Q1. Longest Nice Substring Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string

```
class Solution {
    public String longestNiceSubstring(String s) {
         if (s.length() < 2) return "";
        char[] arr = s.toCharArray();
        Set<Character> set = new HashSet<>();
        for (char c: arr) set.add(c);
        for (int i = 0; i < arr.length; i++) {
            char c = arr[i];
            if (set.contains(Character.toUpperCase(c)) &&
set.contains(Character.toLowerCase(c))) continue;
            String sub1 = longestNiceSubstring(s.substring(0, i));
            String sub2 = longestNiceSubstring(s.substring(i+1));
            return sub1.length() >= sub2.length() ? sub1 : sub2;
        }
        return s;
    }
}
```

Q2. Longest Even Odd Subarray With Threshold You are given a 0-indexed integer array nums and an integer threshold. Find the length of the longest subarray of nums starting at index l and ending at index r (0 <= l <= r < nums.length) that satisfies the following conditions: nums[l] % 2 == 0 For all indices i in the range [l, r - 1], nums[i] % 2 != nums[i + 1] % 2 For all indices i in the range [l, r], nums[i] <= threshold

```
class Solution {
    public int longestAlternatingSubarray(int[] nums, int val) {
        int n=nums.length;
        int count=0;
        for(int i=0;i<n;i++){
            if(nums[i] %2 ==0 && nums[i]<=val){
                int j=i+1;
                while(j < n && nums[j] <= val && (nums[j] % 2 != nums[j - 1] %
2)){
                    j++;
                }
                count=Math.max(count,j-i);
            }
            else continue;
        }
        return count;
    }
}
```

Q3. Minimum Size Subarray Sum Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

```
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int i=0;
        int sum=0;
        int len=Integer.MAX_VALUE;
        for(int j=0;j<nums.length;j++) {
            sum+=nums[j];
            while(sum>=target) {
                len = Math.min(len,j-i+1);
                sum-=nums[i];
                i++;
            }
        }
        if(len == Integer.MAX_VALUE) return 0;
        return len;
    }
}
```

Q4. Max Consecutive Ones III Given a binary array nums and an integer k, return the maximum number of consecutive 1's in the array if you can flip at most k 0's.

```
public class Solution {
    public int longestOnes(int[] nums, int k) {
    int left = 0, right = 0, maxOnes = 0, zeroCount = 0;
        while (right < nums.length) {
            if (nums[right] == 0) {
                zeroCount++;
            }
            while (zeroCount > k) {
                if (nums[left] == 0) {
                    zeroCount--;
                }
                left++;
            }
            maxOnes = Math.max(maxOnes, right - left + 1);
            right++;
        }
        return maxOnes;
    }
}
```

Q5. Count Number of Nice Subarrays Given an array of integers nums and an integer k. A continuous subarray is called nice if there are k odd numbers on it. Return the number of nice sub-arrays.

```
class Solution {
    public int numberOfSubarrays(int[] nums, int k) {
        int total = 0, sum = 0;
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);

        for (int i = 0; i < nums.length; i++) {
            sum += (nums[i] % 2 == 0) ? 0 : 1;
            int rem = sum - k;

            if (map.containsKey(rem)) {
                total += map.get(rem);
            }

            map.put(sum, map.getOrDefault(sum, 0) + 1);
        }

        return total;
    }
}
```

Q6. Is Subsequence Given two strings s and t, return true if s is a subsequence of t, or false otherwise. A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

```
class Solution {
    public boolean isSubsequence(String s, String t) {
        int index = -1;
        for(char c:s.toCharArray()){
            if(t.indexOf(c)!=-1){
                t = t.substring(t.indexOf(c)+1);
            }else
                return false;
        }

        return true;
    }
}
```

Q7. Sort Colors Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.

```
class Solution {
    public void sortColors(int[] nums) {
        Arrays.sort(nums);

    }
}
```

Q8. Reverse Words in a String Given an input string s, reverse the order of the words. A word is defined as a sequence of non-space characters. The words in s will be separated by at least one space. Return a string of the words in reverse order concatenated by a single space.

```
class Solution {
    public String reverseWords(String s) {
        String b[]=s.trim().split("\\s+");
        String a="";
        for(int i=b.length-1;i>=0;i--)
        {
            a+=b[i]+" ";
        }
        return a.trim();
    }
}
```