Jake Tantorski
CMPT435
April 7th, 2020

The idea behind this is that the element at k-1 would be the element in the array that is the kth smallest due to the fact that the array is in order after traversing through it, therefore, at the element k we can see what the value would be.

```
kthSmallest(k,root,counter)
        newArray = inOrderTraversalK(root,counter)          //1
        Return newArray[k-1]                                //1

inOrderTraversalK(root,counter)
        newArray = new int[counter]                         //1
        inOrderHelperK(root,newArray)
        Return newArray                                     //1

inOrderHelperK(r,newArray)
        if( r != null)                                      //1
                inOrderHelperK(r.left,newArray)             //T(n/2)
                newArray[index]  = r.data                   //1
                Index++                                     //2
                inOrderHelperK(r.right,newArray)            //T(n/2)
        End if
```

$T(n) = T(n/2) + T(n/2) + 3$

$T(n) = 2 * T(n/2) + O(1)$

The running time of this function would be O(n) due to the Master Theorem
A = 2 B = 2 D = 0

$d < \log_b a$
$O(n^{\log BA})$
$O(n^1)$
$O(n)$

The idea behind this algorithm is that once you traverse the tree the new array is in order, therefore, you see what numbers are within the range of the two given numbers.

```
k1k2range(k1,k2,root,counter)
        newArray = inOrderTraversalRange(root,counter)    //1
        for( i = 0; i <newArray.length; i++)              //n
                if(newArray[i] >= k1 && newArray[i] <= k2)
                        System.out.print(newArray[i]+ " "); //1
                End if
        End for


inOrderTraversalRange(root,counter)
        newArray = new int[counter]                 //1
        inOrderHelperK(root,newArray)               //1
        Return newArray

inOrderHelperRange(r,newArray)
        if( r != null)
                inOrderHelperRange(r.left,newArray)     //T(n/2)
                newArray[index]  = r.data               //1
                Index++                                 //2
                inOrderHelperRange(r.right,newArray)    T(n/2)
        End if
```

$T(n) = T(n/2) + T(n/2) + 3$

$T(n) = 2 * T(n/2) + O(1)$

The running time of this function would be O(n) due to the Master Theorem
A = 2 B = 2 D = 0

$d < \log_b a$
$O(n^{\log BA})$
$O(n^1)$
$O(n)$