Q1.

```
public class BSTSort {
   static class Node{
      int val;
      Node left;
      Node right;
      Node(int val){
        this.val = val;
      }
      Node(int val, Node left, Node right){
         this.val = val;
         this.left = left;
         this.right = right;
      }
   }

   Node root;


   BSTSort(List<Integer> list){
      root = new Node(list.get(0));
      for(int i=1;i<list.size();i++){
         insert(root,list.get(i));
      }
   }

   private void insert(Node node, int a){
      if( a > node.val){
         if(node.right != null){
            insert(node.right,a);
         }else{
            node.right = new Node(a);
         }
      }else{
         if(node.left!=null){
            insert(node.left,a);
         }else{
            node.left = new Node(a);
         }
      }
   }

   private List<Integer> inorderTraversal(Node node, List<Integer> list){
```

```
    if(node == null) return list;

    inorderTraversal(node.left,list);
    list.add(node.val);
    inorderTraversal(node.right,list);

    return list;
  }

  public List<Integer> sort(){
    return this.inorderTraversal(root,new ArrayList<>());
  }

  public static void main(String[] args){
    System.out.println(new BSTSort(List.of(4,3,78,55,99,0,12,3,5)).sort());
    System.out.println(new
BSTSort(List.of(77,78,99,1001,2003,45,67,89,0,1,45,33,56,78,99,10001,12345,67,32,22,
11,76,89,96,108,120,2000,2001,2020,0,45,66,78,56)).sort());
  }
}
```

**Worst Case:**

```
insertion O(n) => n items * O(n) => O(n*n) => O(n^2), traversal O(n)
Time complexity = O(n^2) + O(n)
```

**Best Case:**

```
insertion O(logn) => n items * O(logn) => O(n*logn), traversal O(n)
Time complexity = O(n*logn) + O(n)
```

BST sorting Time complexity for best case O(nLogn) and for worst case O(n^2).

I ran a test on 500 arrays of size 1000 and i got the following result:

```
INSERTION SORT VERSION
time taken to execute: 2412ms
MERGE SORT VERSION
time taken to execute: 116ms
MERGE SORT PLUS VERSION
time taken to execute: 29ms
BST SORT VERSION
```

```
time taken to execute: 159019ms
```

Q2.

```
class Solution {

private int isDepthBalanced(TreeNode node){
if(node==null) return 0;
int leftDepth = isDepthBalanced(node.left);
int rightDepth = isDepthBalanced(node.right);
if(Math.abs(leftDepth - rightDepth)> 1){
return -1;
}
if(leftDepth == -1 || rightDepth == -1) return -1;
return Math.max(leftDepth,rightDepth) + 1;
}

public boolean isBalanced(TreeNode root) {
if(root==null) return true;
return isDepthBalanced(root) != -1;
}

}
```

Time Complexity: O(n)

Space Complexity: O(1)

Q3.

```
class Solution {
private int goodNodes(TreeNode root, int max){
if(root==null) return 0;
if(root.val >= max){
return 1 + goodNodes(root.left, root.val) + goodNodes(root.right,root.val);
}
return goodNodes(root.left, max) + goodNodes(root.right,max);
}
public int goodNodes(TreeNode root) {
return goodNodes(root, Integer.MIN_VALUE);
}
```

```
}
```

Time Complexity: O(n)

Space Complexity: O(1)

Q4.

```
class Solution {

public TreeNode trimBST(TreeNode root, int low, int high) {
if(root == null) return root;
if(root.left != null){
root.left = trimBST(root.left,low,high);
}
if(root.right != null){
root.right = trimBST(root.right,low,high);
}
if(root.val < low){
return root.right;
}
if(root.val > high){
return root.left;
}
return root;
}
}
```

Time  Complexity : O(n)

Space Complexity: O(1)