MISCELLANEOUS SET 2

Subsets O(2^n)

Given an array of integers that might contain **duplicates**, return all possible subsets.

Eg. 1 2 3 3
()(1)(1 2)(1 2 3)(1 2 3 3)(1 3)(1 3 3)(2)(2 3)(2 3 3)(3)(3 3)

```
class solve{
	static void find(int arr[],int pos, int n,HashSet<String> set,ArrayList<Integer>
	list, String str, ArrayList<ArrayList<Integer>> ans){
		for(int i=pos;i<n;i++){
			list.add(arr[i]);
			str += (char)arr[i];
			if(!set.contains(str)){
				set.add(new String(str));
				ans.add(new ArrayList<>(list));
			}
			if(pos!=n-1)
				find(arr, i+1, n, set, list, str, ans);
			list.remove(list.size()-1);			//Backtracking
			str = str.substring(0, str.length()-1);		//Backtracking
		}
	}
	public static ArrayList <ArrayList <Integer>> AllSubsets(int arr[], int n){
		ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
		ArrayList<Integer> list = new ArrayList<>();
		HashSet<String> set = new HashSet<>();
		find(arr,0, n, set, list, "", ans);
		return ans;
	}
}
```

Partition array to K subsets

Given an integer array A[] of N elements, the task is to complete the function which returns true if the array A[] could be divided into K non-empty subsets such that the sum of elements in every subset is same.

Note: All elements of this array should be part of exactly one partition.

```
class GfG{
	boolean rec(int n,int k,int sum,int subset[],boolean vis[],int a[],int index,int pos){
		if(subset[index]==sum){
			if(k-2==index)
				return true;
```

```
                return rec(n,k,sum,subset,vis,a,index+1,n-1);
            }
            for(int i=pos;i>=0;i--){
                if(!vis[i]){
                    int temp = subset[index] + a[i];
                    if(temp<=sum){
                        subset[index] += a[i];
                        vis[i] = true;
                        if(rec(n,k,sum,subset,vis,a,index,i-1))
                            return true;
                        subset[index] -= a[i];
                        vis[i] = false;
                    }
                }
            }
            return false;
        }
        boolean isKPartitionPossible(int a[], int n, int k){
            if(k==1)
                return true;
            if(n<k)
                return false;
            int sum = 0;
            for(int i=0;i<n;i++){
                sum += a[i];
            }

            int subsetSum = sum/k;
            if(k*subsetSum==sum){
                boolean vis[] = new boolean[n];
                int subset[] = new int[k];
                subset[0] = a[n-1];
                vis[n-1] = true;
                return rec(n,k,subsetSum,subset,vis,a,0,n-1);
            }
            return false;
        }
}
```

Infix to Postfix Conversion

Eg. (A+B)*C → AB+C*

```
class solve{
	public static String infixToPostfix(String exp) {
		HashMap<Character, Integer> map = new HashMap<>();
		map.put('^', 3);
		map.put('*', 2);
		map.put('/', 2);
		map.put('+', 1);
		map.put('-', 1);
		Stack<Character> st = new Stack<>();
		String res = "";
		for(int i=0;i<exp.length();i++){
			char c = exp.charAt(i);
			if(c=='('){
				st.push(c);
			}else if(c==')'){
				while(!st.isEmpty()){
					char top = st.pop();
					if(top=='(')
						break;
					else
						res += top;
				}
			}else{
				//operator
				if(map.containsKey(c)){
					while(!st.isEmpty()){
						char top = st.peek();
						if(top=='('){
							break;
						}else{
							if(map.get(top)>=map.get(c)){
								res += st.pop();
							}else
								break;
						}
					}
					st.push(c);
				}else{ //operand
					res += c;
				}
			}
		}
```

```
        while(!st.isEmpty()){
            res += st.pop();
        }
        return res;
    }
}
```

Maximize numbers of 1's

Given a binary array **A** of size **N** and an integer **M**. Find the maximum number of consecutive 1's produced by flipping at most M 0's.

Eg. 1 0 0 1 1 0 1 0 1 1 1
M = 2
Output : 8

```
public static int maximumOnes(int a[],int n,int m){
    int max = 0;
    int l = 0, r = 0, count = 0;
    while(r<n){
        if(count<=m){
            if(a[r]==0)
                count++;
            r++;
        }
        if(count>m){
            if(a[l]==0)
                count--;
            l++;
        }
        if((r-l>max)&&(count<=m))
            max = r-l;
    }
    return max;
}
```

Multiply two string

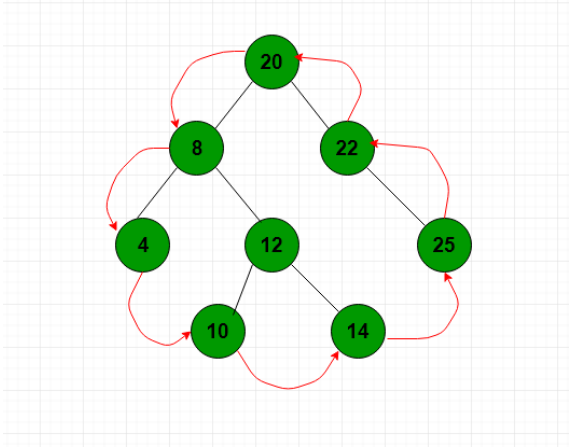
Given two numbers as stings **a and b** your task is to **multiply** them. The output must not contain leading zeroes.

```
public static String multiplyStrings(String a, String b){
    int flag = 0;
    if(a.charAt(0)=='-' && b.charAt(0)!='-'){
        flag = 1;
```

```
        a = a.substring(1);
}
if(a.charAt(0)!='-' && b.charAt(0)=='-'){
        flag = 1;
        b = b.substring(1);
}
if(a.charAt(0)=='-' && b.charAt(0)=='-'){
        a = a.substring(1);
        b = b.substring(1);
}

//check for zero
int f = 0;
for(int i=0;i<a.length();i++){
        if(a.charAt(i)!='0'){
                f = 1;
                break;
        }
}
if(f==0){
        return "0";
}
f = 0;
for(int i=0;i<b.length();i++){
        if(b.charAt(i)!='0'){
                f = 1;
                break;
        }
}
if(f==0){
        return "0";
}

int m[] = new int[a.length()+b.length()];
String s1 = new StringBuffer(a).reverse().toString();
String s2 = new StringBuffer(b).reverse().toString();

for(int i=0;i<s1.length();i++){
        for(int j=0;j<s2.length();j++){
                m[i+j] += (s1.charAt(i)-'0')*(s2.charAt(j)-'0');
        }
}
```

```
        String product = new String();
        for(int i=0;i<m.length;i++){
                int digit = m[i]%10;
                int carry = m[i]/10;
                if(i+1<m.length){
                        m[i+1] += carry;
                }
                product = digit+product;
        }
        //for leading zero
        if(product.length()>1 && product.charAt(0)=='0'){
                product = product.substring(1);
        }
        //for negative
        if(flag == 1){
                product = '-'+product;
        }
        return product;
}
```

Boundary Traversal of Binary Tree



20 8 4 10 14 25 22 .

```
class Solution{
        void printRightBoundary(Node node){
                if(node!=null){
                        if(node.right!=null){
                                printRightBoundary(node.right);
                                System.out.print(node.data+" ");
                        }else if(node.left !=null){
```

```
                printRightBoundary(node.left);
                System.out.print(node.data+" ");
            }
        }
    }
    void printLeaves(Node node){
        if(node!=null){
            printLeaves(node.left);
            if(node.left==null && node.right==null)
                System.out.print(node.data+" ");
            printLeaves(node.right);
        }
    }
    void printLeftBoundary(Node node){
        if(node!=null){
            if(node.left!=null){
                System.out.print(node.data+" ");
                printLeftBoundary(node.left);
            }else if(node.right !=null){
                System.out.print(node.data+" ");
                printLeftBoundary(node.right);
            }
        }
    }
    void printBoundary(Node node){
        if(node==null)
            return;
        System.out.print(node.data+" ");
        //print left boundary
        printLeftBoundary(node.left);
        //print leaves
        printLeaves(node.left);
        printLeaves(node.right);
        //print right boundary
        printRightBoundary(node.right);
    }
}
```

Next Greater Element with same Digits

Given an integer N, the task is to find the next greater element with the same digits. If no such number exists the return -1.

```
static int nextPermutation(int n){
        String str = Integer.toString(n);
        char arr[] = str.toCharArray();
        int i;
        for(i=str.length()-1;i>0;i--){
                if(arr[i]>arr[i-1]){
                        break;
                }
        }
        if(i==0)
                return -1;
        int min = i;
        for(int j=i+1;j<str.length();j++){
                if(arr[j]>arr[i-1] && arr[j]<arr[min]){
                        min = j;
                }
        }
        //swap i-1 and min
        char temp = arr[i-1];
        arr[i-1] = arr[min];
        arr[min] = temp;
        Arrays.sort(arr, i, str.length());
        int res = 0;
        for(i=0;i<str.length();i++){
                int val = arr[i] - '0';
                res = res * 10 + val;
        }
        return res;
}
```

Largest BST

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.

```
class Solution{
        static class Object{
                boolean isBst;
                int res;
                int min,max;
                Object(boolean isBst, int res, int min,int max){
                        this.isBst = isBst;
                        this.res = res;
```

```
                this.min = min;
                this.max = max;
            }
        }
        static Object find(Node root){
            if(root==null)
                return null;
            if(root.left==null && root.right==null){
                return new Object(true, 1, root.data, root.data);
            }
            //PostOrderTraversal
            Object l = find(root.left);
            Object r = find(root.right);
            if(l==null){
                if(r.isBst && root.data<r.min)
                    return new Objecr(true, r.res+1, root.data, r.max);
                return new Object(false, r.res, 0,0);
            }
            if(r==null){
                if(l.isBst && root.data>l.max)
                    return new Object(true, l.res+1, l.min, root.data);
                return new Object(false, l.res, 0, 0);
            }
            if(l.isBst && root.data>l.max && r.isBst && root.data<r.min)
                return new Object(true, l.res+r.res+1, l.min, r.max);
            return new Objec(false, Math.max(l.res, r.res), 0, 0);
        }
        // Return the size of the largest sub-tree which is also a BST
        static int largestBst(Node root){
            Object n = find(root);
            return n.res;
        }
}
```

Longest Palindrome Substring

Given a string S, find the longest palindromic substring in S. **Substring of string S:** S[i . . . . j] where 0 ≤ i ≤ j < len(S)**. Palindrome string:** A string which reads the same backwards. More formally, S is palindrome if reverse(S) = S. **Incase of conflict**, return the substring which occurs first (with the least starting index).

```
public String longestPalindrome(String s) {
    int n = s.length();
```

```
        boolean dp[][] = new boolean[n][n];
        //for single character string
        int max = 1;
        for(int i=0;i<n;i++)
                dp[i][i] = true;

        //for string length 2
        int start = 0;
        for(int i=0;i<n-1;i++){
                if(s.charAt(i)==s.charAt(i+1)){
                        dp[i][i+1] = true;
                        if(max<2){
                                start = i;
                                max = 2;
                        }
                }
        }

         //for string length greater than 2
        for(int k=3;k<=n;k++){
                for(int i=0;i<n-k+1;i++){
                        int j = i+k-1;
                        if(dp[i+1][j-1] && s.charAt(i)==s.charAt(j)){
                                dp[i][j] = true;
                                if(k>max){
                                        max = k;
                                        start = i;
                                }
                        }
                }
        }
        return s.substring(start, start+max);
}
```

Convert Sorted Array to Binary Search Tree

Given a sorted array of distinct integers of size N, the task is to convert it into a height-balanced Binary Search Tree.

```
class Sol{
        static Node insert(int a[], int l, int r){
                if(l>r)
                        return null;
```

```
            int m = (l+r)/2;
            Node n = new Node(a[m]);
            n.left = insert(a, l, m-1);
            n.right = insert(a, m+1, r);
            return n;
        }
        public static Node sortedArrayToBST(int a[], int n){
            return insert(a, 0, n-1);
        }
}
```