Question 2:

The algorithm of balanceTreeTwo ( ) balances the tree by taking every other node and rotating it. Since transformToList ( ) brings all the nodes on the right side in the correct order: left node then right and so on. Since they are already situated in the previous form of the tree balanceTreeTwo ( ) just rotates the odd nodes with the loop running acquired from the formula of M and N which is based on the length of the list. So we run the function until M for the odd nodes and then we compute K from the formula given to continue rotating the nodes to the left K number of times.

The total time complexity of balanceTreeOne ( ) is the sum of the time complexity of the sortedTree ( ) function and then the time complexity of implementing the array in the binary search tree. The time complexity, on average, of the sortedTree ( ) is O(1) because it is constant. Regardless of the size of the array it will always take the same amount of time to return the array.

```
public  int[] sortedTree() {
		return arr;
	}
```

The time complexity, on average, of the arrayToBst ( ) is O(n log n) where is the number of nodes being sorted. This function takes the sorted array and puts it in the format of the binary search tree.

```
Node arrayToBst (int [] a, int start, int end) {
		if(start > end) {
			return null;
		}
		int mid = a.length / 2;
		Node node = new Node (a[mid]);

		node.left = arrayToBst(a, start, mid - 1);
		node.right = arrayToBst(a, mid + 1, end);

		return node;
	}
```

So on an average case the time complexity of the balanceTreeOne ( ) is O(N log N) + O(1).

```
public BST.Node balanceTreeOne() {
		sortedTree();
		root = arrayToBst(arr, 0, arr.length);

		return root;
	}
```

The space complexity of balanceTreeOne ( ) is O(N) where N is the number of nodes being sorted from the array.

The time complexity of the balanceTreeTwo ( ) function is the sum of the cost of all the internal operations. The transformToList ( ) function has a time complexity of O (N) where N is the number of times that the loop runs which rotates all the left nodes on the right side of the root.

```
public void transformToList() {
            Node current = root;

            while(current != null) {
                  if(current.left != null) {
                        rotateRight(child);
                        current = current.next;
                  }
                  else {
                        current = current.next;
                  }
            }
      }
```

The time complexity of the variable x is 2*O(log N) where N is the value of which the log is being taken of. Time complexity of the variable M is 3 * O(N) because there are if- else statements which has a linear relationship with the number of inputs.

```
public int floor(double key, Node node) {
            int value = node.key;
            if(key == node.key) {
                  value = node.key;
            }
            else if (key > node.key) {
                  if(node.right != null) {
                        value = floor(key, node.right);
                  }
            }
            else if(key < node.key) {
                  if(node.left != null) {
                        value = floor(key, node.left);
                  }
            }

            return value;
      }
```

The time complexity of the while loop and the for loop is 2 * O(N) because both of the loop are being incremented by a constant amount making it a linear relationship. So the time complexity of the balanceTreeTwo ( ) is O(N) + 2*O(log N) + 3*O(N) + 2*O(N).

```
public void balanceTreeTwo() {
            transformToList();
            Node node = root;
            int N = size;
            double x = Math.log(N)/Math.log(2);
            double M = (N+1) - Math.pow(2,floor(x,root));

            for (int i = 0; i < M; i++) {
                  node = rotateLeft(node);
                  node = node.right;
            }
```

```
        node = root;
        double K = x - 1;
        while(K > 1) {
            node = node.right;
            node = rotateLeft(node);
            if(K == 1) {
                node = rotateLeft(root);
            }
            K--;
        }
    }
```

The space complexity of the balanceTreeTwo ( ) is O(*N*) because it balances the binary search tree.