# TWO-POINTERS

## Leetcode 167. Two Sum II - Input Array Is Sorted

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int left = 0;
        int right = numbers.length - 1;

        while (left < right) {
            int total = numbers[left] + numbers[right];

            if (total == target) {
                return new int[]{left + 1, right + 1};
            } else if (total > target) {
                right--;
            } else {
                left++;
            }
        }
        return new int[]{-1, -1}; // If no solution is found
    }
}
```

## Leetcode 977. Squares of a Sorted Array

```
class Solution {
    public int[] sortedSquares(int[] nums) {
        int[] res = new int[nums.length];
        int left = 0;
        int right = nums.length - 1;

        for (int i = nums.length - 1; i >= 0; i--) {
            if (Math.abs(nums[left]) > Math.abs(nums[right])) {
                res[i] = nums[left] * nums[left];
                left++;
            } else {
                res[i] = nums[right] * nums[right];
                right--;
            }
        }
        return res;
    }
}
```

## Leetcode 283. Move Zeroes

```
class Solution {
    public void moveZeroes(int[] nums) {
        int left = 0;

        for (int right = 0; right < nums.length; right++) {
            if (nums[right] != 0) {
                int temp = nums[right];
                nums[right] = nums[left];
                nums[left] = temp;
                left++;
            }
        }
    }
}
```

# Max Consecutive Primes

```
static int maxConsecutivePrimes(int[] arr) {
    int left = 0, maxLen = 0;
    for (int right = 0; right < arr.length; right++) {
        if (!isPrime(arr[right])) {
            left = right + 1;
        } else {
            maxLen = Math.max(maxLen, right - left + 1);
        }
    }
    return maxLen;
}


static boolean isPrime(int n) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
  }
return true;
}
```

# Leetcode 763. Partition Labels

```
class Solution {
    public List<Integer> partitionLabels(String s) {
        List<Integer> result = new ArrayList<>();
        int[] lastOccurence = new int[26];

        // Store the last occurence of each character
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            lastOccurence[c - 'a'] = i;
        }

        // Pointers for storing partition boundaries
        int start = 0;
        int end = 0;

        // Traverse and cut the partitions
        for (int i = 0; i < s.length(); i++) {
            end = Math.max(end, lastOccurence[s.charAt(i) - 'a']);
            if (i == end) {
                // All the characters of partition are self-contained and can be cut now.
                result.add(end - start + 1);
                start = i + 1; // Re-assigning the pointer for next partition
            }
        }

        return result;
    }
}
```

## Leetcode 424. Longest Repeating Character Replacement

```
class Solution {
    public int characterReplacement(String s, int k) {
        int ans = 0;
        int n = s.length();
        for (char c = 'A'; c <= 'Z'; c++) {
            int i = 0, j = 0, replaced = 0;
            while (j < n) {
                if (s.charAt(j) == c) {
                    j++;
                } else if (replaced < k) {
                    j++;
                    replaced++;
                } else if (s.charAt(i) == c) {
                    i++;
                } else {
                    i++;
                    replaced--;
                }
                ans = Math.max(ans, j - i);
            }
        }
        return ans;
    }
}
```

## Mentorpick Number of Smaller

```
public int[] solve(int an, int a[], int bn, int b[]) {
    int[] ans = new int[bn];
    int i = 0;
    int j = 0;
    while (i < an && j < bn) {
        if (a[i] < b[j]) {
            i++;
        } else {
            ans[j] = i;
            j++;
        }
    // For remaining elements in b
    while (j < bn) {
        ans[j] = i;
        j++;
    }
    return ans;
}
```

## Mentorpick -Intersection of Two Arrays

```
public List<Integer> intersectionOfTwoArrays(List<Integer> a, List<Integer> b) {
   List<Integer> ans = new ArrayList<>();
   int i = 0, j = 0;
   while (i < a.size() && j < b.size()) {
      if (a.get(i).equals(b.get(j))) {
         if (ans.size() == 0 || !ans.get(ans.size() - 1).equals(a.get(i))) {
            ans.add(a.get(i));
         }
         i++;
         j++;
      } else if (a.get(i) < b.get(j)) {
         i++;
      } else {
         j++;
      }
   }
   return ans;
}
```

## Mentorpick-Is Subsequence

```
public String isSubsequence(String s1, String s2) {
   int i = 0, j = 0;

   while (i < s1.length() && j < s2.length()) {
      if (s1.charAt(i) == s2.charAt(j)) {
         i++;
      }
      j++;
   }

   return (i == s1.length()) ? "True" : "False";
}
```

# Mentorpick-Merge Sorted Array II

```
public void mergeBothInA(int a[], int an, int b[], int bn) {

    List<Integer> ans = new ArrayList<>();

    int i = 0, j = 0;

    // Merge while both arrays have elements
    while (i < an && j < bn) {
        if (a[i] <= b[j]) {
            ans.add(a[i]);
            i++;
        } else {
            ans.add(b[j]);
            j++;
        }
    }

    // Remaining elements of a
    while (i < an) {
        ans.add(a[i]);
        i++;
    }

    // Remaining elements of b
    while (j < bn) {
        ans.add(b[j]);
        j++;
    }

    // Copy back to array a
    int k = 0;
    for (int ele : ans) {
        a[k++] = ele;
    }
}
```