Some usefuel code snippets

Check whether a given tree is a binary search tree

```
class Solution
{
    // Validating the given tree is BST or not by recursively checking the nodes and its subtrees are within valid range or not
    public boolean isValidBST(TreeNode root) {
            return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    boolean isValidBST(TreeNode root, long min, long max) {
        if(root == null) return true;

        if(root.val <= min || root.val >= max) return false;

        // when we got left, the min stays the same, only the max changes
        // when we go right, the max stays the same, only the min changes
        return isValidBST(root.left, min, root.val) && isValidBST(root.right, root.val, max);
    }
}
```

Convert a sorted array into a binary search tree

```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return builder(0, nums.length-1, nums);
    }
    public TreeNode builder(int min, int max, int arr[]) {
        if (min > max) return null;
        int mid = min + (max-min)/2;

        TreeNode root = new TreeNode(arr[mid]);
        root.left = builder(min, mid-1, arr);
        root.right = builder(mid+1, max, arr);
        return root;
    }
}
```

Graphs

Find the diameter of an unweighted acyclic graph (adjacency matrix, BFS)

Clarification: We run BFS $|V|$ times, s.t. we consider all possible longest paths from every node.

Thus, the overall time complexity is: $|V| \cdot \mathcal{O}(|V| + |E|) \leq \mathcal{O}(|V| \cdot |V| + |V| \cdot |E|) \overset{(1)}{\leq} \mathcal{O}(|V|^3)$ where we use the fact that in an undirected graph, there are at most

$\frac{|V|^2 - |V|}{2} \quad (1)$ edges and since the given graph is acyclic, we can surely say that this is an upper bound.

```
int getGraphDiameter() {
        boolean g[][] = graph;
        int max = -1;
        for (int i = 0; i < g.length; i++) {
            int s = bfs(i, g.length, g);
            if (s == -1) return -1;
            max = Math.max(max, s);
        }
        return max;
    }

public int bfs(int startingVertex, int numVertices, boolean arr[][]) {
        Queue<Pair> qu = new LinkedList<Pair>();
        boolean visited[] = new boolean[numVertices];
        qu.add(new Pair(startingVertex, 0));
        int dist = 0;

        while (!qu.isEmpty()) {
                Pair cur = qu.poll();
                if (!visited[cur.node]) {
                        dist = cur.distance;
                        visited[cur.node] = true;
                        for (int i = 0; i < arr[cur.node].length; i++) {
                                if (arr[cur.node][i] && !visited[i])
                                qu.add(new Pair(i, dist + 1));
                        }
                }
        }

    // if no every vertex got visited, the graph consists of more than one strongly
connected component --> no generic
diameter
        for (boolean bo : visited) {
                if (!bo) return -1;
        }
        return dist;
}
```