12)KTH LARGEST ELEMENT IN THE BST :

METHOD 1:
The first method is by using the global variable and the count variable and the reverse InOrder Traversal of the binary tree.We pass the count variable by address do the value of the count acts like global.
As soon as the count value becomes equal to k, we return that value.
Time Complexity : O(N)
Space Complexity : O(1)

METHOD 2:
This method is less efficient in terms of space.We traverse through the binary tree and store its inorder order traversal in the array.Then the Kth element from the behind is the answer.
Time Complexity : O(N)
Space Complexity : O(N)

METHOD 3: (Preferred)
This method is same as the first , but here we have not used the global variable.The main twist in this method is, we have to check if the count after traversing the right part is equal to k or not. If it is k then we return the element we got from the right part else we increase the count by 1 and then check if the count is equal or not. If count is still not equal then the answer liew in the left sub part so we return the answer returned from the left sub part.

CODE FOR METHOD 1:

```
class Solution
{
   public:
   int answer;
   void reverseInOrderTraversal(Node* root,int &count,int k){
      if(root!=NULL){
         reverseInOrderTraversal(root->right,count,k);
         count++;
         if(count==k){
            answer=root->data;
            return;
         }
         reverseInOrderTraversal(root->left,count,k);
      }
   }
   int kthLargest(Node *root, int k){
      int count=0;
      reverseInOrderTraversal(root,count,k);
      return answer;
   }
};
```

CODE OF METHOD 3:

```
class Solution
{
   public:
   int reverseInOrderTraversal(Node* root,int &count,int k){
      if(root!=NULL){
         int element=reverseInOrderTraversal(root->right,count,k);
         if(count==k){
            return element;
         }
         else{
            count++;
            if(count==k){
               return root->data;
            }
         }
         return reverseInOrderTraversal(root->left,count,k);
      }
   }
   int kthLargest(Node *root, int k){
      int count=0;
      int answer=reverseInOrderTraversal(root,count,k);
      return answer;
   }
};
```