1. Given the root of a binary tree, return all duplicate subtrees.

For each kind of duplicate subtrees, you only need to return the root node of any one of them.

Two trees are duplicate if they have the same structure with the same node values.

Ans: /**

```
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    private Map<String, Integer> counter;
    private List<TreeNode> ans;

    public List<TreeNode> findDuplicateSubtrees(TreeNode root) {
        counter = new HashMap<>();
        ans = new ArrayList<>();
        dfs(root);
```

```
        return ans;
    }

    private String dfs(TreeNode root) {
        if (root == null) {
            return "#";
        }
        String v = root.val + "," + dfs(root.left) + "," + dfs(root.right);
        counter.put(v, counter.getOrDefault(v, 0) + 1);
        if (counter.get(v) == 2) {
            ans.add(root);
        }
        return v;
    }
}
```

2. You are given the root node of a binary search tree (BST) and a value to insert into the tree. Return *the root node of the BST after the insertion*. It is guaranteed that the new value does not exist in the original BST.

Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return any of them.

Ans:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
```

```
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode insertIntoBST(TreeNode root, int val) {
        if (root == null) {
            return new TreeNode(val);
        }
        if (root.val > val) {
            root.left = insertIntoBST(root.left, val);
        } else {
            root.right = insertIntoBST(root.right, val);
        }
        return root;
    }
}
```

3. Given an array of strings words representing an English Dictionary, return *the longest word in* words *that can be built one character at a time by other words in* words.

If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

Note that the word should be built from left to right with each additional character being added to the end of a previous word.

Ans:
```
class Solution {
    private Set<String> s;

    public String longestWord(String[] words) {
        s = new HashSet<>(Arrays.asList(words));
        int cnt = 0;
        String ans = "";
        for (String w : s) {
            int n = w.length();
            if (check(w)) {
                if (cnt < n) {
                    cnt = n;
                    ans = w;
                } else if (cnt == n && w.compareTo(ans) < 0) {
                    ans = w;
                }
            }
        }
        return ans;
    }
```

```
    private boolean check(String word) {
        for (int i = 1, n = word.length(); i < n; ++i) {
            if (!s.contains(word.substring(0, i))) {
                return false;
            }
        }
        return true;
    }
}
```

4. Given the root of a binary search tree, rearrange the tree in in-order so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child.

Ans:
```
public TreeNode increasingBST(TreeNode root) {
    List<Integer> list = new ArrayList<>();
    dfs(root, list);
    Integer[] arr = new Integer[list.size()];
    Arrays.sort(list.toArray(arr));
    TreeNode res = new TreeNode(arr[0]);
    TreeNode temp = res;
    int index = 1;
    while(index < list.size()){
        TreeNode node = new TreeNode(arr[index]);
        temp.right = node;
        temp = node;
```

```
        index++;
    }
    return res;
}
public void dfs(TreeNode node, List<Integer> list){
    if(node == null){
        return;
    }
    list.add(node.val);
    dfs(node.left,list);
    dfs(node.right,list);
}
```

5. A binary tree is uni-valued if every node in the tree has the same value.

Given the root of a binary tree, return true *if the given tree is uni-valued, or* false *otherwise.*

Ans:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
```

```
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isUnivalTree(TreeNode root) {
        return dfs(root, root.val);
    }

    private boolean dfs(TreeNode root, int val) {
        if (root == null) {
            return true;
        }
        return root.val == val && dfs(root.left, val) && dfs(root.right, val);
    }
}
```

6. Given a string date representing a Gregorian calendar date formatted as YYYY-MM-DD, return *the day number of the year*.

Ans:

```
class Solution {
    public int dayOfYear(String date) {
        int y = Integer.parseInt(date.substring(0, 4));
        int m = Integer.parseInt(date.substring(5, 7));
        int d = Integer.parseInt(date.substring(8));
        int v = y % 400 == 0 || (y % 4 == 0 && y % 100 != 0) ? 29 : 28;
```

```
        int[] days = {31, v, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        int ans = d;
        for (int i = 0; i < m - 1; ++i) {
            ans += days[i];
        }
        return ans;
    }
}
```

7. Given a date, return the corresponding day of the week for that date.

The input is given as three integers representing the day, month and year respectively.

Return the answer as one of the following values {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}.

Ans:

```
import java.util.Calendar;


class Solution {
    private static final String[] WEEK
        = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};


    public static String dayOfTheWeek(int day, int month, int year) {
        Calendar calendar = Calendar.getInstance();
        calendar.set(year, month - 1, day);
        return WEEK[calendar.get(Calendar.DAY_OF_WEEK) - 1];
    }
}
```

8. Given a sentence that consists of some words separated by a single space, and a searchWord, check if searchWord is a prefix of any word in sentence.

Return *the index of the word in* sentence *(1-indexed) where* searchWord *is a prefix of this word*. If searchWord is a prefix of more than one word, return the index of the first word (minimum index). If there is no such word return -1.

A prefix of a string s is any leading contiguous substring of s.

Ans: //O(N)time O(1)space

```
public int isPrefixOfWord(String sentence, String searchWord) {
    int res = -1;
    String[] arr = sentence.split(" ");
    for(int i = 0; i < arr.length; i++){
        if(arr[i].startsWith(searchWord)){
            return i + 1;
        }
    }
    return res;
}
```

9. Given a parentheses string s containing only the characters '(' and ')'. A parentheses string is balanced if:

- **Any left parenthesis '(' must have a corresponding two consecutive right parenthesis '))'.**
- **Left parenthesis '(' must go before the corresponding two consecutive right parenthesis '))'.**

In other words, we treat '(' as an opening parenthesis and '))' as a closing parenthesis.

- **For example, "())", "())(())))" and "(()()))))" are balanced, ")()", "()))" and "(()))" are not balanced.**

You can insert the characters '(' and ')' at any position of the string to balance it if needed.

Return *the minimum number of insertions* needed to make s balanced.

Ans:
```
class Solution {
    public boolean checkValidString(String s) {
        int n = s.length();
        boolean[][] dp = new boolean[n][n];
        for (int i = 0; i < n; ++i) {
            dp[i][i] = s.charAt(i) == '*';
        }
        for (int i = n - 2; i >= 0; --i) {
            for (int j = i + 1; j < n; ++j) {
                char a = s.charAt(i), b = s.charAt(j);
                dp[i][j] = (a == '(' || a == '*') && (b == '*' || b == ')')
                    && (i + 1 == j || dp[i + 1][j - 1]);
                for (int k = i; k < j && !dp[i][j]; ++k) {
                    dp[i][j] = dp[i][k] && dp[k + 1][j];
                }
            }
        }
        return dp[0][n - 1];
    }
}
```

10. You are given a 0-indexed 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using all the elements from original.

The elements from indices 0 to n - 1 (inclusive) of original should form the first row of the constructed 2D array, the elements from indices n to 2 * n - 1 (inclusive) should form the second row of the constructed 2D array, and so on.

Return *an* m x n *2D array constructed according to the above procedure, or an empty 2D array if it is impossible*.

Ans:
```
class Solution {
    public int[][] construct2DArray(int[] original, int m, int n) {
        if (m * n != original.length) {
            return new int[0][0];
        }
        int[][] ans = new int[m][n];
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                ans[i][j] = original[i * n + j];
            }
        }
        return ans;
    }
}
```

11. Given two positive integers a and b, return *the number of common factors of* a *and* b.

An integer x is a common factor of a and b if x divides both a and b.

Ans: class Solution {

```
    public int commonFactors(int a, int b) {
        int g = gcd(a, b);
        int ans = 0;
        for (int x = 1; x <= g; ++x) {
            if (g % x == 0) {
                ++ans;
            }
        }
        return ans;
    }

    private int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }
}
```

12. Given an integer array nums of size n, return *the number with the value closest to* 0 *in* nums. If there are multiple answers, return *the number with the largest value*.

Ans: class Solution {

```
    public int findClosestNumber(int[] nums) {
        int ans = 0, d = 1 << 30;
        for (int x : nums) {
            int y = Math.abs(x);
            if (y < d || (y == d && x > ans)) {
                ans = x;
```

```
                d = y;
            }
        }
        return ans;
    }
}
```