FIND THE MISSING NUMBER IN AN ARRAY

// xor

//T- O(2N)

//S- O(1)

class Solution {

    public int missingNumber(int[] nums) {

        int xor = 0;

        int n = nums.length;

        for (int i = 0; i<=n; i++){

            xor = xor^i;

        }

        for (int i = 0; i<n; i++){

            xor = xor^nums[i];

        }

        return xor;

    }

}

MAXIMUM CONSECUTIVE ONES

// T- O(N)

// S- O(1)

// ptr concept and logic

class Solution {

    public int findMaxConsecutiveOnes(int[] nums) {

        int c = 0;

        int n = nums.length;

        int maxx = Integer.MIN\_VALUE;

        for (int i = 0; i<n; i++){

            if (nums[i] == 0) {

                maxx = Math.max(c, maxx);

                c = 0;

            }else{

                c++;

            }

        }

        maxx = Math.max(c, maxx);

        return maxx;

    }

}

JUMP GAME II

// T- O(N2)

// S- O(2N) - dp array and recursion stack space

class Solution {

    public static void recur(int ind, int[] arr, int[] dp) {

        if (ind < 0)

            return;

        else if (arr[ind] == 0) {

            dp[ind] = Integer.MAX\_VALUE;

        } else {

            int minn = Integer.MAX\_VALUE;

            int val = arr[ind];

            for (int i = 1; i <= val; i++) {

                int idx = ind + i;

                if (idx > arr.length - 1) break;

                int value = dp[idx];

                if (value < minn)

                    minn = value;

            }

            if (minn == Integer.MAX\_VALUE) dp[ind] = minn;

            else dp[ind] = 1 + minn;

        }

        recur(ind - 1, arr, dp);

    }

    public int jump(int[] nums) {

        int n = nums.length;

        if (n == 1)

            return 0;

        if (n == 2)

            return 1;

        int[] dp = new int[n];

        dp[n - 1] = 0;

        recur(n - 2, nums, dp);

        return dp[0];

    }

}

GROUP ANAGRAMS

    // T- O(n\*k\*logk)

    // S- O(n\*k)

    class Solution {

        public List<List<String>> groupAnagrams(String[] strs) {

            Map<String, List<String>> map1 = new HashMap<String, List<String>>();

            for (int i = 0; i<strs.length; i++){

                char[] tempArray = strs[i].toCharArray();

                Arrays.sort(tempArray);

                String newString = new String(tempArray);

                if (!map1.containsKey(newString)){

                    List<String> newLst = new ArrayList<String>();

                    newLst.add(strs[i]);

                    map1.put(newString, newLst);

                }else {

                    List<String> l = map1.get(newString);

                    l.add(strs[i]);

                    map1.put(newString,l);

                }

            }

            List<List<String>> lstAns = new ArrayList<List<String>>();

            for (List<String> element: map1.values()){

                lstAns.add(element);

            }

            return lstAns;

        }

    }

PEAK INDEX IN MOUNTAIN ARRAY

// T- O(N)

// S- O(1)

class Solution {

    public int peakIndexInMountainArray(int[] arr) {

        int n = arr.length;

        int peak = -1;

        for (int i = 1; i<n; i++){

            if (arr[i-1]<arr[i] && arr[i+1]<arr[i]){

                peak = i;

                break;

            }

        }

        return peak;

    }

}

NUMBER OF ISLANDS

// T- O(n\*m)

// S- O(min(m,n))

public class Solution {

private int n;

private int m;

public int numIslands(char[][] grid) {

    int count = 0;

    n = grid.length;

    if (n == 0) return 0;

    m = grid[0].length;

    for (int i = 0; i < n; i++){

        for (int j = 0; j < m; j++)

            if (grid[i][j] == '1') {

                DFSMarking(grid, i, j);

                ++count;

            }

    }

    return count;

}

private void DFSMarking(char[][] grid, int i, int j) {

    if (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] != '1') return;

    grid[i][j] = '0';

    DFSMarking(grid, i + 1, j);

    DFSMarking(grid, i - 1, j);

    DFSMarking(grid, i, j + 1);

    DFSMarking(grid, i, j - 1);

}

}