Kth largest/smallest element in Binary Search Tree

Problem statement: Given a binary search tree find the kth largest and smallest element in Binary Search Tree.

Examples:

```
Input: N=6

       Arr=[5,3,6,2,4,1]

       K=3



Output: Kth largest element is 4

        Kth smallest element is 3
```

```
Input: N=7

       Arr=[10,40,45,20,25,30,50]

       k=3



Output: Kth largest element is 4

        Kth smallest element is 3
```

Solution:

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Approach:

Here first, we take the input array and we insert all elements of the array into a BST.

And after that, we take a variable K.

Then I have to find the Kth largest and smallest element in BST.

So, I created two functions one is kthlargest and the other is kthsmallest.

The first function gives us the Kth largest element of that BST and the second function gives us the Kth smallest element of that BST.

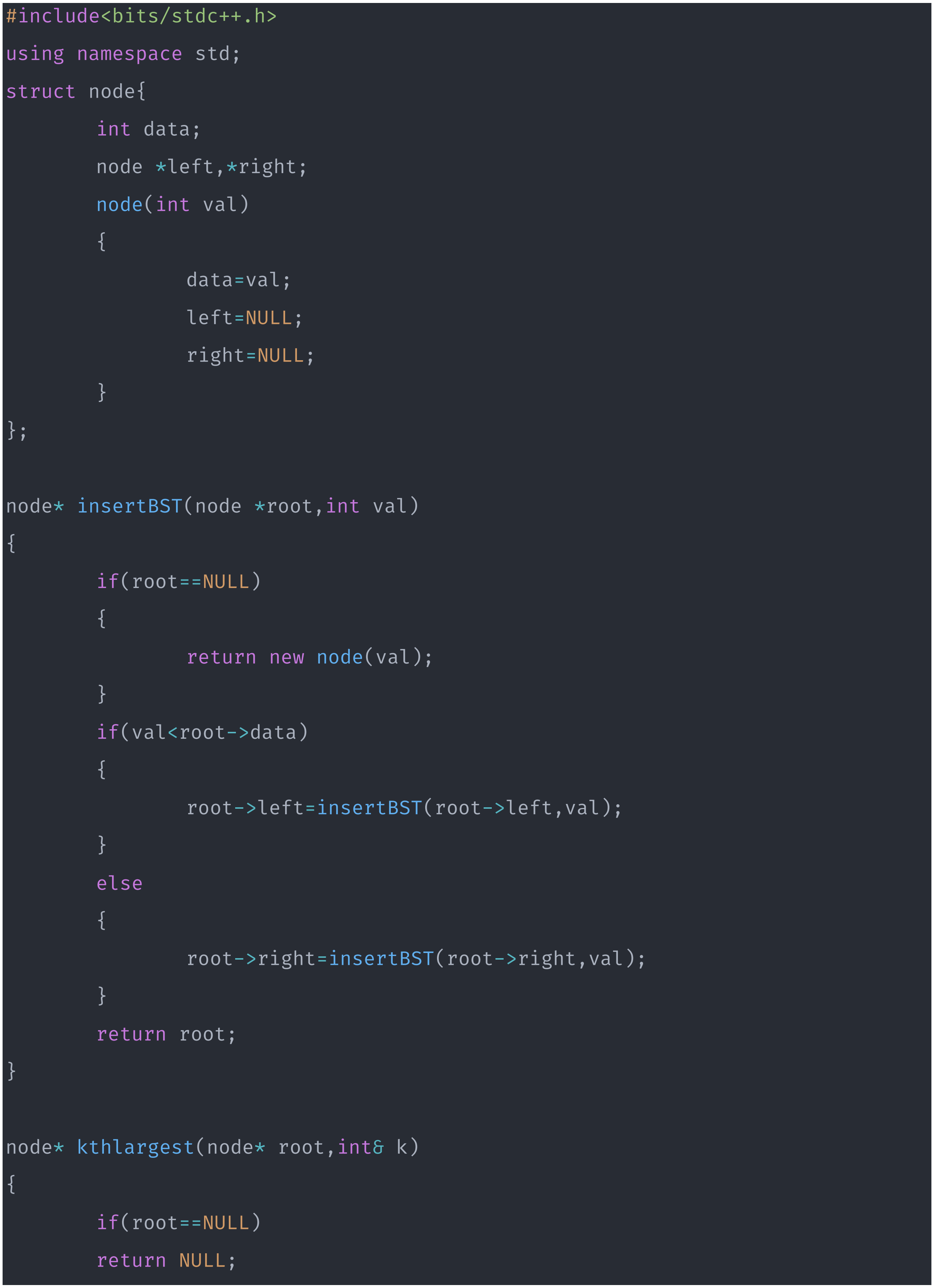
Both functions have two arguments one is root and another is K. In Kthlargest element function I call that function in reverse inorder traversal that means first right subtree after that root and last left subtree if I decrement k by 1 in each function because the maximum value of that BST is the right-most subtree value that's why we decrement K by 1 each function if any instances we found K is equal to 0 then simply return the root(which is our desired value). The question is why we reduce K by 1 because we traverse like reverse inorder traversal so if we construct an array by traversal the 0 indexed value will be the rightmost value and the and the next value will be the next function which terminated after that.

Now, comes to the kthsmallest element same as the kthlargest element just one change that is we follow inorder traversal here because we will get the element in a sorted order that's why we call the left subtree first and after that root and after that, we call right subtree and we decrement K by 1 in each function. If any instance we found k is equal to 0 then we return the root. If any function we found that root is equal to NULL then return NULL.

In this program we have one corner case if we found NULL after executing the function then the K is greater than N in that case we simply print Invalid input.

Code:

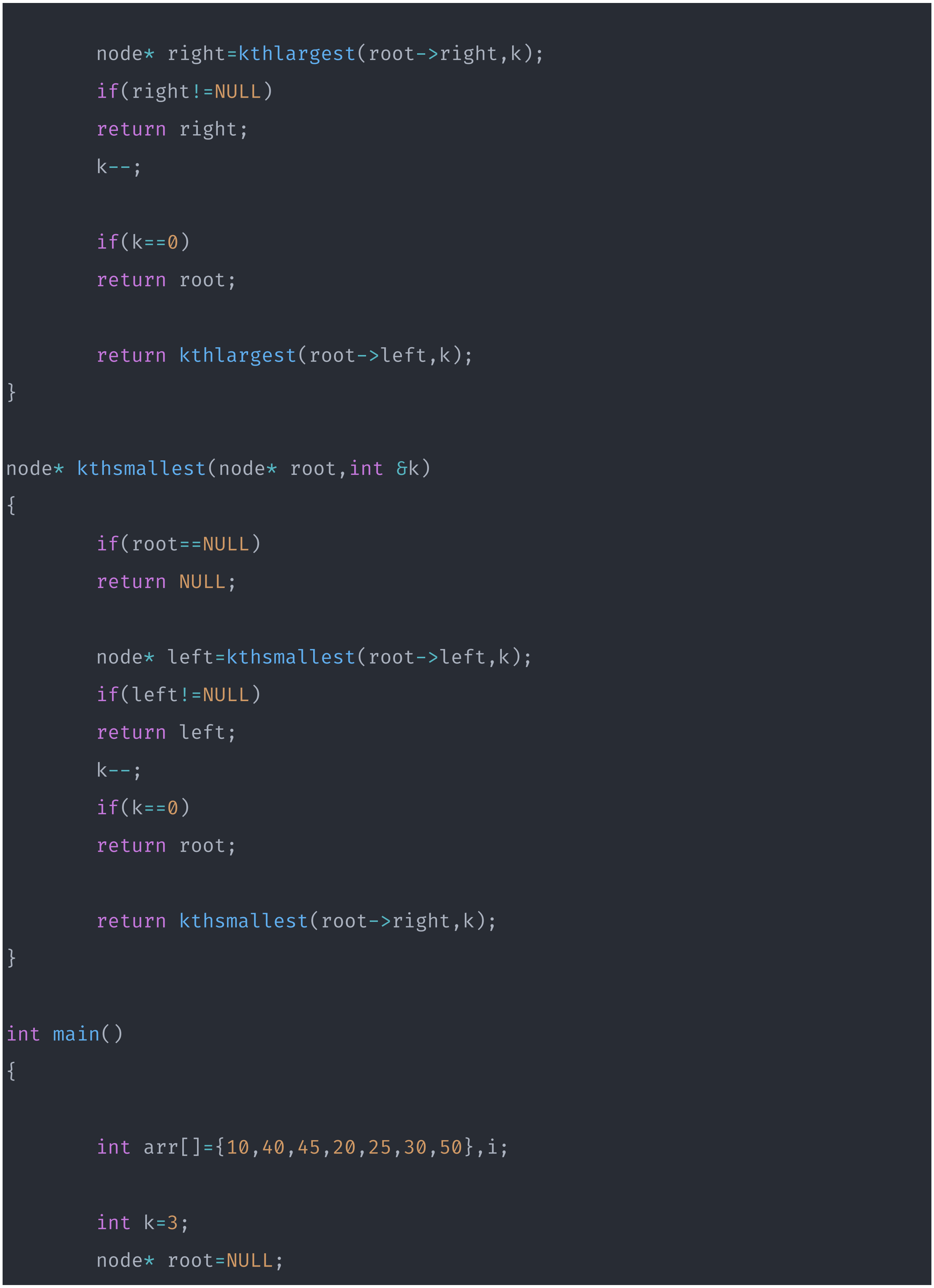
- C++ Code
- Java Code



```cpp
#include<bits/stdc++.h>
using namespace std;
struct node{
        int data;
        node *left,*right;
        node(int val)
        {
                data=val;
                left=NULL;
                right=NULL;
        }
};

node* insertBST(node *root,int val)
{
        if(root==NULL)
        {
                return new node(val);
        }
        if(val<root->data)
        {
                root->left=insertBST(root->left,val);
        }
        else
        {
                root->right=insertBST(root->right,val);
        }
        return root;
}

node* kthlargest(node* root,int& k)
{
        if(root==NULL)
        return NULL;
```



```
        node* right=kthlargest(root->right,k);
        if(right!=NULL)
        return right;
        k--;

        if(k==0)
        return root;

        return kthlargest(root->left,k);
}

node* kthsmallest(node* root,int &k)
{
        if(root==NULL)
        return NULL;

        node* left=kthsmallest(root->left,k);
        if(left!=NULL)
        return left;
        k--;
        if(k==0)
        return root;

        return kthsmallest(root->right,k);
}

int main()
{

        int arr[]={10,40,45,20,25,30,50},i;

        int k=3;
        node* root=NULL;
        for(i=0;i<7;i++)
        {
                root=insertBST(root,arr[i]);
        }
        cout<<"\n";

        int p=k;
        node* large=kthlargest(root,k);
        k=p;
        node* small=kthsmallest(root,k);
        if(large==NULL)
        {
                cout<<"Invalid input"<<"\n";
        }
        else
        cout<<"kth largest element is  "<<large->data<<"\n";

        if(small==NULL)
        {
                cout<<"Invalid input"<<"\n";
        }
        else
        {
                cout<<"kth smallest element is  "<<small->data<<"\n";
        }
}
```

Output:

```
kth largest element is 40
kth smallest element is 25
```

Time Complexity: O(min(K,N))

Space Complexity: O(min(K,N))