i. For the balanceTreeTwo() function, it invoke the buildBST function, it's a recursion function, in each recursion , it add the middle element to the tree, and then recursion again but the parameter is the two side of the array. It's two parameter begin and end, represent the range that is going to build. for the first step , it put the middle value in the top of the tree, with the recursion, it will be a balance tree.

```
public void buildBST(BST bst, int[] v, int begin, int end) {
	if(begin > end)
		return;
	if(begin == end)
		bst.put(v[begin]);
	else if(begin + 1 == end)
	{
		bst.put(v[begin]);
		bst.put(v[end]);
	}else
	{
		int middle = (begin+end)/2;
		bst.put(v[middle]);
		buildBST(bst, v, begin, middle - 1);
		buildBST(bst, v, middle+1, end);
	}
}
```

ii. the total time complexity of the balanceTreeOne() is O(n)

$O(BalanceTreeOne)=O(sortedTree)+O(put(int[a]))$

for sortedTree, it does a traverse in order, it's time complexity is O(n), for put(int[a]), which is called in buildBST, because each index is viewed once, it's time complexity is also O(n). So the total time complexity of the balanceTreeOne() is O(n)

iii. the space complexity of the balanceTreeOne is O(2n). because it needs an extra array to store the data which is sorted.

the total time complexity of the balanceTreeTwo() is O(logn)

iv.

For the first left rotations loop, consider the worst situation, it's time complexity is O(n-logn), for the second left rotations loop, it's time complexity is O(logn), so the finally time complexity is O(logn)

v.

the space complexity of balanceTreeTwo() is O(n), because it need't extra space to complete the operation, so it's depend on the quantity of number.