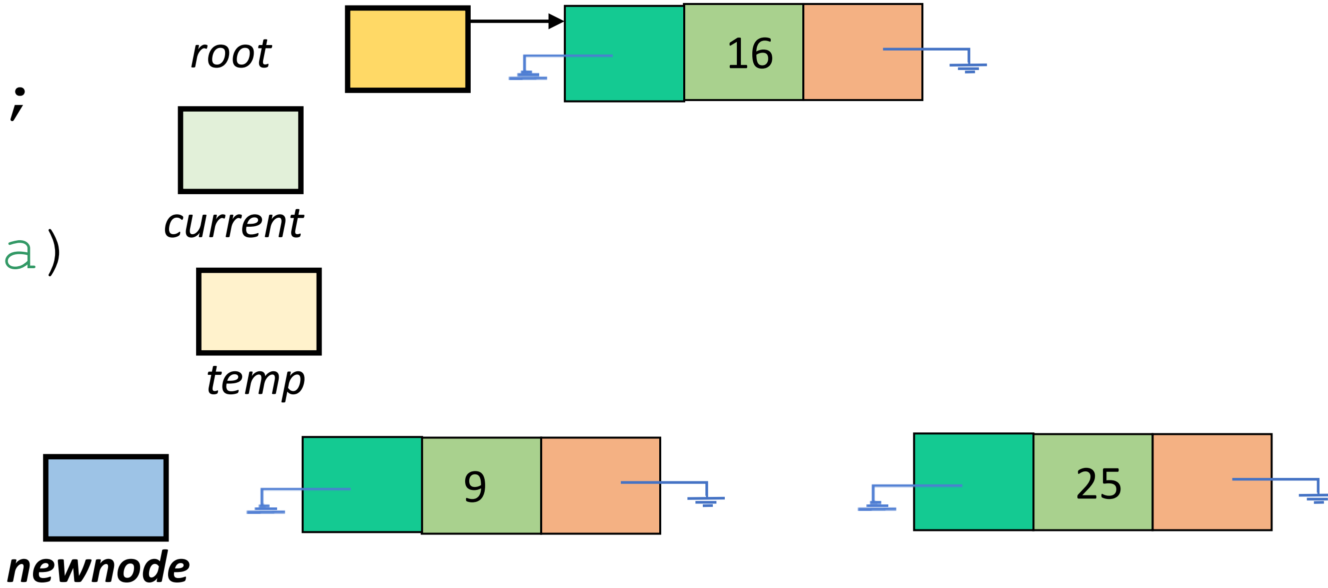


(iii) Adding new node when BST is not empty

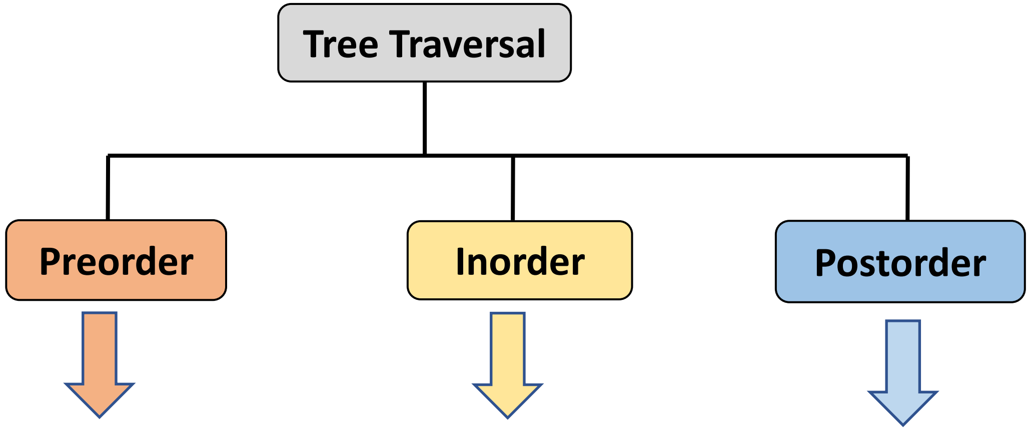
current = root;

```
while(current!=NULL){
      temp = current;
      if(current->data > newnode->data)
        current = current->left;
      else
        current = current->right;
    } //end of while loop
    if(temp->data > newnode->data)
      temp-> left = newnode;
    else
      temp-> right = newnode;
```





Binary Search Tree: Traversal



Travel Policies →

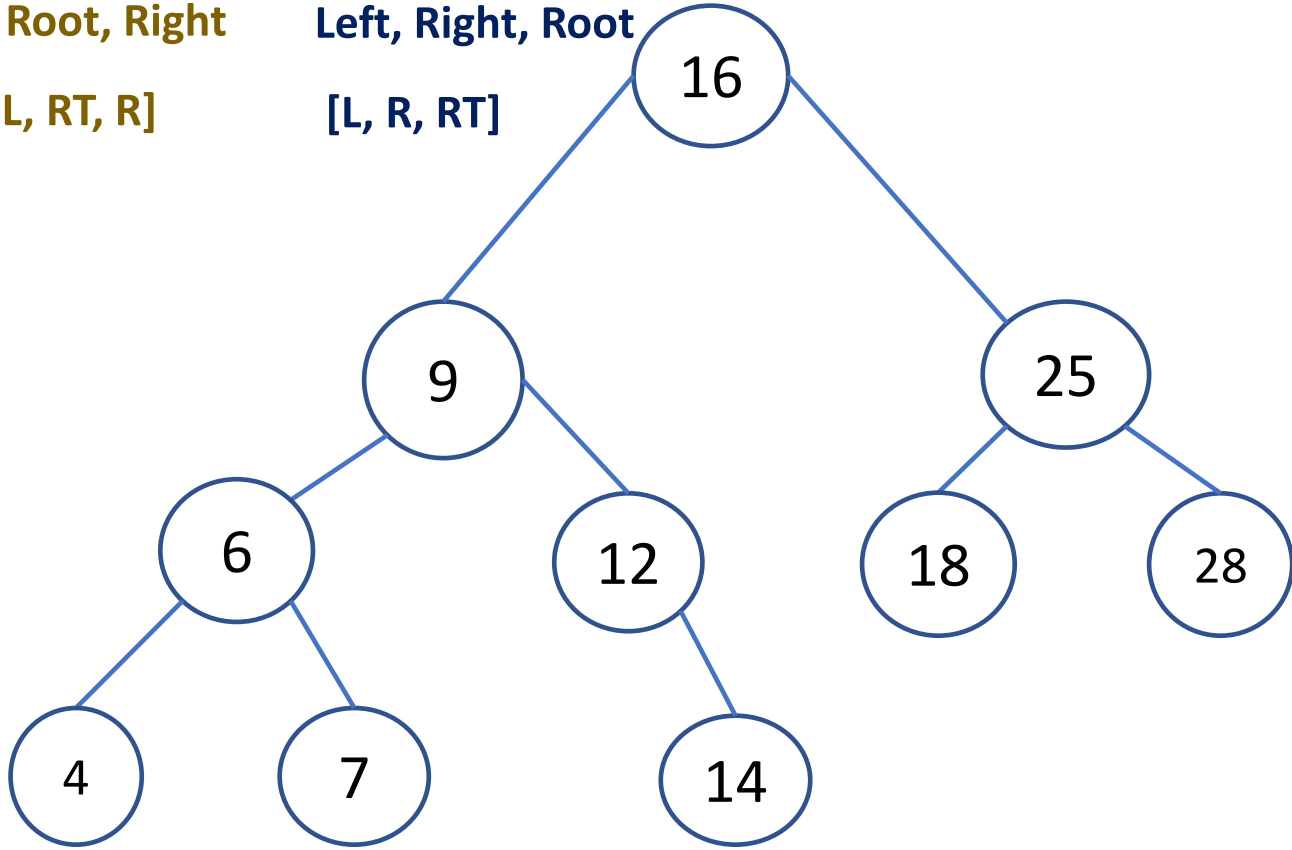
Preorder	Inorder	Postorder
Root, Left, Right	**Left, Root, Right**	**Left, Right, Root**
[RT, L, R]	**[L, RT, R]**	**[L, R, RT]**

```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
}
```

Preorder → 16, 9, 6, 4, 7, 12, 14, 25, 18, 28

Inorder →

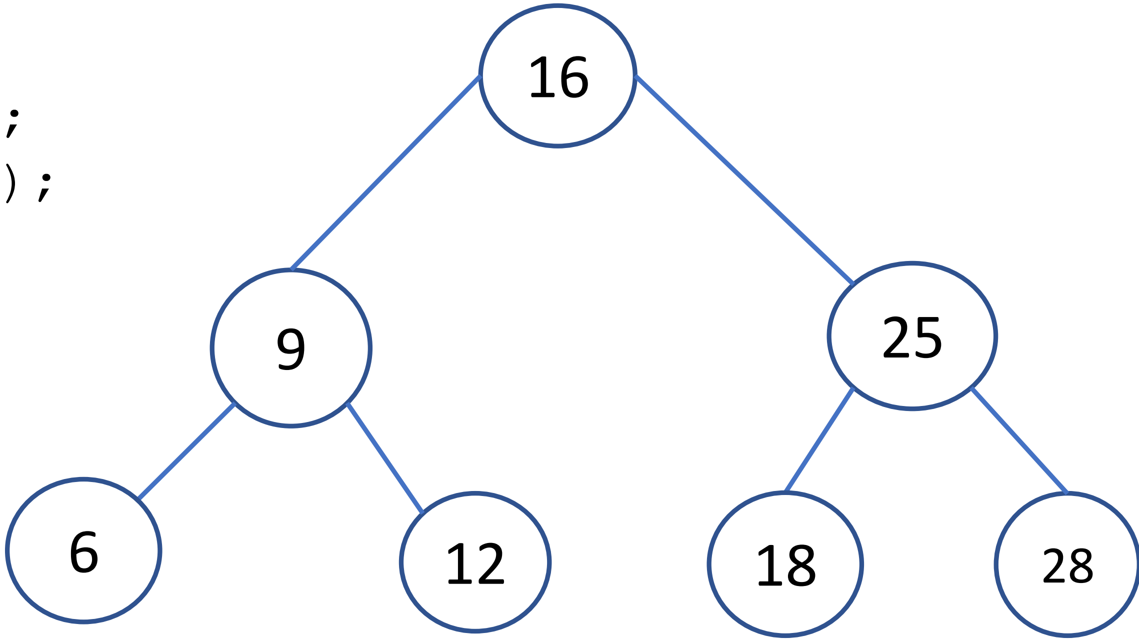
Postorder → 4, 7, 6, 14, 12, 9, 18, 28, 25, 16



Binary Search Tree: PreOrderTraversal

```
PreOrderTraversal(root); //calling preordertraversal

void PreOrderTraversal(struct BTree *preordertravel)
{
    if(preordertravel!= NULL){
    printf("%d ", preordertravel->data);
    PreOrderTraversal(preordertravel->left);
    PreOrderTraversal(preordertravel->right);
  }
}
```

```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
   }
```

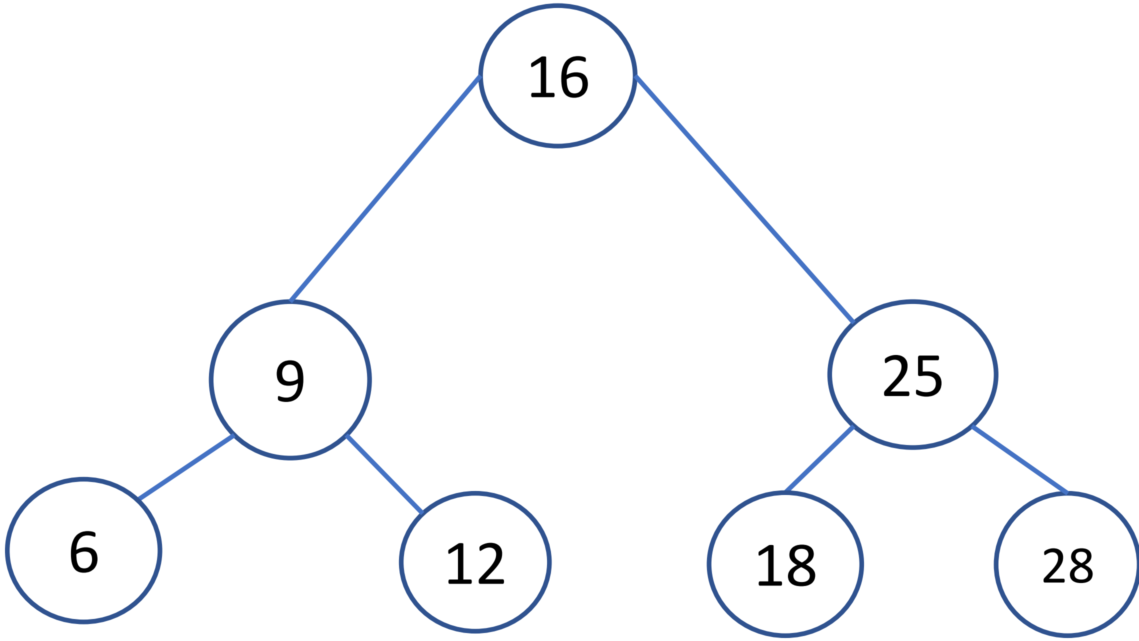


Preorder → 16, 9, 6, 12, 25, 18, 28

Binary Search Tree: InOrderTraversal

```
InOrderTravesal(root); //calling inordertraversal

void InOrderTraversal(struct BTree *inordertravel)
{
  if(inordertravel!= NULL){
    InOrderTraversal(inordertravel->left);
    printf("%d ", inordertravel->data);
    InOrderTraversal(inordertravel->right);
  }
}
```

```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
   }
```

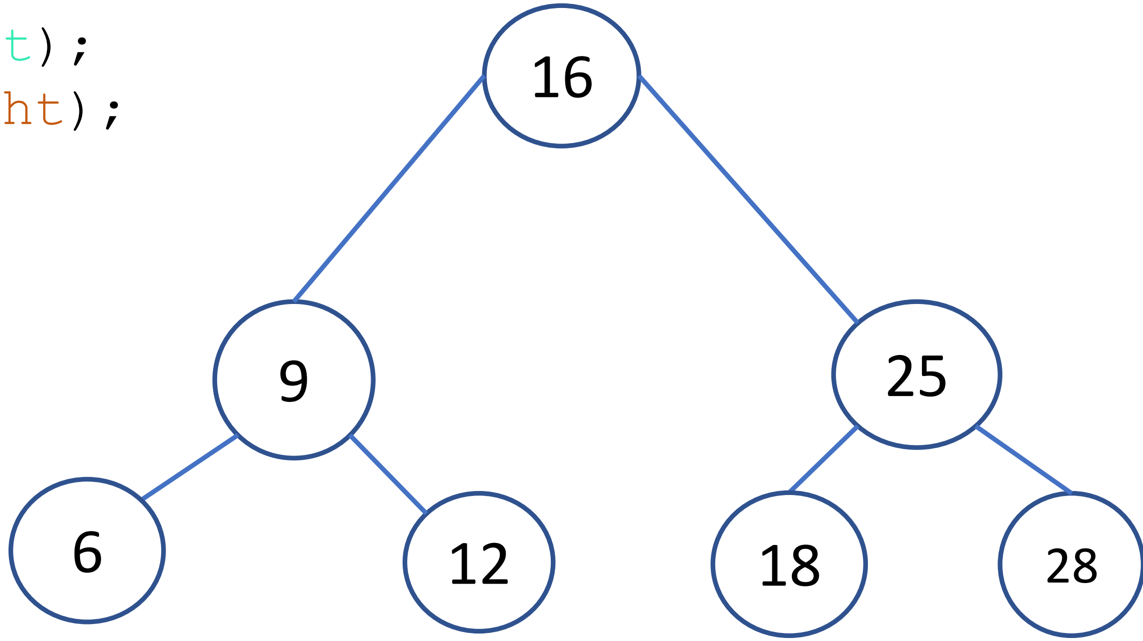


Inorder → 6, 9, 12, 16, 18, 25, 28

Binary Search Tree: PostOrderTraversal

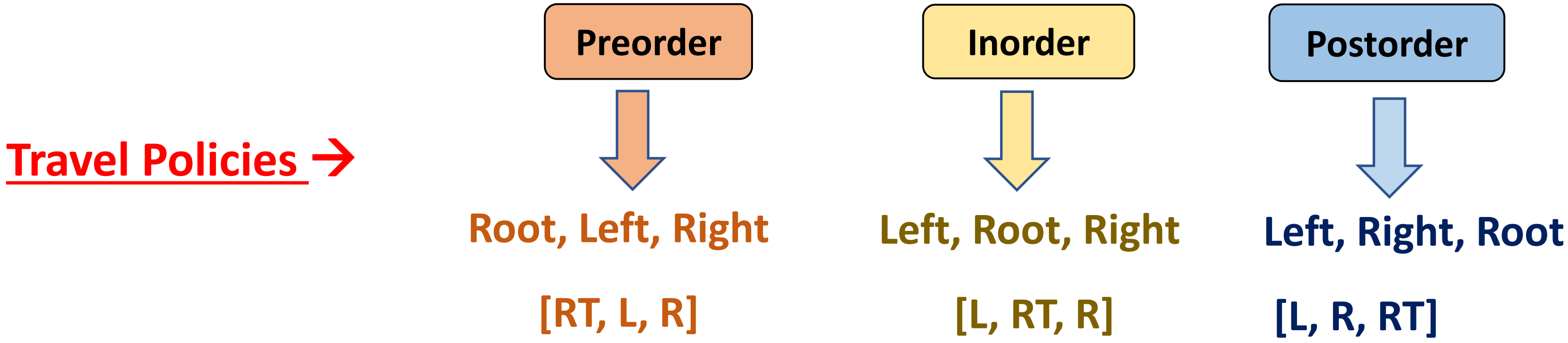
```
PostOrderTravesal(root); //calling postordertraversal

void PostOrderTraversal(struct BTree *postordertravel)
{
  if(postordertravel!= NULL){
    PostOrderTraversal(postordertravel->left);
    PostOrderTraversal(postordertravel->right);
    printf("%d ", postordertravel->data);

  }
}
```

```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
   }
```



Postorder → 6, 12, 9, 18, 28, 25, 16

Exercise: Binary Search Tree Traversal (1)



Travel Policies →

Preorder	Inorder	Postorder
Root, Left, Right	Left, Root, Right	Left, Right, Root
[RT, L, R]	[L, RT, R]	[L, R, RT]

The preorder traversal of a binary search tree is: 12, 8, 6, 2, 7, 9, 16, 15

What is the postorder traversal of a binary search tree is:

(A) 2, 6, 7, 8, 9, 12, 15, 16

(B) 2, 7, 6, 9, 8, 15, 16, 12

(C) 7, 2, 6, 8, 9, 15, 16, 12

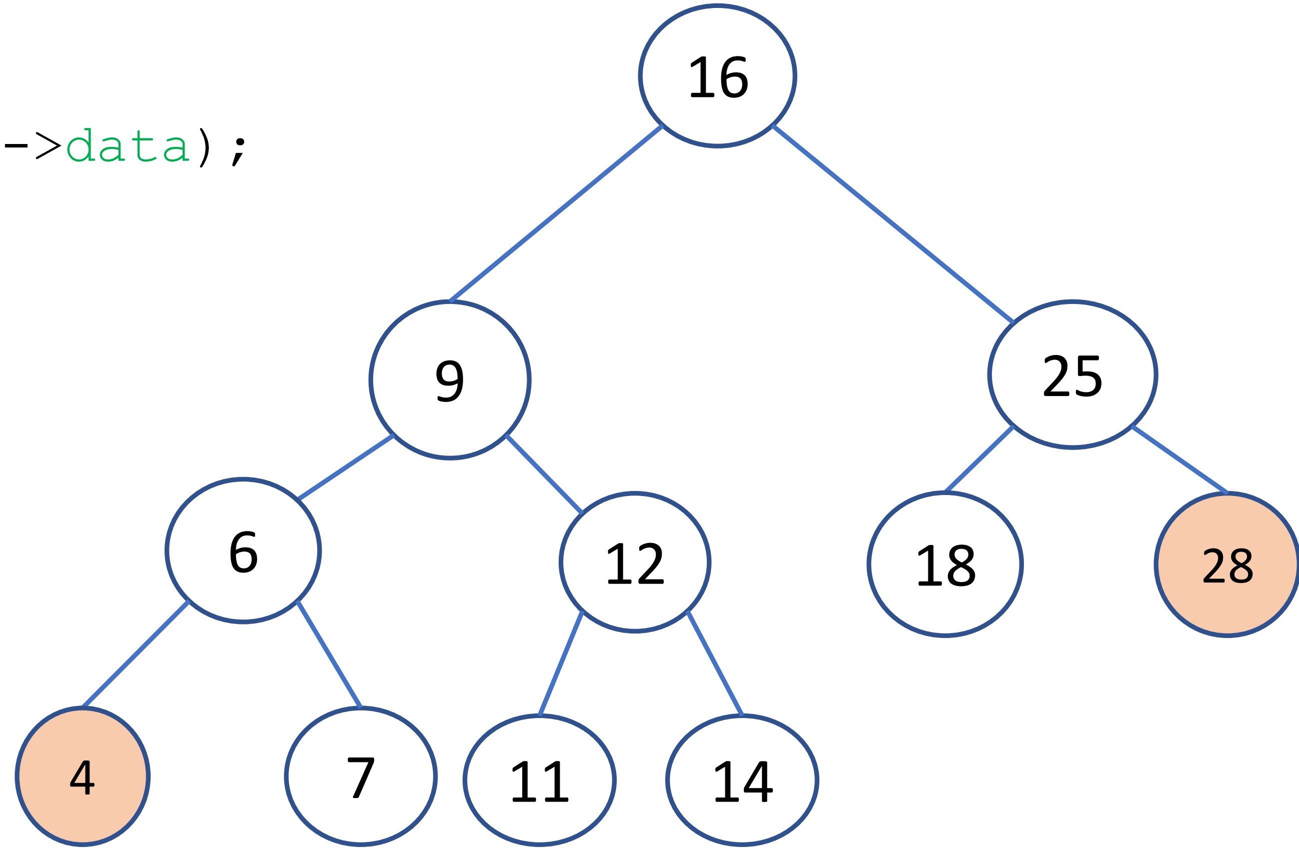
(D) 7, 6, 2, 9, 8, 15, 16, 12

Binary Search Tree: Search

```
Search(root, key);

void Search(struct BSTree* sear, int key)
{

 while(sear != NULL){
      if (key == sear -> data)
         printf(“Element found", sear->data);
         return;
      else if (key < sear -> data)
              sear = sear -> left;
      else
             sear = sear -> right;

  printf(“Element not found\n");
}
```

O (logn)

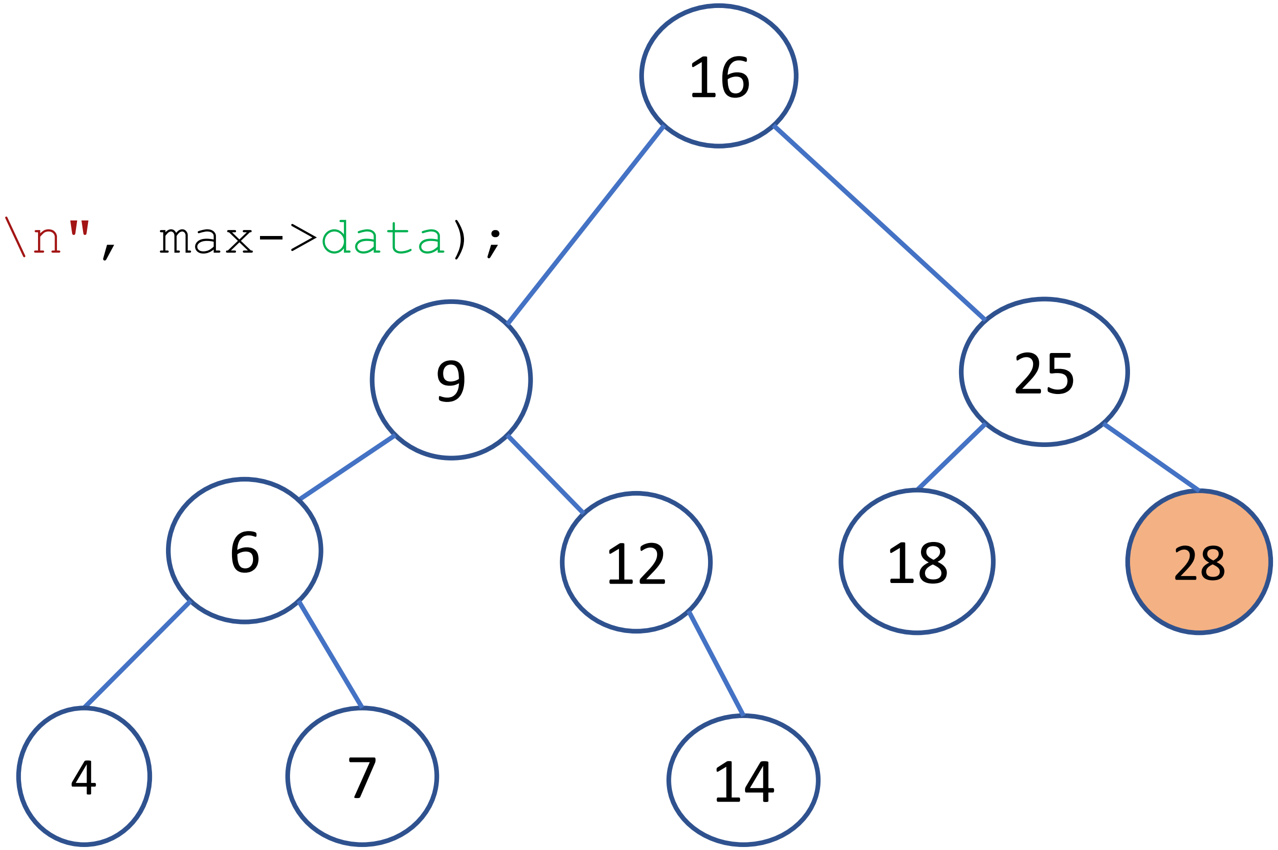
```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
   }
```



Binary Search Tree: Maximum

```
MaxValue(root);

Void MaxValue(struct BSTree* max)
{

   while(max -> right != NULL){
         max = max -> right;
   }
   printf("Maximum value in BST is: %d \n", max->data);
}
```

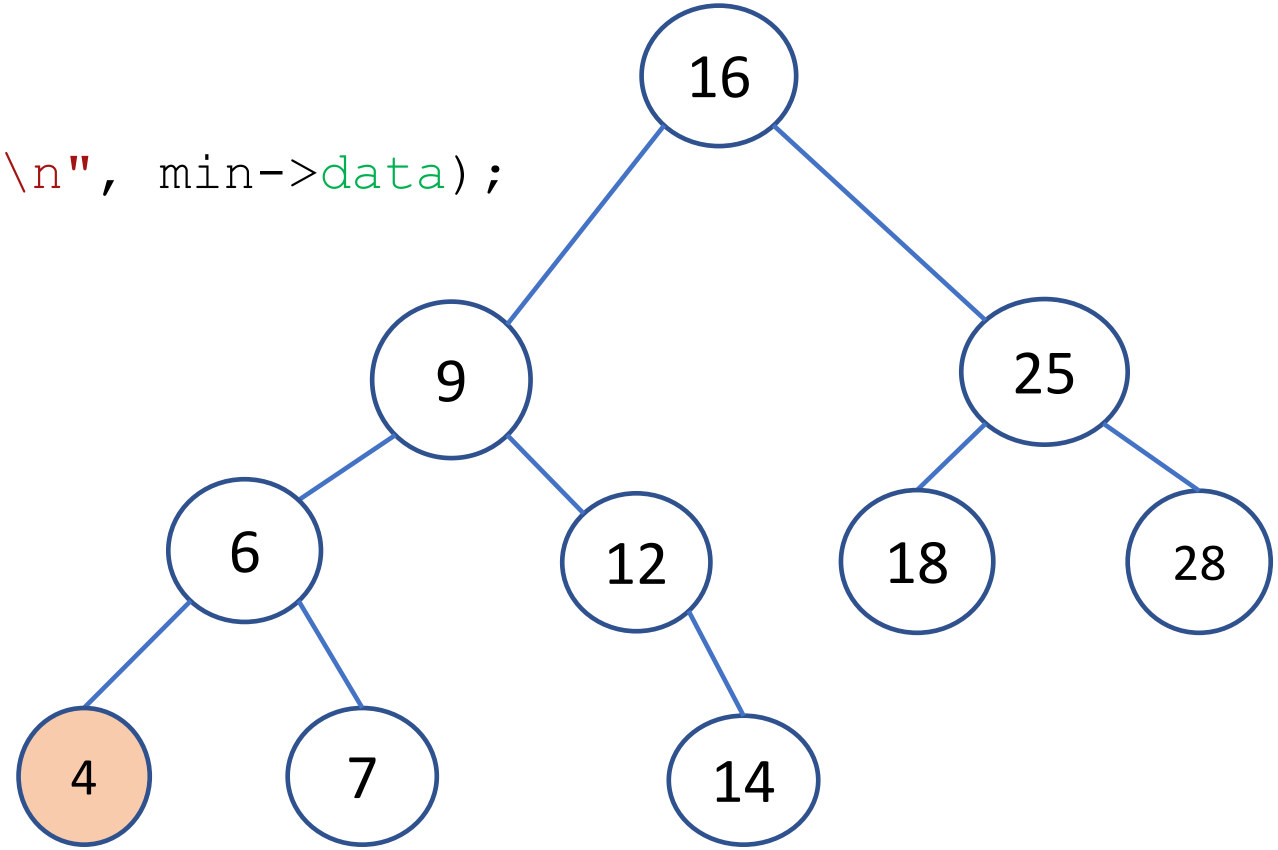
```
struct BSTree {
            struct BSTree *left;
            int data;
            struct BSTree *right;
    }
```



Binary Search Tree: Minimum

```
MinValue (root);

Void MinValue(struct BSTree* min)
{
  while(min -> left != NULL){
    min = min -> left;
  }
  printf("Minimum value in BST is: %d \n", min->data);


}
```

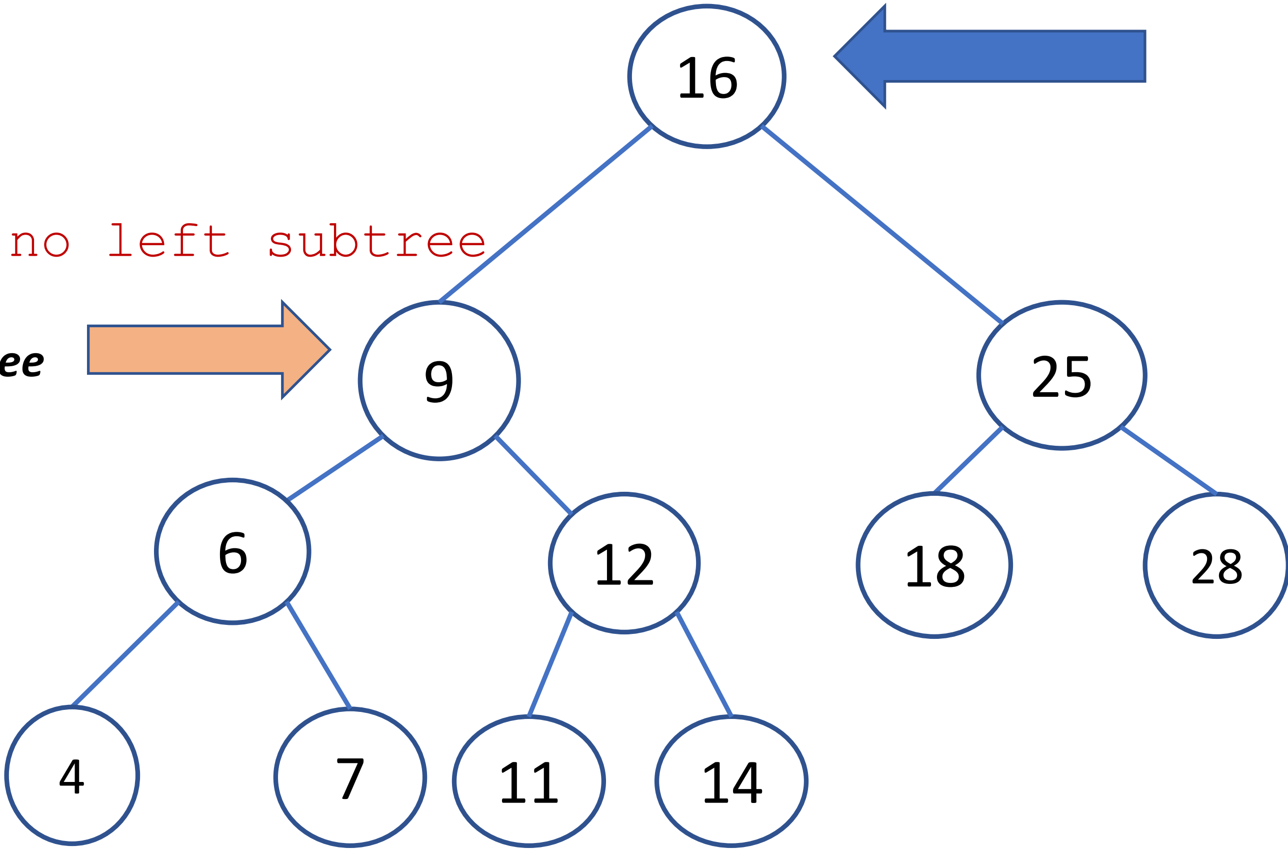
```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
   }
```



Binary Search Tree: Predecessor

```
Predecessor(root); //instead of root pass the whichever node
predecessor you wish to get

void Predecessor(struct BSTree* pred)
{

  if(pred -> left!= NULL){
      MaxValue(pred -> left);
    //printf("%d ", pred->data);
  // Here:
  // Required to handle when there is no left subtree
  }
}
```

```
struct BSTree {
    struct BSTree *left;
    int data;
    struct BSTree *right;
}
```



Maximum values in the Left Subtree

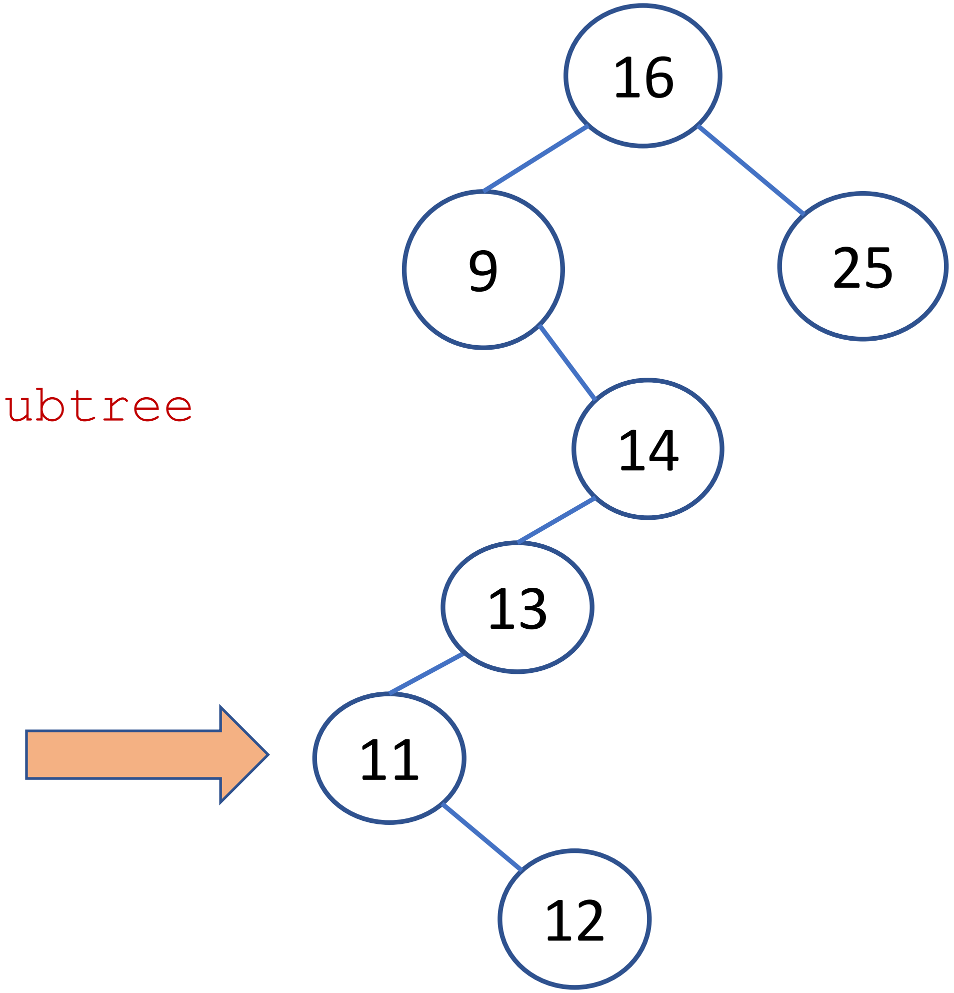
Binary Search Tree: Predecessor

```
struct BSTree {
    struct BSTree *left;
    int data;
    struct BSTree *right;
}
```

```
Predecessor(root); //instead of root pass the whichever node
predecessor you wish to get

void Predecessor(struct BSTree* pred)
{

  if(pred -> left!= NULL){
      MaxValue(pred -> left);
    //printf("%d ", pred->data);
  // Here:
  // Required to handle when there is no left subtree
  }
}
```

Maximum values in the Left Subtree

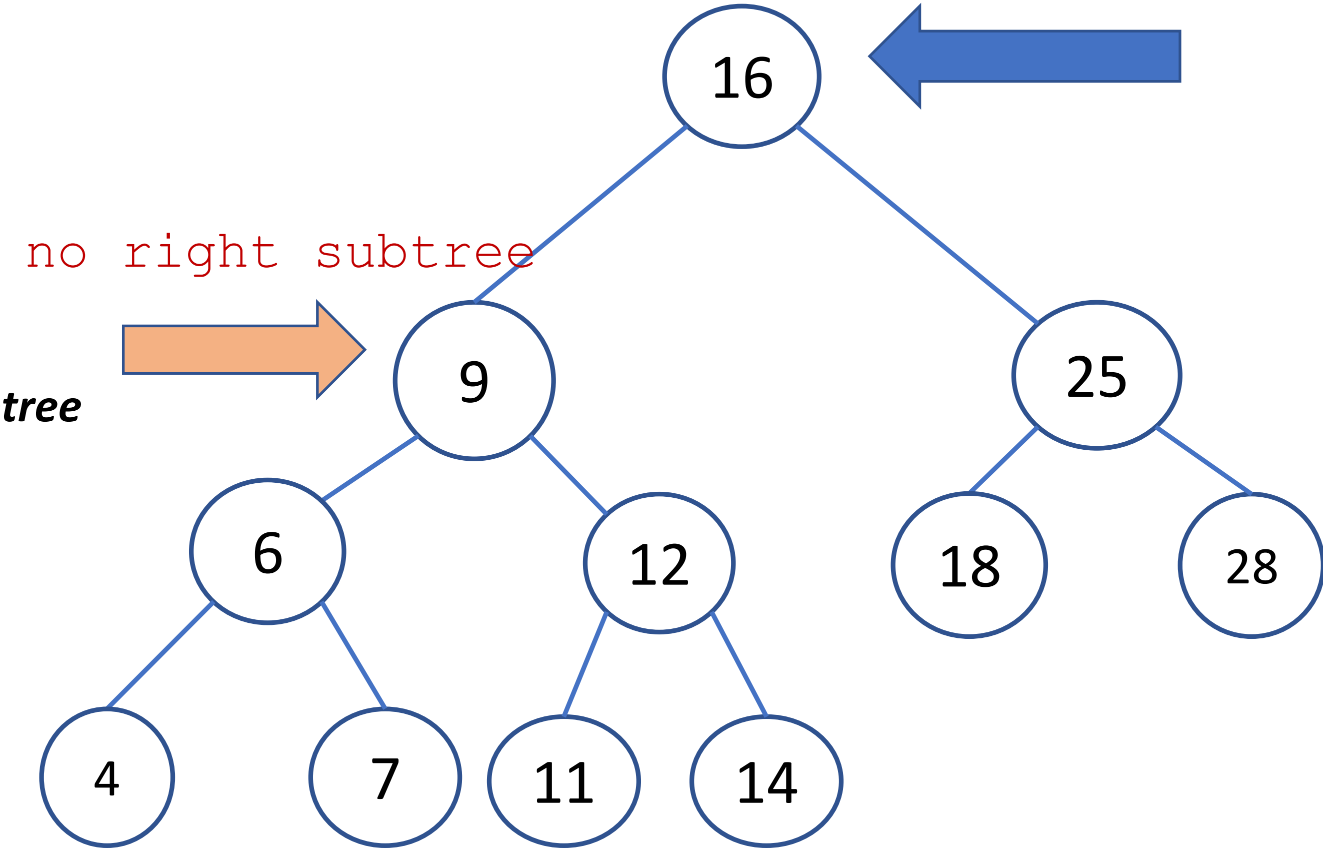
Predecessor is first left ancestor, if the nodes does not have left subtree



Binary Search Tree: Successor

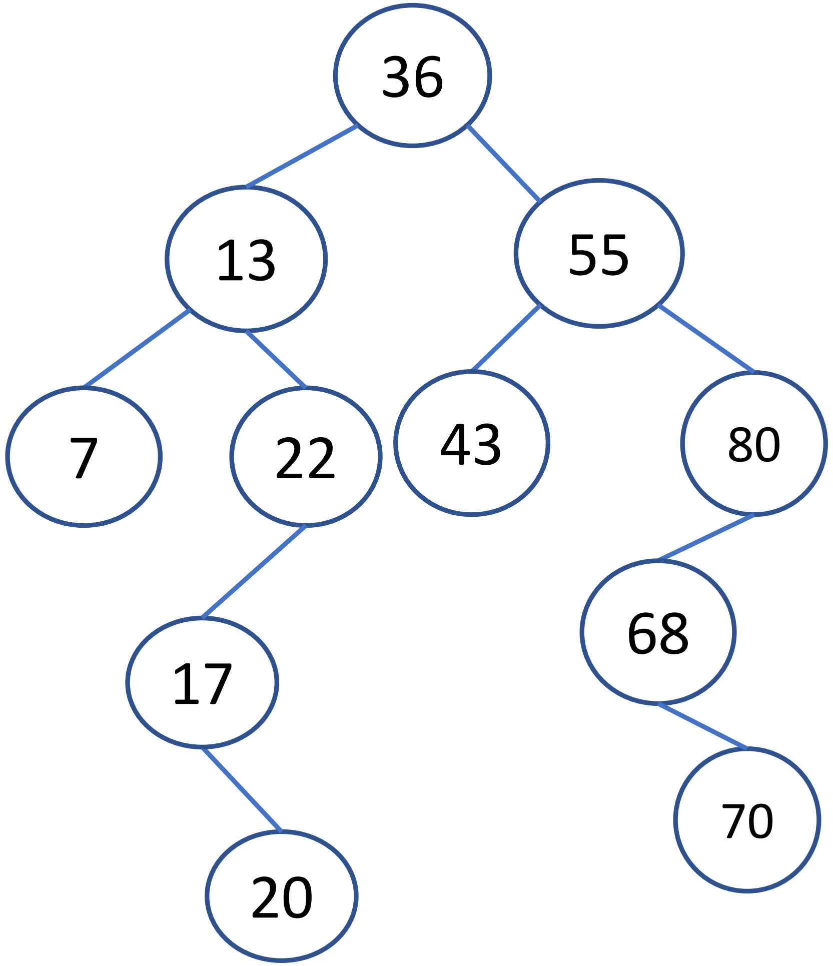
```
Successor(root); //instead of root pass the whichever node
successor you wish to get

void Successor(struct BSTree* suce)
{
  suce = root;
  if(suce -> right!= NULL){
      MinValue(suce -> right);
    //printf("%d ", suce->data);
  // Here:
  // Required to handle when there is no right subtree
  }
}
```

```
struct BSTree {
        struct BSTree *left;
        int data;
        struct BSTree *right;
}
```



Minimum values in the Right Subtree

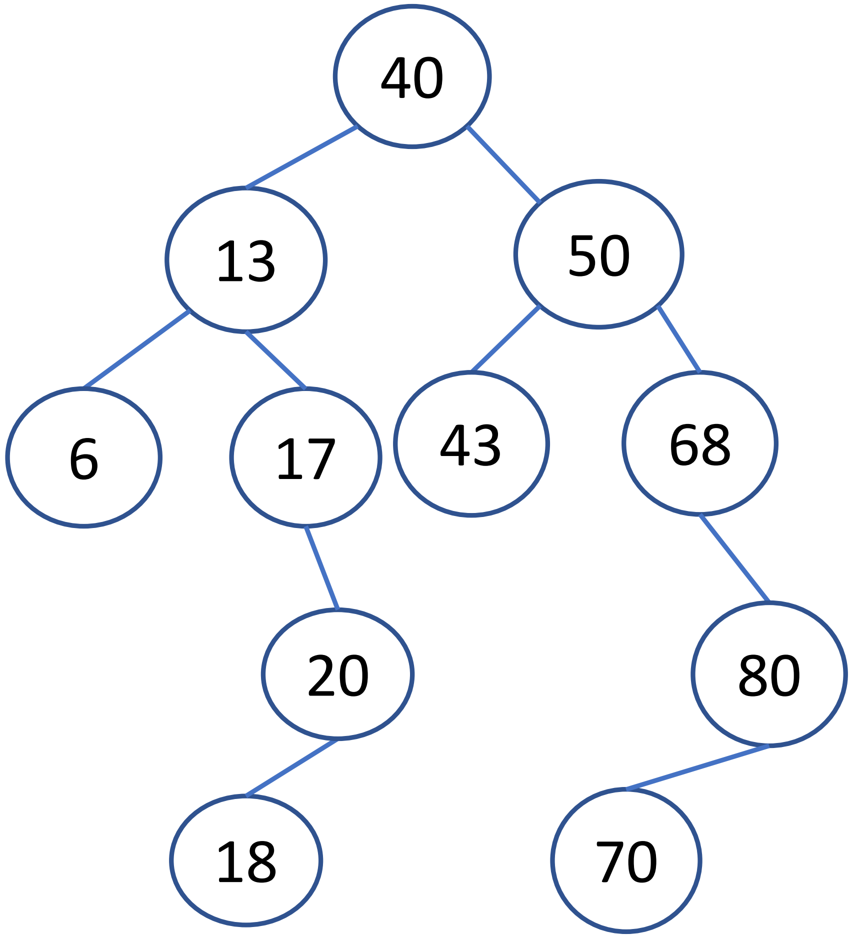
Exercise: Predecessor and Successor (1)



Consider the following binary search tree, what are the predecessor and successor of the corresponding nodes in the table?

Node	Predecessor	Successor
36		
13		
22		
55		
80		

Exercise: Predecessor and Successor (2)

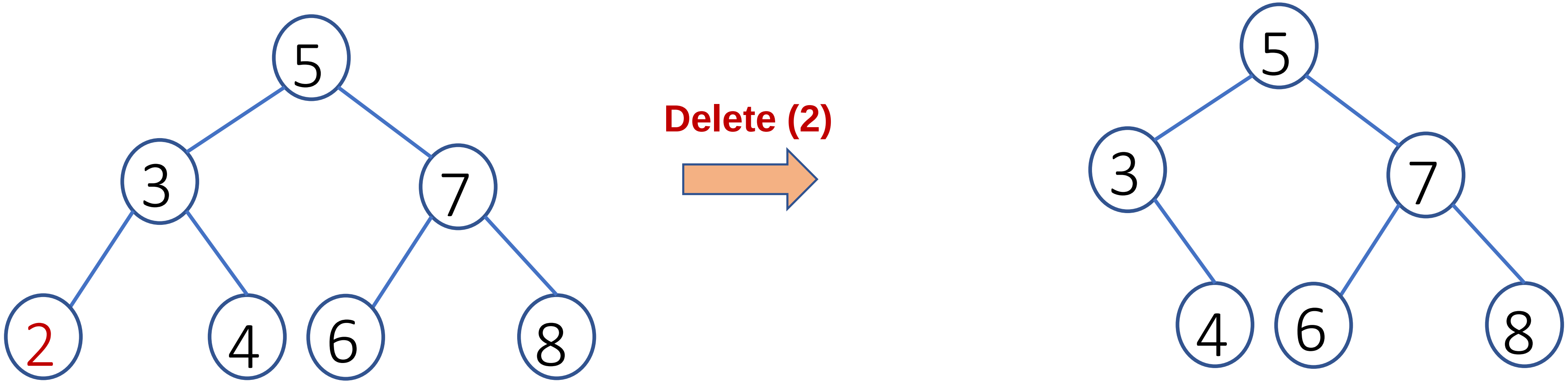


Consider the following binary search tree, what are the predecessor and successor of the corresponding nodes in the table?

Node	Predecessor	Successor
40		
13		
50		
68		
70		

Binary Search Tree: Delete

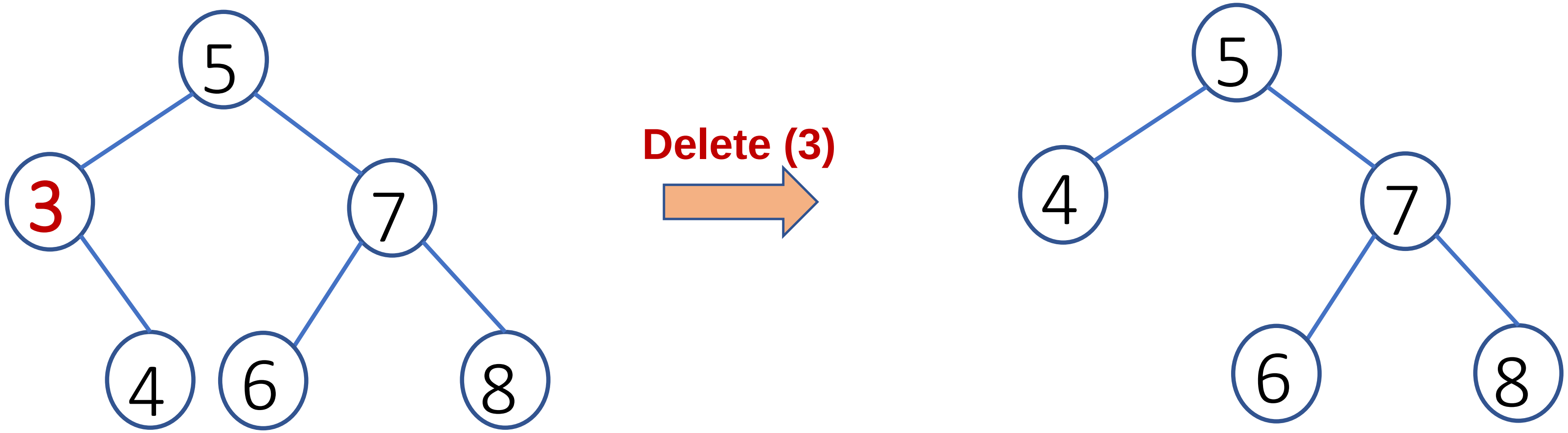
- **Deleting a leaf node**
- Deleting a node with one child
- Deleting a node with two children's



Delete (2)

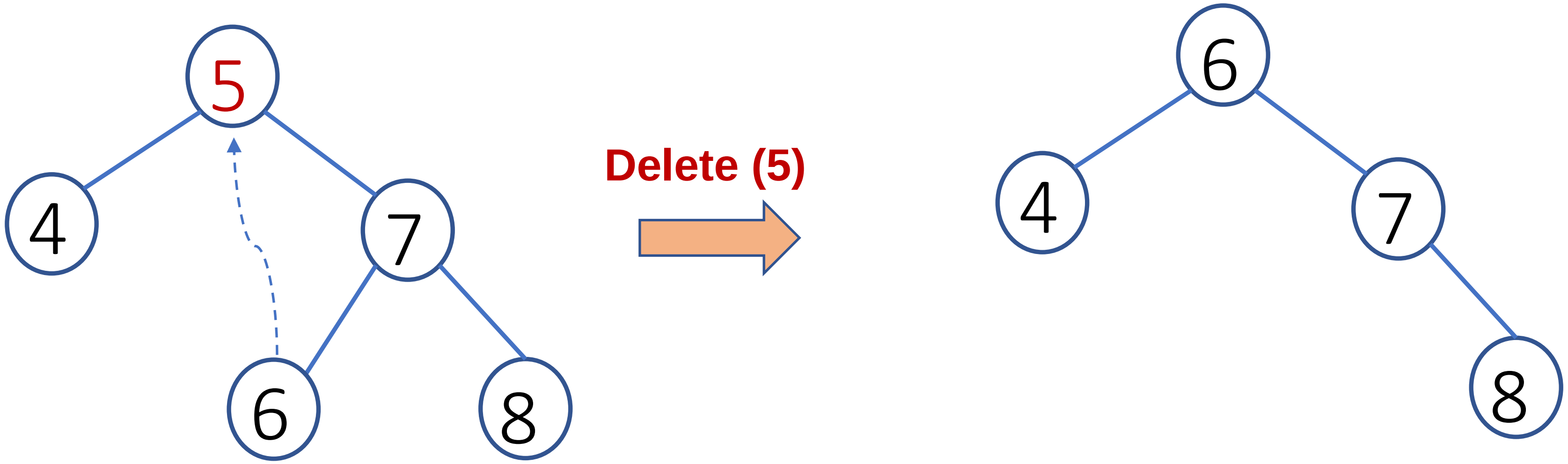
Binary Search Tree: Delete

- **Deleting a node with one child**



Binary Search Tree: Delete

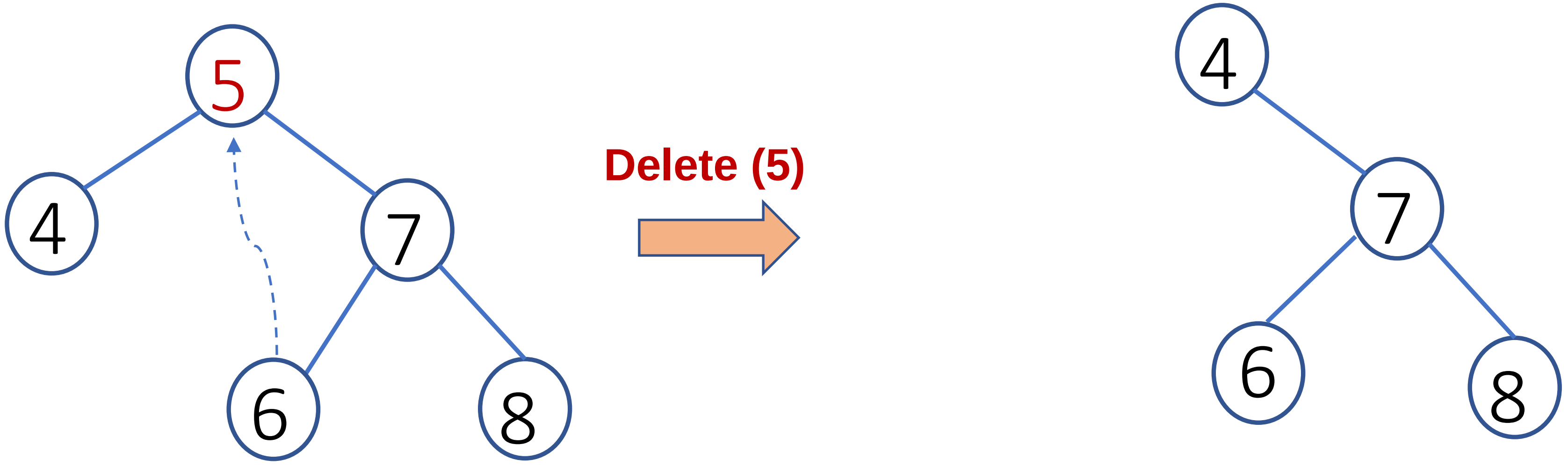
- **Deleting a node with two children's**



Find the inorder successor → Minimum values in the right subtree

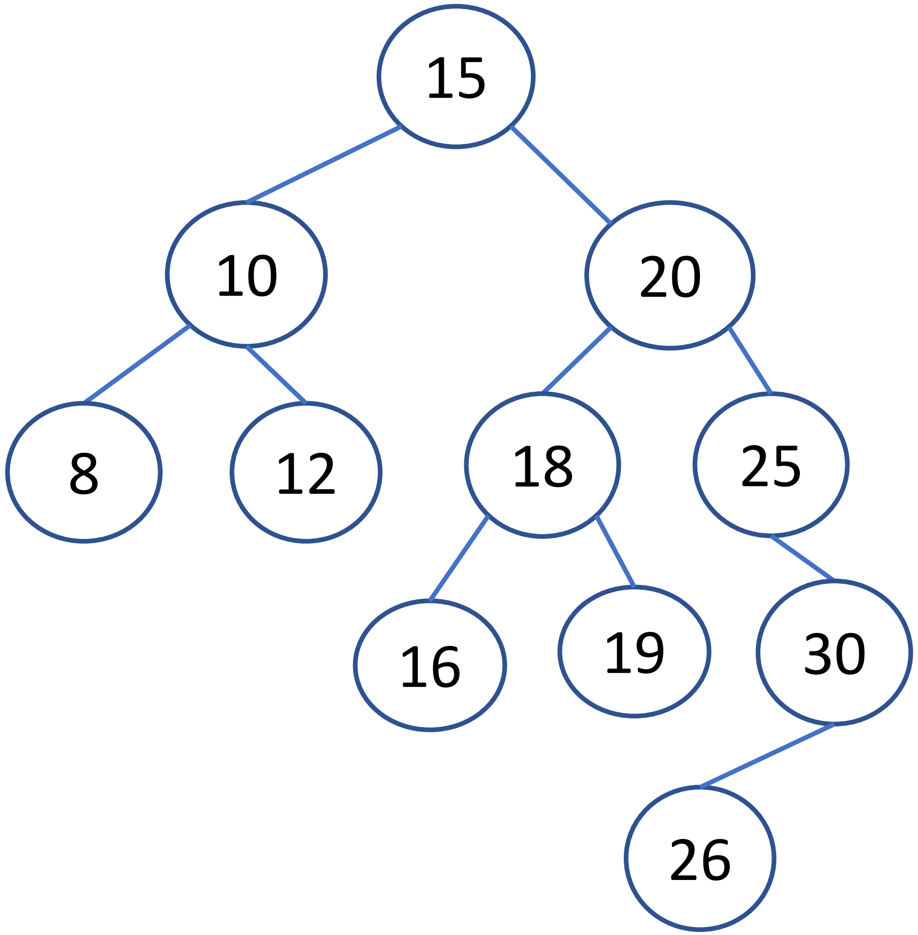
Binary Search Tree: Delete

- **Deleting a node with two children's**



Find the inorder successor → Maximum values in the left subtree

Exercise: Binary Search Tree Deletion (1)



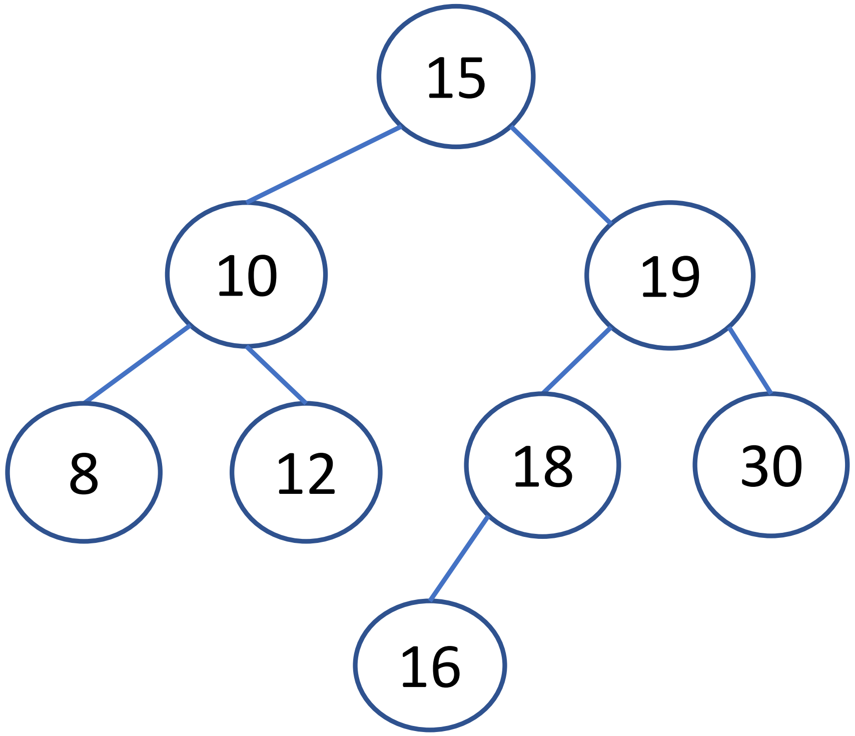
Consider the following binary search tree,

(i) What are the leaf level nodes after deleting 26?

(ii) What is the tree structure after deletion of 25?

(iii) What is the structure of the tree after deleting 20?

Exercise: Binary Search Tree Deletion (2)



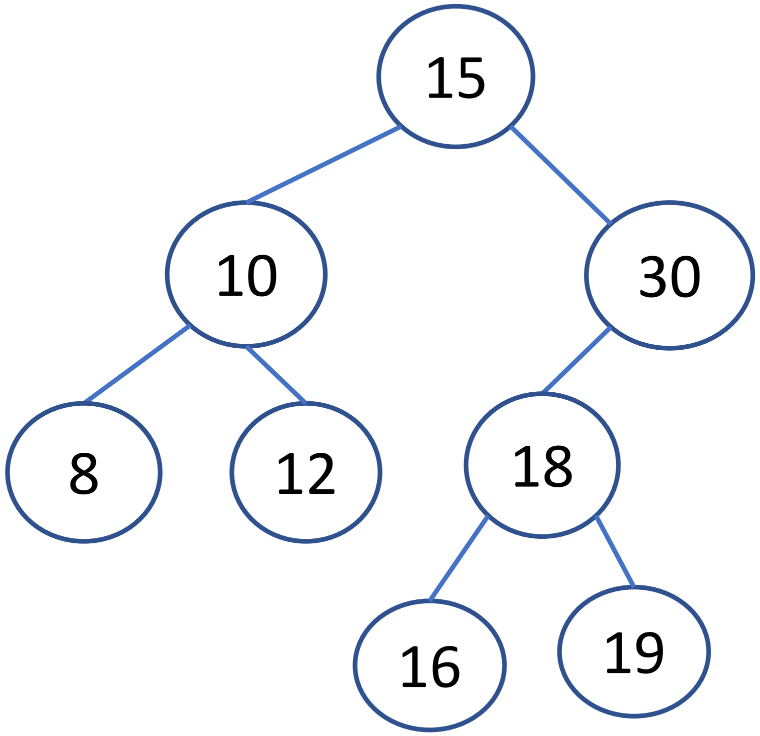
Consider the following binary search tree,

(i) What are the leaf level nodes after deleting 26?

(ii) What is the tree structure after deletion of 25?

(iii) What is the structure of the tree after deleting 20?

Exercise: Binary Search Tree Deletion (2)



Consider the following binary search tree,

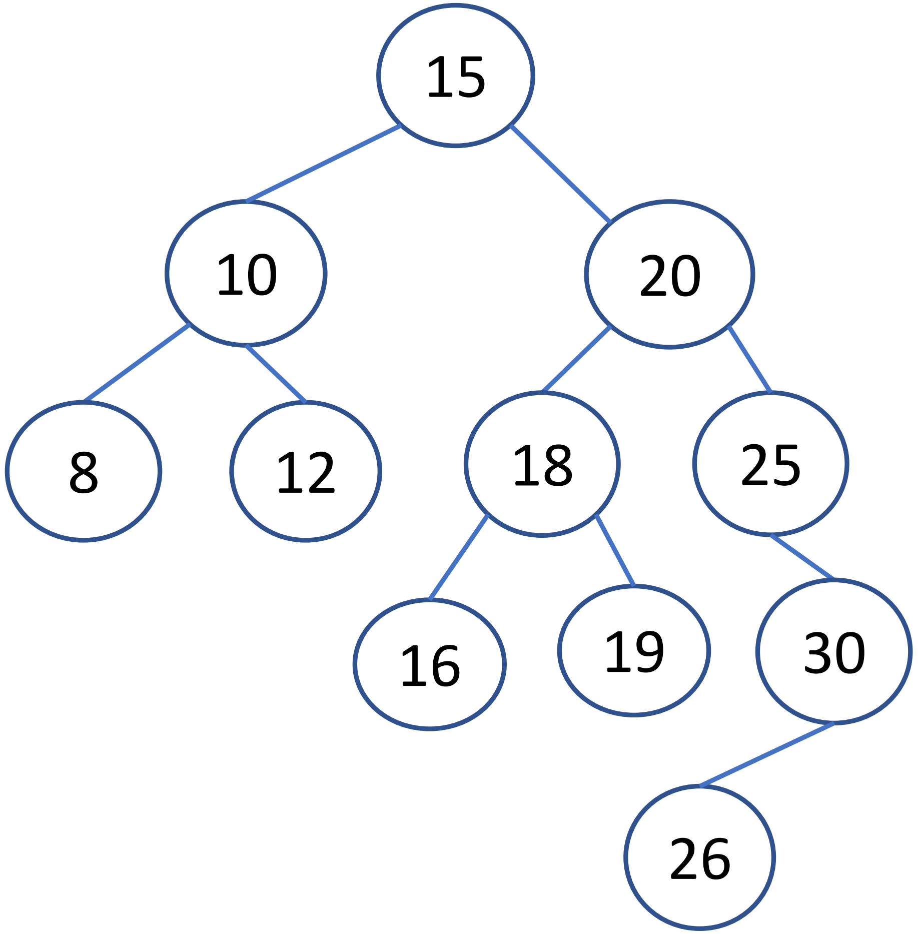
(i) What are the leaf level nodes after deleting 26?

(ii) What is the tree structure after deletion of 25?

(iii) What is the structure of the tree after deleting 20?

Level-Order Tree Traversal:

```
Level order traversal:

 (i)Visit the starting node (root)
 (ii) Enqueue () all the elements in that
    level l+1
 (iii)Upon next level, visit all the nodes
 (iv)Repeat until all the levels visited
```



Exercise: Binary Search Tree

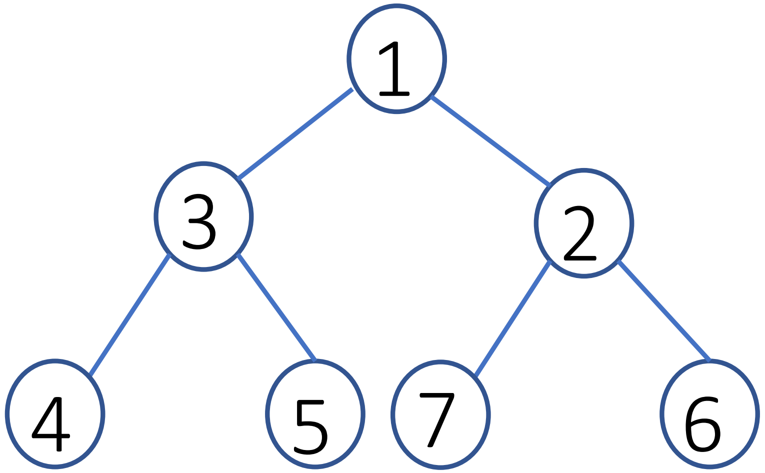
Suppose we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences <u>could not be</u> the sequence of nodes examined?

(i) 2, 252, 401, 398, 330, 344, 397, 363
(ii) 925, 202, 911, 240, 912, 245, 363
(iii) 924, 220, 911, 244, 898, 258, 362, 363
(iv) 935, 278, 347, 621, 299, 392, 358, 363

Exercise: Binary Search Tree

For the set of {1, 4, 5, 10, 16, 17, 21} of the keys, draw binary search tree of height 2, 3, 4, 5, and 6

Binary Tree: Travel with all possible scenarios!



Pre-order (RT, L, R) :

Reverse→ Pre-order (R, L, RT) :

In-order (L, RT, R) :

Reverse → In-order (R, RT, L) :

Post-order (L, R, RT) :

Reverse → Post-order (RT, R, L) :

thank you!

email:
k.kondepu@iitdh.ac.in