## Two Pointers

### SIMPLE LEVEL (5 QUESTIONS)

### 1. Reverse an Array

**Description:** Reverse the given array in place using two pointers without using extra space.

```
class ReverseArray {
   public static void main(String[] args) {
      int[] arr = {1, 2, 3, 4, 5};
      int left = 0, right = arr.length - 1;

      while (left < right) {
         int temp = arr[left];
         arr[left] = arr[right];
         arr[right] = temp;
         left++;
         right--;
      }

      for (int x : arr)
         System.out.print(x + " ");
   }
}
```

**2. Check Whether an Array Is Palindrome**

**Description:** Check if the array reads the same from both ends using two pointers.

```
class PalindromeArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 2, 1};
        int left = 0, right = arr.length - 1;

        while (left < right) {
            if (arr[left] != arr[right]) {
                System.out.println("Not Palindrome");
                return;
            }
            left++;
            right--;
        }
        System.out.println("Palindrome");
    }
}
```

**3. Find a Pair with Given Sum in Sorted Array**

**Description:** Determine whether any two elements in a sorted array add up to a given sum.

```
class PairSum {
   public static void main(String[] args) {
      int[] arr = {1, 2, 4, 7, 11, 15};
      int sum = 15;
      int left = 0, right = arr.length - 1;

      while (left < right) {
         int s = arr[left] + arr[right];
         if (s == sum) {
            System.out.println("Pair Found");
            return;
         } else if (s < sum) {
            left++;
         } else {
            right--;
         }
      }
      System.out.println("Pair Not Found");
   }
}
```

**4. Move All Zeros to End**

**Description:** Move all zero elements to the end of the array while maintaining the order of non-zero elements.

```
class MoveZeros {
   public static void main(String[] args) {
      int[] arr = {0, 1, 0, 3, 12};
      int left = 0;

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

      for (int x : arr)
         System.out.print(x + " ");
   }
}
```

**5. Count Pairs with Sum Less Than K**

**Description:** Count the number of pairs in a sorted array whose sum is less than a given value.

```
class CountPairs {
   public static void main(String[] args) {
      int[] arr = {1, 2, 3, 4, 5};
      int k = 7;
      int left = 0, right = arr.length - 1, count = 0;

      while (left < right) {
         if (arr[left] + arr[right] < k) {
            count += (right - left);
            left++;
         } else {
            right--;
         }
      }
      System.out.println(count);
   }
}
```

**MODERATE LEVEL (5 QUESTIONS)**

**6. Remove Duplicates from Sorted Array**

**Description:** Remove duplicate elements from a sorted array in place using two pointers.

```
class RemoveDuplicates {
    public static void main(String[] args) {
        int[] arr = {1, 1, 2, 2, 3, 4, 4};
        int index = 1;

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] != arr[i - 1]) {
                arr[index++] = arr[i];
            }
        }

        for (int i = 0; i < index; i++)
            System.out.print(arr[i] + " ");
    }
}
```

**7. Squares of a Sorted Array**

**Description:** Return a new array of squares of each number sorted in non-decreasing order.

```
class SortedSquares {
   public static void main(String[] args) {
      int[] arr = {-4, -1, 0, 3, 10};
      int n = arr.length;
      int[] result = new int[n];
      int left = 0, right = n - 1, pos = n - 1;

      while (left <= right) {
         if (arr[left] * arr[left] > arr[right] * arr[right]) {
            result[pos--] = arr[left] * arr[left];
            left++;
         } else {
            result[pos--] = arr[right] * arr[right];
            right--;
         }
      }

      for (int x : result)
         System.out.print(x + " ");
   }
}
```

**8. Find a Triplet with Given Sum**

**Description:** Check whether any three elements in the array sum up to a given target.

```
class TripletSum {
   public static void main(String[] args) {
      int[] arr = {1, 2, 4, 5, 6};
      int target = 10;

      for (int i = 0; i < arr.length - 2; i++) {
         int left = i + 1, right = arr.length - 1;
         while (left < right) {
            int sum = arr[i] + arr[left] + arr[right];
            if (sum == target) {
               System.out.println("Triplet Found");
               return;
            } else if (sum < target) left++;
            else right--;
         }
      }
      System.out.println("No Triplet");
   }
}
```

**9. Sort an Array of 0s, 1s, and 2s**

**Description:** Sort the array containing only 0s, 1s, and 2s using the Dutch National Flag algorithm.

```
class DutchFlag {
   public static void main(String[] args) {
      int[] arr = {2, 0, 2, 1, 1, 0};
      int low = 0, mid = 0, high = arr.length - 1;

      while (mid <= high) {
         if (arr[mid] == 0) {
            int temp = arr[low];
            arr[low++] = arr[mid];
            arr[mid++] = temp;
         } else if (arr[mid] == 1) {
            mid++;
         } else {
            int temp = arr[mid];
            arr[mid] = arr[high];
            arr[high--] = temp;
         }
      }

      for (int x : arr)
         System.out.print(x + " ");
   }
}
```

**10. Container With Most Water**

**Description:** Find two lines that together with the x-axis form a container holding the maximum water.

```
class MaxWater {
   public static void main(String[] args) {
      int[] height = {1,8,6,2,5,4,8,3,7};
      int left = 0, right = height.length - 1;
      int max = 0;

      while (left < right) {
         int area = Math.min(height[left], height[right]) * (right - left);
         max = Math.max(max, area);
         if (height[left] < height[right]) left++;
         else right--;
      }
      System.out.println(max);
   }
}
```

**HARD LEVEL (5 QUESTIONS)**

**11. 4-Sum Problem**

**Description:** Find all unique quadruplets in the array that sum to a given target value.

```
import java.util.Arrays;

class FourSum {
   public static void main(String[] args) {
      int[] arr = {1, 0, -1, 0, -2, 2};
      int target = 0;
      Arrays.sort(arr);

      for (int i = 0; i < arr.length - 3; i++) {
         for (int j = i + 1; j < arr.length - 2; j++) {
            int left = j + 1, right = arr.length - 1;
            while (left < right) {
               int sum = arr[i] + arr[j] + arr[left] + arr[right];
               if (sum == target) {
                  System.out.println(arr[i] + " " + arr[j] + " " + arr[left] + " " + arr[right]);
                  left++; right--;
               } else if (sum < target) left++;
               else right--;
            }
         }
      }
   }
}
```

**12. Trapping Rain Water**

**Description:** Calculate how much rainwater can be trapped between bars of different heights.

```
class RainWater {
   public static void main(String[] args) {
      int[] height = {4,2,0,3,2,5};
      int left = 0, right = height.length - 1;
      int leftMax = 0, rightMax = 0, water = 0;

      while (left < right) {
         if (height[left] < height[right]) {
            leftMax = Math.max(leftMax, height[left]);
            water += leftMax - height[left++];
         } else {
            rightMax = Math.max(rightMax, height[right]);
            water += rightMax - height[right--];
         }
      }
      System.out.println(water);
   }
}
```

**13. Minimum Difference Pair**

**Description:** Find the minimum difference between any two elements in the array.

```
import java.util.Arrays;

class MinDifference {
   public static void main(String[] args) {
      int[] arr = {3, 8, 15, 17};
      Arrays.sort(arr);
      int minDiff = Integer.MAX_VALUE;

      for (int i = 1; i < arr.length; i++) {
         minDiff = Math.min(minDiff, arr[i] - arr[i - 1]);
      }
      System.out.println(minDiff);
   }
}
```

**14. Longest Mountain in Array**

**Description:** Find the length of the longest subarray that strictly increases and then strictly decreases.

```
class LongestMountain {
   public static void main(String[] args) {
      int[] arr = {2,1,4,7,3,2,5};
      int maxLen = 0;

      for (int i = 1; i < arr.length - 1; i++) {
         if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1]) {
            int left = i, right = i;
            while (left > 0 && arr[left] > arr[left - 1]) left--;
            while (right < arr.length - 1 && arr[right] > arr[right + 1]) right++;
            maxLen = Math.max(maxLen, right - left + 1);
         }
      }
      System.out.println(maxLen);
   }
}
```

**15. Partition Array into Two Equal Sum Subarrays**

**Description:** Check whether the array can be split into two contiguous parts having equal sum.

```
class EqualPartition {
   public static void main(String[] args) {
      int[] arr = {1, 5, 11, 5};
      int total = 0;
      for (int x : arr) total += x;

      int sum = 0;
      for (int i = 0; i < arr.length; i++) {
         sum += arr[i];
         if (sum * 2 == total) {
            System.out.println("Partition Possible");
            return;
         }
      }
      System.out.println("Not Possible");
   }
}
```

**Sliding Window programs**

**SIMPLE LEVEL (5 QUESTIONS)**

**1. Maximum Sum of Subarray of Size K**

**Description:** Find the maximum sum of any contiguous subarray of size k using a fixed-size sliding window.

```
class MaxSumSubarray {
    public static void main(String[] args) {
        int[] arr = {2, 1, 5, 1, 3, 2};
        int k = 3;
        int windowSum = 0, maxSum = 0;

        for (int i = 0; i < arr.length; i++) {
            windowSum += arr[i];

            if (i >= k - 1) {
                maxSum = Math.max(maxSum, windowSum);
                windowSum -= arr[i - (k - 1)];
            }
        }
        System.out.println(maxSum);
    }
}
```

**2. Average of Subarrays of Size K**

**Description:** Compute the average of all contiguous subarrays of size k.

```
class AverageSubarrays {
    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 6, -1, 4, 1, 8, 2};
        int k = 5;
        int windowSum = 0;

        for (int i = 0; i < arr.length; i++) {
            windowSum += arr[i];
            if (i >= k - 1) {
                System.out.println((double) windowSum / k);
                windowSum -= arr[i - (k - 1)];
            }
        }
    }
}
```

**3. Minimum Sum Subarray of Size K**

**Description:** Find the minimum sum of any contiguous subarray of size k.

```
class MinSumSubarray {
    public static void main(String[] args) {
        int[] arr = {2, 1, 5, 1, 3, 2};
        int k = 3;
        int windowSum = 0, minSum = Integer.MAX_VALUE;

        for (int i = 0; i < arr.length; i++) {
            windowSum += arr[i];
            if (i >= k - 1) {
                minSum = Math.min(minSum, windowSum);
                windowSum -= arr[i - (k - 1)];
            }
        }
        System.out.println(minSum);
    }
}
```

**4. Count Subarrays of Size K with Sum Greater Than X**

**Description:** Count the number of subarrays of size k whose sum is greater than a given value x.

```
class CountSubarrays {
   public static void main(String[] args) {
      int[] arr = {2, 3, 4, 1, 5};
      int k = 2, x = 6;
      int windowSum = 0, count = 0;

      for (int i = 0; i < arr.length; i++) {
         windowSum += arr[i];
         if (i >= k - 1) {
            if (windowSum > x) count++;
            windowSum -= arr[i - (k - 1)];
         }
      }
      System.out.println(count);
   }
}
```

**5. Maximum Number of Ones in Subarray of Size K**

**Description:** Find the maximum number of 1s present in any subarray of size k.

```
class MaxOnes {
    public static void main(String[] args) {
        int[] arr = {1, 0, 1, 1, 0, 1};
        int k = 3;
        int count = 0, maxCount = 0;

        for (int i = 0; i < arr.length; i++) {
            count += arr[i];
            if (i >= k - 1) {
                maxCount = Math.max(maxCount, count);
                count -= arr[i - (k - 1)];
            }
        }
        System.out.println(maxCount);
    }
}
```

**MODERATE LEVEL (5 QUESTIONS)**

**6. Longest Subarray with Sum K**

**Description:** Find the length of the longest contiguous subarray whose sum equals k (positive numbers).

```
class LongestSubarraySumK {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 1, 1, 1, 1};
        int k = 5;
        int left = 0, sum = 0, maxLen = 0;

        for (int right = 0; right < arr.length; right++) {
            sum += arr[right];
            while (sum > k) {
                sum -= arr[left++];
            }
            if (sum == k) {
                maxLen = Math.max(maxLen, right - left + 1);
            }
        }
        System.out.println(maxLen);
    }
}
```

**7. Smallest Subarray with Sum ≥ S**

**Description:** Find the minimum length of a contiguous subarray whose sum is greater than or equal to S.

```
class MinSubarrayLength {
   public static void main(String[] args) {
      int[] arr = {2, 1, 5, 2, 3, 2};
      int s = 7;
      int left = 0, sum = 0, minLen = Integer.MAX_VALUE;

      for (int right = 0; right < arr.length; right++) {
         sum += arr[right];
         while (sum >= s) {
            minLen = Math.min(minLen, right - left + 1);
            sum -= arr[left++];
         }
      }
      System.out.println(minLen == Integer.MAX_VALUE ? 0 : minLen);
   }
}
```

**8. Longest Substring with At Most K Distinct Characters**

**Description:** Find the length of the longest substring containing at most k distinct characters.

```
import java.util.HashMap;

class LongestKDistinct {
   public static void main(String[] args) {
      String str = "araaci";
      int k = 2;
      int left = 0, maxLen = 0;
      HashMap<Character, Integer> map = new HashMap<>();

      for (int right = 0; right < str.length(); right++) {
         map.put(str.charAt(right), map.getOrDefault(str.charAt(right), 0) + 1);

         while (map.size() > k) {
            char c = str.charAt(left);
            map.put(c, map.get(c) - 1);
            if (map.get(c) == 0) map.remove(c);
            left++;
         }
         maxLen = Math.max(maxLen, right - left + 1);
      }
      System.out.println(maxLen);
   }
}
```

**9. Maximum Sum Subarray with All Unique Elements**

**Description:** Find the maximum sum of a subarray that contains only distinct elements.

```
import java.util.HashSet;

class MaxUniqueSum {
   public static void main(String[] args) {
      int[] arr = {4, 2, 4, 5, 6};
      int left = 0, sum = 0, maxSum = 0;
      HashSet<Integer> set = new HashSet<>();

      for (int right = 0; right < arr.length; right++) {
         while (set.contains(arr[right])) {
            set.remove(arr[left]);
            sum -= arr[left++];
         }
         set.add(arr[right]);
         sum += arr[right];
         maxSum = Math.max(maxSum, sum);
      }
      System.out.println(maxSum);
   }
}
```

**10. Count Substrings with Exactly K Distinct Characters**

**Description:** Count all substrings of a string that contain exactly k distinct characters.

```
class CountExactKDistinct {
    public static void main(String[] args) {
        String s = "pqpqs";
        int k = 2;
        System.out.println(countAtMost(s, k) - countAtMost(s, k - 1));
    }

    static int countAtMost(String s, int k) {
        int left = 0, count = 0;
        int[] freq = new int[256];

        for (int right = 0; right < s.length(); right++) {
            if (freq[s.charAt(right)]++ == 0) k--;
            while (k < 0) {
                if (--freq[s.charAt(left++)] == 0) k++;
            }
            count += right - left + 1;
        }
        return count;
    }
}
```

**HARD LEVEL (5 QUESTIONS)**

**11. Longest Substring Without Repeating Characters**

**Description:** Find the length of the longest substring that contains no repeating characters.

```
import java.util.HashSet;

class LongestUniqueSubstring {
    public static void main(String[] args) {
        String s = "abcabcbb";
        int left = 0, maxLen = 0;
        HashSet<Character> set = new HashSet<>();

        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            maxLen = Math.max(maxLen, right - left + 1);
        }
        System.out.println(maxLen);
    }
}
```

**12. Longest Substring After Replacing at Most K Characters**

**Description:** Find the length of the longest substring after replacing at most k characters to make all characters same.

```
class CharacterReplacement {
   public static void main(String[] args) {
      String s = "AABABBA";
      int k = 1;
      int left = 0, maxCount = 0, maxLen = 0;
      int[] freq = new int[26];

      for (int right = 0; right < s.length(); right++) {
         maxCount = Math.max(maxCount, ++freq[s.charAt(right) - 'A']);

         while ((right - left + 1) - maxCount > k) {
            freq[s.charAt(left) - 'A']--;
            left++;
         }
         maxLen = Math.max(maxLen, right - left + 1);
      }
      System.out.println(maxLen);
   }
}
```

**13. Minimum Window Substring**

**Description:** Find the smallest substring in a string that contains all characters of another string.

```
import java.util.HashMap;

class MinWindowSubstring {
    public static void main(String[] args) {
        String s = "ADOBECODEBANC", t = "ABC";
        HashMap<Character, Integer> map = new HashMap<>();
        for (char c : t.toCharArray())
            map.put(c, map.getOrDefault(c, 0) + 1);

        int left = 0, count = map.size(), minLen = Integer.MAX_VALUE, start = 0;

        for (int right = 0; right < s.length(); right++) {
            char c = s.charAt(right);
            if (map.containsKey(c)) {
                map.put(c, map.get(c) - 1);
                if (map.get(c) == 0) count--;
            }

            while (count == 0) {
                if (right - left + 1 < minLen) {
                    minLen = right - left + 1;
                    start = left;
                }
                char lc = s.charAt(left++);
                if (map.containsKey(lc)) {
                    map.put(lc, map.get(lc) + 1);
                    if (map.get(lc) > 0) count++;
                }
            }
        }
        System.out.println(minLen == Integer.MAX_VALUE ? "" : s.substring(start, start +
minLen));
    }
}
```

**14. Sliding Window Maximum**

**Description:** Find the maximum element in every contiguous subarray of size k.

```
import java.util.Deque;
import java.util.LinkedList;

class SlidingWindowMax {
   public static void main(String[] args) {
      int[] arr = {1,3,-1,-3,5,3,6,7};
      int k = 3;
      Deque<Integer> dq = new LinkedList<>();

      for (int i = 0; i < arr.length; i++) {
         while (!dq.isEmpty() && dq.peekFirst() <= i - k)
            dq.pollFirst();

         while (!dq.isEmpty() && arr[dq.peekLast()] < arr[i])
            dq.pollLast();

         dq.addLast(i);

         if (i >= k - 1)
            System.out.print(arr[dq.peekFirst()] + " ");
      }
   }
}
```

**15. Count Anagram Substrings**

**Description:** Count the number of substrings of a string that are anagrams of a given pattern.

```
class CountAnagrams {
    public static void main(String[] args) {
        String text = "cbaebabacd";
        String pat = "abc";
        int[] freq = new int[26];
        for (char c : pat.toCharArray())
            freq[c - 'a']++;

        int left = 0, count = pat.length(), result = 0;

        for (int right = 0; right < text.length(); right++) {
            if (freq[text.charAt(right) - 'a']-- > 0)
                count--;

            if (right - left + 1 == pat.length()) {
                if (count == 0) result++;
                if (++freq[text.charAt(left) - 'a'] > 0)
                    count++;
                left++;
            }
        }
        System.out.println(result);
    }
}
```

**Prefix Sum programs**

**SIMPLE LEVEL (5 QUESTIONS)**

**1. Prefix Sum Array Construction**

**Description:** Construct the prefix sum array where each element stores the sum of all previous elements including itself.

```
class PrefixSumArray {
   public static void main(String[] args) {
      int[] arr = {1, 2, 3, 4, 5};
      int[] prefix = new int[arr.length];

      prefix[0] = arr[0];
      for (int i = 1; i < arr.length; i++) {
         prefix[i] = prefix[i - 1] + arr[i];
      }

      for (int x : prefix)
         System.out.print(x + " ");
   }
}
```

**2. Range Sum Query**

**Description:** Find the sum of elements between indices L and R using the prefix sum technique.

```
class RangeSum {
   public static void main(String[] args) {
      int[] arr = {1, 3, 5, 7, 9};
      int L = 1, R = 3;
      int[] prefix = new int[arr.length];

      prefix[0] = arr[0];
      for (int i = 1; i < arr.length; i++)
         prefix[i] = prefix[i - 1] + arr[i];

      int sum = (L == 0) ? prefix[R] : prefix[R] - prefix[L - 1];
      System.out.println(sum);
   }
}
```

**3. Check If Any Subarray Has Sum Zero**

**Description:** Determine whether there exists a subarray whose sum is zero using prefix sums.

```
import java.util.HashSet;

class ZeroSumSubarray {
   public static void main(String[] args) {
      int[] arr = {1, -1, 2, -2, 3};
      HashSet<Integer> set = new HashSet<>();
      int sum = 0;

      for (int x : arr) {
         sum += x;
         if (sum == 0 || set.contains(sum)) {
            System.out.println("Yes");
            return;
         }
         set.add(sum);
      }
      System.out.println("No");
   }
}
```

**4. Find Cumulative Sum of Array**

**Description:** Compute the cumulative sum of the array where each element is replaced by the sum up to that index.

```
class CumulativeSum {
    public static void main(String[] args) {
        int[] arr = {5, 10, 15};
        for (int i = 1; i < arr.length; i++) {
            arr[i] += arr[i - 1];
        }

        for (int x : arr)
            System.out.print(x + " ");
    }
}
```

**5. Sum of Even Indexed Elements Using Prefix Sum**

**Description:** Precompute prefix sums to efficiently find the sum of elements at even indices.

```
class EvenIndexSum {
    public static void main(String[] args) {
        int[] arr = {4, 5, 6, 7, 8};
        int[] prefix = new int[arr.length];

        prefix[0] = arr[0];
        for (int i = 1; i < arr.length; i++) {
            prefix[i] = prefix[i - 1] + (i % 2 == 0 ? arr[i] : 0);
        }

        System.out.println(prefix[arr.length - 1]);
    }
}
```

**MODERATE LEVEL (5 QUESTIONS)**

**6. Count Subarrays with Given Sum K**

**Description:** Count the number of subarrays whose sum equals a given value k.

```
import java.util.HashMap;

class SubarraySumK {
   public static void main(String[] args) {
      int[] arr = {1, 2, 3, -2, 5};
      int k = 5;
      HashMap<Integer, Integer> map = new HashMap<>();
      map.put(0, 1);
      int sum = 0, count = 0;

      for (int x : arr) {
         sum += x;
         if (map.containsKey(sum - k))
            count += map.get(sum - k);
         map.put(sum, map.getOrDefault(sum, 0) + 1);
      }
      System.out.println(count);
   }
}
```

**7. Longest Subarray with Equal Number of 0s and 1s**

**Description:** Find the length of the longest subarray containing equal number of 0s and 1s.

```
import java.util.HashMap;

class EqualZeroOne {
   public static void main(String[] args) {
      int[] arr = {0, 1, 0, 1, 1, 1, 0};
      HashMap<Integer, Integer> map = new HashMap<>();
      map.put(0, -1);
      int sum = 0, maxLen = 0;

      for (int i = 0; i < arr.length; i++) {
         sum += (arr[i] == 0 ? -1 : 1);
         if (map.containsKey(sum)) {
            maxLen = Math.max(maxLen, i - map.get(sum));
         } else {
            map.put(sum, i);
         }
      }
      System.out.println(maxLen);
   }
}
```

**8. Maximum Subarray Sum Using Prefix Sum**

**Description:** Find the maximum sum of any contiguous subarray using the prefix sum approach.

```
class MaxSubarrayPrefix {
   public static void main(String[] args) {
      int[] arr = {-2, 1, -3, 4, -1, 2, 1};
      int minPrefix = 0, maxSum = Integer.MIN_VALUE;
      int prefix = 0;

      for (int x : arr) {
         prefix += x;
         maxSum = Math.max(maxSum, prefix - minPrefix);
         minPrefix = Math.min(minPrefix, prefix);
      }
      System.out.println(maxSum);
   }
}
```

**9. Count Subarrays with Sum Divisible by K**

**Description:** Count the number of subarrays whose sum is divisible by k.

```
import java.util.HashMap;

class SubarrayDivisibleK {
   public static void main(String[] args) {
      int[] arr = {4, 5, 0, -2, -3, 1};
      int k = 5;
      HashMap<Integer, Integer> map = new HashMap<>();
      map.put(0, 1);
      int sum = 0, count = 0;

      for (int x : arr) {
         sum += x;
         int rem = ((sum % k) + k) % k;
         count += map.getOrDefault(rem, 0);
         map.put(rem, map.getOrDefault(rem, 0) + 1);
      }
      System.out.println(count);
   }
}
```

**10. Find Pivot Index**

**Description:** Find the index where the sum of elements on the left equals the sum of elements on the right.

```
class PivotIndex {
   public static void main(String[] args) {
      int[] arr = {1, 7, 3, 6, 5, 6};
      int total = 0, leftSum = 0;

      for (int x : arr) total += x;

      for (int i = 0; i < arr.length; i++) {
         if (leftSum == total - leftSum - arr[i]) {
            System.out.println(i);
            return;
         }
         leftSum += arr[i];
      }
      System.out.println(-1);
   }
}
```

**HARD LEVEL (5 QUESTIONS)**

**11. Longest Subarray with Sum Zero**

**Description:** Find the length of the longest contiguous subarray whose sum is zero.

```
import java.util.HashMap;

class LongestZeroSum {
   public static void main(String[] args) {
      int[] arr = {15, -2, 2, -8, 1, 7, 10, 23};
      HashMap<Integer, Integer> map = new HashMap<>();
      int sum = 0, maxLen = 0;

      for (int i = 0; i < arr.length; i++) {
         sum += arr[i];
         if (sum == 0)
            maxLen = i + 1;
         if (map.containsKey(sum)) {
            maxLen = Math.max(maxLen, i - map.get(sum));
         } else {
            map.put(sum, i);
         }
      }
      System.out.println(maxLen);
   }
}
```

**12. Count Subarrays with Equal 0s, 1s and 2s**

**Description:** Count the number of subarrays having equal numbers of 0s, 1s, and 2s.

```
import java.util.HashMap;

class Equal012 {
    public static void main(String[] args) {
        int[] arr = {0, 1, 0, 2, 0, 1, 2};
        int c0 = 0, c1 = 0, c2 = 0;
        HashMap<String, Integer> map = new HashMap<>();
        map.put("0#0", 1);
        int count = 0;

        for (int x : arr) {
            if (x == 0) c0++;
            else if (x == 1) c1++;
            else c2++;

            String key = (c1 - c0) + "#" + (c2 - c1);
            count += map.getOrDefault(key, 0);
            map.put(key, map.getOrDefault(key, 0) + 1);
        }
        System.out.println(count);
    }
}
```

**13. Maximum Length Subarray with Sum Divisible by K**

**Description:** Find the length of the longest subarray whose sum is divisible by k.

```
import java.util.HashMap;

class LongestDivisibleK {
   public static void main(String[] args) {
      int[] arr = {2, 7, 6, 1, 4, 5};
      int k = 3;
      HashMap<Integer, Integer> map = new HashMap<>();
      map.put(0, -1);
      int sum = 0, maxLen = 0;

      for (int i = 0; i < arr.length; i++) {
         sum += arr[i];
         int rem = ((sum % k) + k) % k;
         if (map.containsKey(rem)) {
            maxLen = Math.max(maxLen, i - map.get(rem));
         } else {
            map.put(rem, i);
         }
      }
      System.out.println(maxLen);
   }
}
```

**14. Maximum Subarray Sum After One Deletion**

**Description:** Find the maximum subarray sum when at most one element can be deleted.

```
class MaxSubarrayOneDelete {
   public static void main(String[] args) {
      int[] arr = {1, -2, 0, 3};
      int n = arr.length;
      int[] fw = new int[n];
      int[] bw = new int[n];

      fw[0] = arr[0];
      bw[n - 1] = arr[n - 1];

      for (int i = 1; i < n; i++)
         fw[i] = Math.max(arr[i], fw[i - 1] + arr[i]);

      for (int i = n - 2; i >= 0; i--)
         bw[i] = Math.max(arr[i], bw[i + 1] + arr[i]);

      int maxSum = Integer.MIN_VALUE;
      for (int i = 0; i < n; i++) {
         maxSum = Math.max(maxSum, fw[i]);
         if (i > 0 && i < n - 1)
            maxSum = Math.max(maxSum, fw[i - 1] + bw[i + 1]);
      }
      System.out.println(maxSum);
   }
}
```

**15. Count Number of Subarrays with XOR = K**

**Description:** Count the number of subarrays whose XOR is equal to a given value k.

```
import java.util.HashMap;

class SubarrayXORK {
   public static void main(String[] args) {
      int[] arr = {4, 2, 2, 6, 4};
      int k = 6;
      HashMap<Integer, Integer> map = new HashMap<>();
      map.put(0, 1);
      int xor = 0, count = 0;

      for (int x : arr) {
         xor ^= x;
         if (map.containsKey(xor ^ k))
            count += map.get(xor ^ k);
         map.put(xor, map.getOrDefault(xor, 0) + 1);
      }
      System.out.println(count);
   }
}
```

All programs on

1. **Maximum subarray sum with constraints**
2. **Partition array into subarrays with certain properties**
3. **String matching / anagram-related problems**

**SIMPLE LEVEL (10 PROGRAMS)**

**1. Maximum Subarray Sum (Kadane's Algorithm)**

**Description:** Find the maximum possible sum of a contiguous subarray.

```
class MaxSubarray {
   public static void main(String[] args) {
      int[] arr = {-2,1,-3,4,-1,2,1};
      int max = arr[0], curr = arr[0];

      for (int i = 1; i < arr.length; i++) {
         curr = Math.max(arr[i], curr + arr[i]);
         max = Math.max(max, curr);
      }
      System.out.println(max);
   }
}
```

**2. Maximum Subarray Sum of Size K**

**Description:** Find the maximum sum of any contiguous subarray of fixed size k.

```
class MaxSumSizeK {
   public static void main(String[] args) {
      int[] arr = {2,1,5,1,3,2};
      int k = 3, sum = 0, max = 0;

      for (int i = 0; i < arr.length; i++) {
         sum += arr[i];
         if (i >= k - 1) {
            max = Math.max(max, sum);
            sum -= arr[i - (k - 1)];
         }
      }
      System.out.println(max);
   }
}
```

### 3. Check If Array Can Be Split into Two Equal Sum Parts

**Description:** Determine whether the array can be partitioned into two contiguous parts with equal sum.

```
class EqualPartition {
   public static void main(String[] args) {
      int[] arr = {1,5,11,5};
      int total = 0, sum = 0;

      for (int x : arr) total += x;
      for (int x : arr) {
         sum += x;
         if (sum * 2 == total) {
            System.out.println("Yes");
            return;
         }
      }
      System.out.println("No");
   }
}
```

**4. Check if Two Strings Are Anagrams**

**Description:** Check whether two strings contain the same characters with the same frequencies.

```
class AnagramCheck {
   public static void main(String[] args) {
      String a = "listen", b = "silent";
      int[] freq = new int[26];

      for (char c : a.toCharArray()) freq[c - 'a']++;
      for (char c : b.toCharArray()) freq[c - 'a']--;

      for (int x : freq)
         if (x != 0) {
            System.out.println("No");
            return;
         }
      System.out.println("Yes");
   }
}
```

**5. Maximum Prefix Sum**

**Description:** Find the maximum sum among all prefixes of the array.

```
class MaxPrefixSum {
   public static void main(String[] args) {
      int[] arr = {3,-2,5,-1};
      int sum = 0, max = Integer.MIN_VALUE;

      for (int x : arr) {
         sum += x;
         max = Math.max(max, sum);
      }
      System.out.println(max);
   }
}
```

**6. Count Partitions with Equal Even–Odd Sum**

**Description:** Count indices where left and right subarrays have equal even-odd difference.

```
class EvenOddPartition {
   public static void main(String[] args) {
      int[] arr = {2,1,6,4};
      int total = 0, count = 0;

      for (int x : arr) total += (x % 2 == 0 ? x : -x);

      int curr = 0;
      for (int x : arr) {
         curr += (x % 2 == 0 ? x : -x);
         if (curr * 2 == total) count++;
      }
      System.out.println(count);
   }
}
```

**7. Maximum Subarray Sum Ending at Each Index**

**Description:** Print the maximum subarray sum ending at every index.

```
class MaxEndingHere {
    public static void main(String[] args) {
        int[] arr = {1,-2,3,4,-1};
        int curr = arr[0];
        System.out.print(curr + " ");

        for (int i = 1; i < arr.length; i++) {
            curr = Math.max(arr[i], curr + arr[i]);
            System.out.print(curr + " ");
        }
    }
}
```

**8. Find All Prefixes with Positive Sum**

**Description:** Count how many prefixes have a strictly positive sum.

```
class PositivePrefixes {
   public static void main(String[] args) {
      int[] arr = {-1,2,3,-2};
      int sum = 0, count = 0;

      for (int x : arr) {
         sum += x;
         if (sum > 0) count++;
      }
      System.out.println(count);
   }
}
```

**9. Check If String Contains Any Anagram of Pattern**

**Description:** Check whether any substring of the string is an anagram of a given pattern.

```
class ContainsAnagram {
    public static void main(String[] args) {
        String s = "cbaebabacd", p = "abc";
        int[] freq = new int[26];

        for (char c : p.toCharArray()) freq[c - 'a']++;
        int count = p.length(), left = 0;

        for (int right = 0; right < s.length(); right++) {
            if (freq[s.charAt(right) - 'a']-- > 0) count--;
            if (right - left + 1 == p.length()) {
                if (count == 0) {
                    System.out.println("Yes");
                    return;
                }
                if (++freq[s.charAt(left++) - 'a'] > 0) count++;
            }
        }
        System.out.println("No");
    }
}
```

**10. Maximum Single Element Subarray**

**Description:** Find the maximum subarray sum when only one element is allowed.

```
class MaxSingle {
   public static void main(String[] args) {
      int[] arr = {-3,-1,-2};
      int max = arr[0];

      for (int x : arr)
         max = Math.max(max, x);

      System.out.println(max);
   }
}
```

**MODERATE LEVEL (10 PROGRAMS)**

**11. Maximum Subarray Sum with At Least K Elements**

**Description:** Find the maximum subarray sum with length at least k.

```
class MaxAtLeastK {
   public static void main(String[] args) {
      int[] arr = {1,2,3,-10,5};
      int k = 2;
      int[] maxEnd = new int[arr.length];

      maxEnd[0] = arr[0];
      for (int i = 1; i < arr.length; i++)
         maxEnd[i] = Math.max(arr[i], maxEnd[i-1] + arr[i]);

      int sum = 0, max = Integer.MIN_VALUE;
      for (int i = 0; i < k; i++) sum += arr[i];
      max = sum;

      for (int i = k; i < arr.length; i++) {
         sum += arr[i] - arr[i-k];
         max = Math.max(max, sum + Math.max(0, maxEnd[i-k]));
      }
      System.out.println(max);
   }
}
```

**12. Partition Array into K Equal Sum Subarrays**

**Description:** Check whether the array can be partitioned into k contiguous subarrays with equal sum.

```
class KPartition {
   public static void main(String[] args) {
      int[] arr = {2,1,1,2,1,1};
      int k = 3, sum = 0, curr = 0, count = 0;

      for (int x : arr) sum += x;
      if (sum % k != 0) {
         System.out.println("No");
         return;
      }

      for (int x : arr) {
         curr += x;
         if (curr == sum / k) {
            count++;
            curr = 0;
         }
      }
      System.out.println(count == k ? "Yes" : "No");
   }
}
```

**13. Count All Anagram Substrings**

**Description:** Count the number of substrings that are anagrams of a given pattern.

```
class CountAnagrams {
    public static void main(String[] args) {
        String s = "abab", p = "ab";
        int[] freq = new int[26];
        for (char c : p.toCharArray()) freq[c - 'a']++;

        int left = 0, count = p.length(), ans = 0;
        for (int right = 0; right < s.length(); right++) {
            if (freq[s.charAt(right) - 'a']-- > 0) count--;
            if (right - left + 1 == p.length()) {
                if (count == 0) ans++;
                if (++freq[s.charAt(left++) - 'a'] > 0) count++;
            }
        }
        System.out.println(ans);
    }
}
```

**14. Maximum Sum Subarray with No More Than K Negatives**

**Description:** Find the maximum subarray sum containing at most k negative numbers.

```
class MaxKNegatives {
   public static void main(String[] args) {
      int[] arr = {1,-2,3,-4,5};
      int k = 1, left = 0, neg = 0, sum = 0, max = 0;

      for (int right = 0; right < arr.length; right++) {
         sum += arr[right];
         if (arr[right] < 0) neg++;

         while (neg > k) {
            if (arr[left] < 0) neg--;
            sum -= arr[left++];
         }
         max = Math.max(max, sum);
      }
      System.out.println(max);
   }
}
```

**15. Split Binary Array into Equal 0s and 1s**

**Description:** Count partitions where left and right subarrays have equal numbers of 0s and 1s.

```
class BinarySplit {
   public static void main(String[] args) {
      int[] arr = {0,1,0,1};
      int total = 0, curr = 0, count = 0;

      for (int x : arr) total += (x == 0 ? -1 : 1);
      for (int x : arr) {
         curr += (x == 0 ? -1 : 1);
         if (curr * 2 == total) count++;
      }
      System.out.println(count);
   }
}
```

**16. Maximum Circular Subarray Sum**

**Description:** Find the maximum sum of a subarray considering circular wrapping.

```
class CircularMax {
    public static void main(String[] args) {
        int[] arr = {5,-2,3,4};
        int total = 0, max = kadane(arr), min = kadaneInvert(arr);

        for (int x : arr) total += x;
        System.out.println(Math.max(max, total - min));
    }

    static int kadane(int[] a) {
        int max = a[0], curr = a[0];
        for (int i = 1; i < a.length; i++) {
            curr = Math.max(a[i], curr + a[i]);
            max = Math.max(max, curr);
        }
        return max;
    }

    static int kadaneInvert(int[] a) {
        int min = a[0], curr = a[0];
        for (int i = 1; i < a.length; i++) {
            curr = Math.min(a[i], curr + a[i]);
            min = Math.min(min, curr);
        }
        return min;
    }
}
```

**17. Partition Array into Increasing Subarrays**

**Description:** Check if array can be split into strictly increasing contiguous subarrays.

```
class IncreasingPartition {
   public static void main(String[] args) {
      int[] arr = {1,2,3,1,2};
      int parts = 1;

      for (int i = 1; i < arr.length; i++)
         if (arr[i] <= arr[i-1]) parts++;

      System.out.println(parts);
   }
}
```

**18. Longest Subarray with Sum ≤ K**

**Description:** Find the longest subarray whose sum does not exceed k.

```
class LongestSumLEK {
   public static void main(String[] args) {
      int[] arr = {1,2,1,0,1};
      int k = 4, sum = 0, left = 0, max = 0;

      for (int right = 0; right < arr.length; right++) {
         sum += arr[right];
         while (sum > k) sum -= arr[left++];
         max = Math.max(max, right - left + 1);
      }
      System.out.println(max);
   }
}
```

**19. Check If String Can Be Rearranged to Palindrome**

**Description:** Determine whether characters of a string can be rearranged to form a palindrome.

```
class PalindromeRearrange {
   public static void main(String[] args) {
      String s = "aabbc";
      int[] freq = new int[26];
      for (char c : s.toCharArray()) freq[c - 'a']++;

      int odd = 0;
      for (int x : freq) if (x % 2 != 0) odd++;
      System.out.println(odd <= 1 ? "Yes" : "No");
   }
}
```

**20. Maximum Sum of Exactly K Non-Overlapping Subarrays of Size M**

**Description:** Find the maximum total sum of k non-overlapping subarrays of size m.

```
class KNonOverlap {
   public static void main(String[] args) {
      int[] arr = {1,2,3,4,5,6};
      int k = 2, m = 2;
      int[] sums = new int[arr.length];

      int window = 0;
      for (int i = 0; i < arr.length; i++) {
         window += arr[i];
         if (i >= m) window -= arr[i-m];
         if (i >= m-1) sums[i] = window;
      }

      int res = 0;
      for (int i = m-1; i < arr.length; i += m)
         res += sums[i];

      System.out.println(res);
   }
}
```

**HARD LEVEL (10 PROGRAMS)**

**21. Maximum Subarray Sum After One Deletion**

**Description:** Find the maximum subarray sum when at most one element can be removed.

```
class MaxOneDeletion {
   public static void main(String[] args) {
      int[] arr = {1,-2,0,3};
      int n = arr.length;
      int[] fw = new int[n], bw = new int[n];

      fw[0] = arr[0];
      bw[n-1] = arr[n-1];

      for (int i = 1; i < n; i++)
         fw[i] = Math.max(arr[i], fw[i-1] + arr[i]);
      for (int i = n-2; i >= 0; i--)
         bw[i] = Math.max(arr[i], bw[i+1] + arr[i]);

      int max = fw[0];
      for (int i = 1; i < n; i++) {
         max = Math.max(max, fw[i]);
         if (i < n-1) max = Math.max(max, fw[i-1] + bw[i+1]);
      }
      System.out.println(max);
   }
}
```

**22. Split Array into Minimum Subarrays with Sum ≥ K**

**Description:** Partition the array into the minimum number of contiguous subarrays each having sum at least k.

```
class MinPartitions {
   public static void main(String[] args) {
      int[] arr = {3,1,2,1,1};
      int k = 3, sum = 0, count = 0;

      for (int x : arr) {
         sum += x;
         if (sum >= k) {
            count++;
            sum = 0;
         }
      }
      System.out.println(count);
   }
}
```

**23. Longest Subarray with Equal 0s, 1s and 2s**

**Description:** Find the longest subarray containing equal counts of 0s, 1s, and 2s.

```
import java.util.HashMap;

class Equal012 {
   public static void main(String[] args) {
      int[] arr = {0,1,0,2,1,2};
      HashMap<String,Integer> map = new HashMap<>();
      map.put("0#0", -1);

      int c0=0,c1=0,c2=0,max=0;
      for(int i=0;i<arr.length;i++){
         if(arr[i]==0)c0++;
         else if(arr[i]==1)c1++;
         else c2++;
         String key=(c1-c0)+"#"+(c2-c1);
         if(map.containsKey(key))
            max=Math.max(max,i-map.get(key));
         else map.put(key,i);
      }
      System.out.println(max);
   }
}
```

**24. Maximum Sum Subarray with Exactly K Distinct Elements**

**Description:** Find the maximum sum of a subarray containing exactly k distinct elements.

```
import java.util.HashMap;

class MaxKDistinctSum {
   public static void main(String[] args) {
      int[] arr = {1,2,1,2,3};
      int k = 2, sum = 0, max = 0, left = 0;
      HashMap<Integer,Integer> map = new HashMap<>();

      for (int right = 0; right < arr.length; right++) {
         map.put(arr[right], map.getOrDefault(arr[right],0)+1);
         sum += arr[right];

         while (map.size() > k) {
            map.put(arr[left], map.get(arr[left]) - 1);
            if (map.get(arr[left]) == 0) map.remove(arr[left]);
            sum -= arr[left++];
         }
         if (map.size() == k) max = Math.max(max, sum);
      }
      System.out.println(max);
   }
}
```

**25. Count Subarrays with Maximum Element Appearing at Least K Times**

**Description:** Count subarrays where the maximum element appears at least k times.

```
class MaxElementKTimes {
   public static void main(String[] args) {
      int[] arr = {1,3,3,3,2};
      int k = 2, max = 3;
      int left = 0, count = 0, freq = 0;

      for (int right = 0; right < arr.length; right++) {
         if (arr[right] == max) freq++;
         while (freq >= k) {
            count += arr.length - right;
            if (arr[left++] == max) freq--;
         }
      }
      System.out.println(count);
   }
}
```

**26. Minimum Window Substring**

**Description:** Find the smallest substring containing all characters of another string.

```
import java.util.HashMap;

class MinWindow {
   public static void main(String[] args) {
      String s="ADOBECODEBANC", t="ABC";
      HashMap<Character,Integer> map=new HashMap<>();
      for(char c:t.toCharArray()) map.put(c,map.getOrDefault(c,0)+1);

      int left=0,count=map.size(),min=Integer.MAX_VALUE,start=0;
      for(int right=0;right<s.length();right++){
         char c=s.charAt(right);
         if(map.containsKey(c)){
            map.put(c,map.get(c)-1);
            if(map.get(c)==0) count--;
         }
         while(count==0){
            if(right-left+1<min){min=right-left+1;start=left;}
            char lc=s.charAt(left++);
            if(map.containsKey(lc)){
               map.put(lc,map.get(lc)+1);
               if(map.get(lc)>0) count++;
            }
         }
      }
      System.out.println(min==Integer.MAX_VALUE?"":s.substring(start,start+min));
   }
}
```

**27. Maximum Sum Subarray with Length in Range [L, R]**

**Description:** Find the maximum subarray sum whose length lies between L and R.

```
class RangeLengthMax {
   public static void main(String[] args) {
      int[] arr = {1,-2,3,4,-1};
      int L = 2, R = 3, max = Integer.MIN_VALUE;

      for (int i = 0; i < arr.length; i++) {
         int sum = 0;
         for (int j = i; j < arr.length && j - i + 1 <= R; j++) {
            sum += arr[j];
            if (j - i + 1 >= L)
               max = Math.max(max, sum);
         }
      }
      System.out.println(max);
   }
}
```

**28. Partition Array into Subarrays with GCD = 1**

**Description:** Count partitions such that each subarray has GCD equal to 1.

```
class GCDPartition {
   static int gcd(int a,int b){return b==0?a:gcd(b,a%b);}
   public static void main(String[] args){
      int[] arr={2,3,4,9};
      int count=0,g=0;
      for(int x:arr){
         g=gcd(g,x);
         if(g==1){count++;g=0;}
      }
      System.out.println(count);
   }
}
```

**29. Longest Anagrammatic Substring**

**Description:** Find the longest substring that is an anagram of another substring.

```
class LongestAnagram {
   public static void main(String[] args) {
      String s="abba";
      int max=0;
      for(int i=0;i<s.length();i++)
         for(int j=i+1;j<s.length();j++)
            if(isAnagram(s.substring(i,j),s.substring(j)))
               max=Math.max(max,j-i);
      System.out.println(max);
   }
   static boolean isAnagram(String a,String b){
      if(a.length()!=b.length())return false;
      int[] f=new int[26];
      for(char c:a.toCharArray())f[c-'a']++;
      for(char c:b.toCharArray())f[c-'a']--;
      for(int x:f)if(x!=0)return false;
      return true;
   }
}
```

**30. Maximum Subarray Sum with Exactly One Peak**

**Description:** Find the maximum subarray sum containing exactly one local maximum.

```
class OnePeakMax {
   public static void main(String[] args) {
      int[] arr={1,3,2,4,1};
      int max=0;
      for(int i=1;i<arr.length-1;i++){
         if(arr[i]>arr[i-1] && arr[i]>arr[i+1]){
            int l=i,r=i,sum=arr[i];
            while(l>0 && arr[l]>arr[l-1]) sum+=arr[--l];
            while(r<arr.length-1 && arr[r]>arr[r+1]) sum+=arr[++r];
            max=Math.max(max,sum);
         }
      }
      System.out.println(max);
   }
}
```