Time Complexity Analysis

Question #7

```
Public BinarySearchTree transformation(E[] arr, BinaryTree<E> bt){
  int index;                                                        Θ(1)
  E value;                                                          O(1)
  int i=0;                                                          Θ(1)
  Arrays.Sort(arr);                                                 O(n2),
  while(i<arr.length){                                              O(n)
     add(bt.root, arr[i]);                                          O(n.2^n)
  }
  BinarySearchTree bst = new BinarySearchTree();                    Θ(1)
  bst.root = bt.root;                                               Θ(1)
  return bst;
}                      Worst case          Best case                Θ(1)
                       O(n2^n)             Θ(1)

private boolean add(BinaryTree.Node<E> localRoot, E item){
   boolean flag = false;
   if(localRoot != null){                                           Θ(1)
      if(localRoot.left != null){                                   Θ(1)
           flag = add(localRoot.left, item);
      }                                                             T(n-1)
      if(flag == false && localRoot.status == false){               Θ(1)
         localRoot.data = item;
         localRoot.status = true;
         return true;
      }
      if(flag == flag && localRoot.right != null){                  Θ(1)
         return add(localRoot.right, item);
      }                                                             T(n-1)
      return flag;
   }
   return flag                                                      Θ(1)
}
                                                                    Θ(1)
```

$T(n) = 2T(n-1) + 1$ — ①

$T(n) = 2[2T(n-1) + 1] + 1$

$T(n) = 2^2 T(n-2) + 2 + 1$ — ②

$T(n) = 2^2 [2T(n-3)] + 2^2 + 2 + 1$ — ③

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2 + 1$ — ⓚ

$T(n) = 2^k T(n-k) + 2^{k-1} + 2^{k-2} \dots + 2^2 + 2 + 1$

Assume $n-k = 0$

$n = k$

$= 2^n T(0) + 1 + 2 + 2^2 + \dots + 2^{k-1}$

$= 2^n \times 1 + 2^k - 1$

$= 2^n + 2^n - 1$ $\quad O(2^n)$

$T(n) = 2^{n+1} - 1$ $\quad \theta(2^n)$

$O(2^n)$

Best case

$\theta(1)$

Worst case

$O(2^n)$

~~Quest~~

Question #2

```
Public BinarySearchTree transformation(BinarySearchTree bst){
  BinarySearchTree tmp =new BinarySearchTree();          Θ(1)
  if(bst.root !=null){                                    Θ(1)
     bst.root = inOrder(bst.root);                        Θ(n²)
  }
  return bst;           worst case        Best case       Θ(1)
 }                      Θ(n2)             Θ(1)
Private BinaryTree.Node<E> inOrder(BinaryTree.Node<E> root){
  if(root==null)
     return null;                                            Θ(1)
   root=rotation(root);                                      Θ(n2)
   root.left =inOrderder(root.left);                         T(n-1)
   root = rotation(root);                                    Θ(n2)
   root.right =inOrder(root.right);                          T(n-1)
   root =rotation(root);                                     Θ(n2)
   return root;          worst case                          Θ(1)
}                        Θ(n2)            Best case
                                          Θ(1)
Private BinaryTree.Node<E> rotation(BinaryTree.Node<E> root){
      if(root !=null)
          int balance =getBalance(root);
              while(balance>1 || balance<-1){
                  root =bstRotation(root);                  n
                  balance =getBalance(root);                Θ(n)
              }                                             Θ(n)
            return root;

      }
      else{
          return null;
      }                          worst case        Best case
                                 Θ(n2)             Θ(1)
}
Private BinaryTree.Node<E> bstRotation(BinaryTree.Node<E> root){
   Int balance =getBalance(root);
   If(root.left != null && root.left.left !=null){
      if(balance>1 && maxDepth(root.left)- maxDepth(root.left.left)>1{
          root =rightRotate(root);
      }
   }
```

```
if(root.right != null && root.right.right != null)
    if(balance < -1 && maxDepth(root.right) - maxDepth(root.right.right)) 0
       root = leftRotate(root).

}

if (root.left != null && root.left.right != null){
  if(balance > 1 && maxDepth(root.left.right) - maxDepth(root.left.left
     > 0) {
     root.left = leftRotate(root.left);
        root = rightRotate(root);
  }
 if (root.right != null && root.right.left != null){
   if(balance < -1 && maxDepth(root.right.left) - maxDepth(root.right.right
     > 0) {
     root.right = rightRotate(root.right);
     root = leftRotate(root);
     }
  }
  return root;
}
```

Best case	Worst case
$\Theta(1)$	$\Theta(n)$

Question #3

```
Public void insert(E item){
   if(head ==null){                                        } O(1)
      head =new SLNode(1,item);
      head.links[0] =null;
   }
   else{                                                   } O(n)
      add(item);
   }
   size ++;
   if(size > (maxLevel -1)*100){                           } O(n)
      maxLevel ++;
      increaseLevel();
      connect();
   }
}
```

Best case	worst case
O(1)	O(n)

```
private void increaseLevel(){
   SLNode<E> iter =head;                                   } O(1)
   SLNode<E> tmiter ;
   int tmlevel =0;
   while(iter !=null){                                     } O(n)
      if(iter.links.length > tmpLevel){
            tmpLevel =iter.links.length;
      }
      iter=iter.links[0];
   }
   iter =head;
   while(iter.links[0] !=null){
      if(iter.links[0].links.length ==tmpLevel){
         tmpIter = iter.links[0];
         Iter.links[0] =new SLNode(tmpLevel +1,iter.links[0].data);   } O(n)
            Iter.links[0].links[0] = tmpIter.links[0];
      }
      iter= iter.links[0];
   }
}
```

worst case	Best case
O(n)	O(1)

```
Private void add(E item) {
 int lenght;
 int level;
 SLNode<E> iter=head;
 SLNode<E> lastItem=null;                          θ(1)
 boolean flag = false;
 if(head.data.compareTo(item) >0) {
    level=getlevel(8);
    SLNode<E> tmp = new SLNode(level,item);           θ(1)
    tmp.links[0] =head;
    this.head=tmp;
    flag=true;
    connect();

 }
 while(iter != null) {                                O(n)
   lastItem =iter;
   if(iter.links[0].data.compareTo(item) <0 && iter.links[0] !=null)
   {
     if(iter.links[0].data.compareTo(item) >0){           θ(1)
       lenght=findLenght(iter);
       level=getLevel(lenght);
       SLNode<E> tmp= new SLNode(level,item);
       tmp.links[0]=iter.links[0];
       iter.links[0]=tmp;
       connect();
       flag=true;
       break;
     }
 }
 iter=iter.links[0];
 }
 if(lastItem !=null && flag==false){
   lenght =findLenght(lastItem);                      θ(1)
   level=getLevel(lenght);
   SLNode<E> tmp =new SLNode(level,item);
   tmp.links[0]=null;
   lastItem.links[0]=tmp;
}
```

Worst case	Best case
O(n)	θ(1)