Date Page

Day - 163

Binary Search Tree (BST)

* **Binary Search Tree:**

⇒ In normal tree, if we have to find any node then we have to traverse the whole tree.

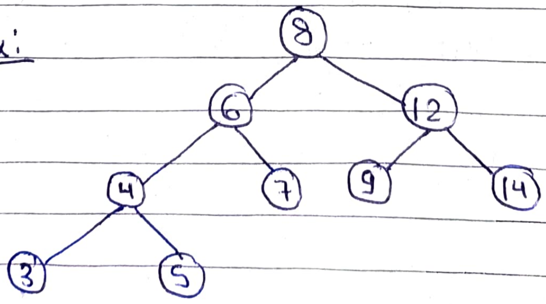
⇒ Because there is no order in the whole tree.

⇒ So, to solve this problem, we have use binary Search tree.

⇒ Here, we define relationship among ~~element~~ nodes.

⇒ Every left side nodes are smaller than the root node & every right side nodes are greater than the root node.

Ex:

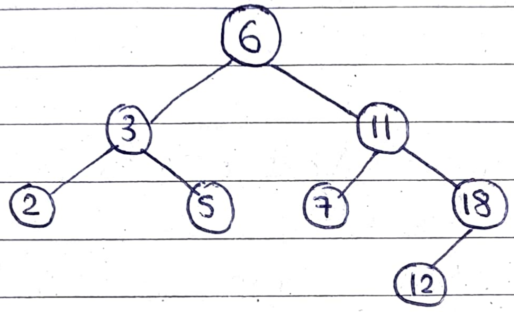


⇒ So, if we want to search 9 then we only have to go to right side.

Date ______ Page ______

⇒ If we want to create a BST from an array –

6	3	11	5	7	18	12	2



⇒ If the value is smaller, go to left side otherwise on the right side.

⇒ If the node data is same then we can pass it to any side.

⇒ When we are creating tree then first we will create a node then return the address.

Code

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

Date ________ Page ________

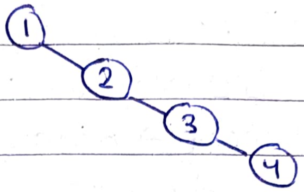
```
int main(){
    int arr[] = {3, 7, 4, 1, 6, 8};
    Node * root = NULL;
    for( i=0; i< 6; i++)
      root = insert(root, arr[i]);
}

Node * insert (Node * root, int target){
    if( !root){
        Node * temp = new Node(target);
        return temp
    }
    if(target < root →data)
    root→left=insert( root →left, target);
    else {
        root → right= insert( root → right,
                    target);
    }
    return root;
}
```

⇒ As we can see, when we are inserting any node, it is adding to the end of the tree as leaf node.

⇒ So, in the worst case, it can be height of the tree.

⇒ So, T.C. → O(h)

⇒ Also, if the tree is like –



⇒ Then if we are adding (5) then, it means we are traversing all the nodes.

⇒ So, in the worst case or edge case like this, T.C → $O(n)$

⇒ Creating a whole tree –
T.C. → $O(n^2)$

⇒ S.C. → $O(n)$

⇒ Inorder traversal of the BST ~~is~~ always give sorted result.

* **<u>Searching a node:</u>**

⇒ Searching is same like inserting any node.

⇒ If we get null that means node not found.

<u>Code</u>

Date Page

```
bool search( Node * root, int target){
        if(! root)
            return 0;
        if( root -> data == target)
            return 1;
        if( root -> data > target)
            return search (root -> left, target);
        else
            return search(root -> right, target);
}
```

T.C. → B.C. → O(h)
A.C. → O(h)
W.C. → O(n)

* **Delete Operation:**

=> If we want to delete a leaf node — then simply delete it & return null.

=> If there is a case that only left or right exist then delete that node & return their existing child.

=> If both the children exist then after deleting the node, we have to take a node from the subtree & replace it.

=> We can select the left subtree rightmost node or right subtree leftmost node.

Date ______ Page ______

⇒ First, we find the node then their rightmost node.

⇒ After that make that node parent to point to the left of the node.

⇒ After that node left will point to root left & node right to root right.

⇒ Then return the node.

⇒ If the parent ~~or~~ and root is same then right of child will point to ~~ro~~ right of root.

T.C. → O(n)