Day - 150

Trees - 2

⇒ In the previous day, we have created tree level wise.

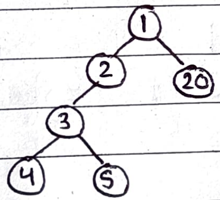
⇒ But today, we will see another method to create tree.

⇒ So, in the new method, we will create side by side that means first we will create or build left side then right side.

⇒ We will fill that side until we get -1.

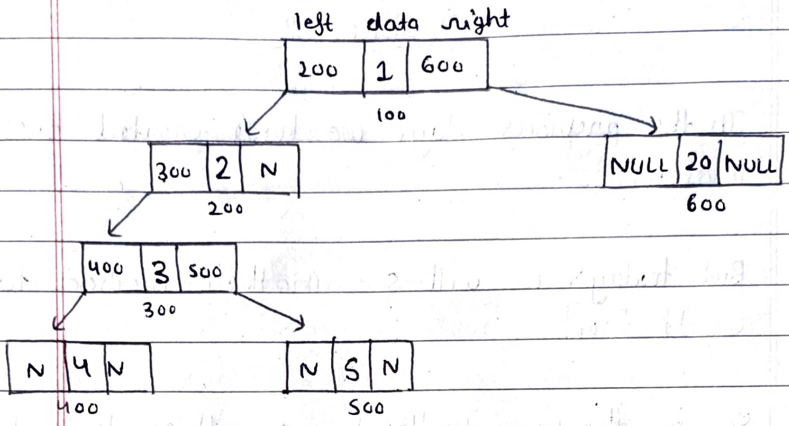
Ex: 1 2 3 4 -1 -1
5 -1 -1 -1 20 -1
-1

Tree



⇒ In this way, we will create a tree.

Internal Representation



Steps:

1. If x == -1, return NULL.
2. Create Node.
3. Left side jaao.
4. Right side jaao.
5. return Node address.

Code

~~Binary Tree()~~

```
class Node {
        public:
        int data;
        Node *left, *right;
        Node(int value){
                data = value;
                left = right = NULL;
        }
}
```

```
Node *   Binary Tree (){
      int x;
      cin >> x;
      if( x== -1)
            return NULL;
      Node * temp = new Node(x);
      temp → left = Binary Tree();
      temp → right = Binary Tree();
      return temp;
}
```

T.C. → O(n) , S.C → O(h) → for best & avg. case & O(n) → for worst case

* Traversal:

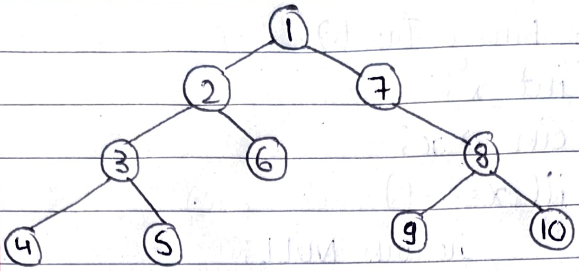
⇒ visiting nodes in a particular order.

* Types of Traversal:

1. Pre-order Traversal → N L R
2. In-order Traversal → L N R
3. Post-order Traversal → L R N

* Pre-order Traversal

N L R



⇒ <u>In Pre-order</u>

⇒ First, we traverse Node then left side and then right side.

<u>Ans →</u> 1 2 3 4 5 6 7 8 9 10

⇒ <u>Inorder Traversal</u>

⇒ L N R

<u>Ans →</u> 4 3 5 2 6 1 7 9 8 10

⇒ <u>Postorder Traversal</u>

L R N

<u>Ans →</u> 4 5 3 6 2 9 10 8 7 1

<u>Code</u>

<u>Preorder</u>

```
void preorder( Node *root){
    if( root == NULL)
        return;
        cout<< root →data;
        preorder( temp→left);
        preorder( temp → right);
}
```

T.C. → O(N) , S.C. → O(H) or O(N)

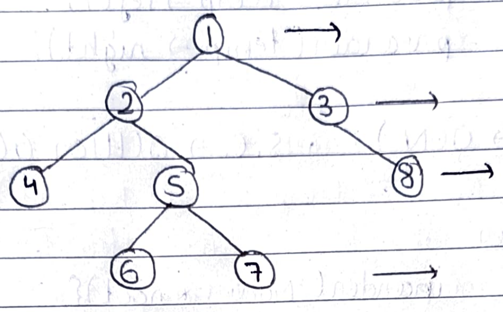
Inorder

```
    void    inorder( Node *root){
        if(root == NULL)
            return;
        inorder( root → left);
        cout<< root →data;
        inorder( root → right );
}
```

Postorder

```
   void  postorder( Node * root){
          if( root == NULL)
              return;
          postorder (root → left);
          postorder ( root → right);
          cout << root → data;
}
```

* Level order Traversal:



Ans →

1 2 3 4 5 8 6 7

⇉ In this traversal, we traverse the tree level wise.