Sliding Window

A sliding window is a computational technique that involves creating a window that slides over data (usually an array or string) to solve problems with minimal time complexity. The window size can be fixed or variable depending on the problem.

Types of Sliding Window:

1. Fixed Size Window
    - Window size remains constant throughout traversal
    - Examples: Finding maximum sum subarray of size k, finding first negative in every window of size k
2. Variable Size Window
    - Window size can change during traversal
    - Used when we need to find longest/shortest subarray satisfying a condition
    - Examples: Longest subarray with sum k, smallest subarray with sum greater than k

The Sliding Window technique optimizes problems by eliminating redundant calculations, reducing time complexity from $O(n^2)$ (brute force) to O(n)in most cases.

*Find max sum subarray of size k // Time Complexity: O(n*k)*

```
// C++ Complete Implementation
#include <iostream>
#include <vector>
#include <climits>
using namespace std;



// C++ Brute Force Solutions
// 1. Fixed Size Window - Find max sum subarray of size k
// Time Complexity: O(n*k)
int maxSumFixedWindowBrute(vector<int>& arr, int k) {
    int n = arr.size();
    if (n < k) return -1;


    int maxSum = INT_MIN;

    // Consider all possible windows of size k
    for (int i = 0; i <= n - k; i++) {
        int currentSum = 0;



        // Calculate sum of current window
        for (int j = i; j < i + k; j++) {
            currentSum += arr[j];
        }

        maxSum = max(maxSum, currentSum);
    }
```

```
    return maxSum;
}


int maxSumFixedWindow(vector<int>& arr, int k) {
    int n = arr.size();
    if (n < k) return -1;

    // First window sum
    int windowSum = 0;
    for (int i = 0; i < k; i++) {
        windowSum += arr[i];
    }

    int maxSum = windowSum;

    // Slide window and update max
    for (int i = k; i < n; i++) {
        windowSum = windowSum + arr[i] - arr[i - k];
        maxSum = max(maxSum, windowSum);
    }

    return maxSum;
}

int main() {
    // Test cases for Fixed Size Window
    cout << "=== Fixed Size Window Tests ===" << endl;


    // Test Case 1: Basic array
    vector<int> arr1 = {1, 4, 2, 10, 23, 3, 1, 0, 20};
    int k1 = 4;
    cout << "\nTest 1:" << endl;
    cout << "Array: ";
    for(int x : arr1) cout << x << " ";
    cout << "\nk = " << k1 << endl;


    cout << "Maximum sum (Sliding Window): " << maxSumFixedWindow(arr1, k1) << endl;
    cout << "Maximum sum (Brute Force): " << maxSumFixedWindowBrute(arr1, k1) << endl;

    // Test Case 2: All same numbers
    vector<int> arr2 = {1, 1, 1, 1, 1};
    int k2 = 2;
    cout << "\nTest 2:" << endl;
    cout << "Array: ";
    for(int x : arr2) cout << x << " ";
    cout << "\nk = " << k2 << endl;

    cout << "Maximum sum (Sliding Window): " << maxSumFixedWindow(arr2, k2) << endl;
    cout << "Maximum sum (Brute Force): " << maxSumFixedWindowBrute(arr2, k2) << endl;


  return 0;
}
```

```
import java.util.*;


class SlidingWindowBrute {
   // 1. Fixed Size Window - Find max sum subarray of size k
   // Time Complexity: O(n*k)
   public static int maxSumFixedWindowBrute(int[] arr, int k) {
      int n = arr.length;
      if (n < k) return -1;

      int maxSum = Integer.MIN_VALUE;

      // Consider all possible windows of size k
      for (int i = 0; i <= n - k; i++) {
         int currentSum = 0;

         // Calculate sum of current window
         for (int j = i; j < i + k; j++) {
            currentSum += arr[j];
         }

         maxSum = Math.max(maxSum, currentSum);
      }

      return maxSum;
   }
    public static int maxSumFixedWindow(int[] arr, int k) {
      int n = arr.length;
      if (n < k) return -1;

      // First window sum
      int windowSum = 0;
      for (int i = 0; i < k; i++) {
         windowSum += arr[i];
      }

      int maxSum = windowSum;

      // Slide window and update max
      for (int i = k; i < n; i++) {
         windowSum = windowSum + arr[i] - arr[i - k];
         maxSum = Math.max(maxSum, windowSum);
      }

      return maxSum;
   }
    public static void main(String[] args) {
      // Test cases for Fixed Size Window
      System.out.println("=== Fixed Size Window Tests ===");

      // Test Case 1: Basic array
      int[] arr1 = {1, 4, 2, 10, 23, 3, 1, 0, 20};
      int k1 = 4;
      System.out.println("\nTest 1:");
      System.out.print("Array: ");
      for(int x : arr1) System.out.print(x + " ");
      System.out.println("\nk = " + k1);
```

```
        System.out.println("Maximum sum (Sliding Window): " + maxSumFixedWindow(arr1, k1));
        System.out.println("Maximum sum (Brute Force): " + maxSumFixedWindowBrute(arr1, k1));

        // Test Case 2: All same numbers
        int[] arr2 = {1, 1, 1, 1, 1};
        int k2 = 2;
        System.out.println("\nTest 2:");
        System.out.print("Array: ");
        for(int x : arr2) System.out.print(x + " ");
        System.out.println("\nk = " + k2);

        System.out.println("Maximum sum (Sliding Window): " + maxSumFixedWindow(arr2, k2));
        System.out.println("Maximum sum (Brute Force): " + maxSumFixedWindowBrute(arr2, k2));
    }
}
```

```
=== Fixed Size Window Tests ===

Test 1:
Array: 1 4 2 10 23 3 1 0 20
k = 4
Maximum sum (Sliding Window): 39
Maximum sum (Brute Force): 39

Test 2:
Array: 1 1 1 1 1
k = 2
Maximum sum (Sliding Window): 2
Maximum sum (Brute Force): 2
```

Q Maximum Sum of Subarray of Size K
Problem: Given an array of integers and a number k, find the maximum sum of any contiguous subarray of size k.
Input:
9
1 3 2 6 -1 4 1 8 2
3 //k
Output:
14
Explanation (subarray [4, 1, 8])

Input
6
2 1 5 1 3 2
3 //k
Output: 9
Explanation: Maximum sum subarray is [5, 1, 3].

Input:
9
4 2 1 7 8 1 2 8 10
4 //k
Output: 21
Explanation: Maximum sum subarray is [1, 2, 8, 10].

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Function to find the maximum sum of any contiguous subarray of size k
int maxSumSubarray(const vector<int>& arr, int k) {
    int maxSum = INT_MIN, currentSum = 0;

    // Calculate the sum of the first 'k' elements
    for (int i = 0; i < k; i++) {
        currentSum += arr[i];
    }
    maxSum = currentSum;

    // Slide the window
    for (int i = k; i < arr.size(); i++) {
        currentSum += arr[i] - arr[i - k];
        maxSum = max(maxSum, currentSum);
    }

    return maxSum;
}

int main() {
    int n, k;
    //cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> arr(n);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    //cout << "Enter the value of k: ";
    cin >> k;


    int result = maxSumSubarray(arr, k);
    cout << result << endl;

    return 0;
}

Java
import java.util.Scanner;

class MaxSumSubarray {
    // Function to find the maximum sum of any contiguous subarray of size k
    public static int maxSumSubarray(int[] arr, int k) {
        int maxSum = Integer.MIN_VALUE, currentSum = 0;

        // Calculate the sum of the first 'k' elements
        for (int i = 0; i < k; i++) {
            currentSum += arr[i];
        }
        maxSum = currentSum;
```

```
        // Slide the window
        for (int i = k; i < arr.length; i++) {
            currentSum += arr[i] - arr[i - k];
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Input the size of the array
        int n = sc.nextInt();
        int[] arr = new int[n];

        // Input the elements of the array
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        // Input the value of k
        int k = sc.nextInt();

        // Find and print the result
        int result = maxSumSubarray(arr, k);
        System.out.println(result);

        sc.close();
    }
}
```

643. Maximum Average Subarray I

```
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
    int n = nums.size();
    double maxSum = 0, currentSum = 0;

    // Calculate the sum of the first 'k' elements
    for (int i = 0; i < k; i++) {
        currentSum += nums[i];
    }
    maxSum = currentSum;

    // Slide the window
    for (int i = k; i < n; i++) {
        currentSum += nums[i] - nums[i - k];
        maxSum = max(maxSum, currentSum);
    }

    // Return the maximum average
    return maxSum / k;
    }
};
```

Java

```
class Solution {
    public double findMaxAverage(int[] nums, int k) {
     int n = nums.length;
        double maxSum = 0, currentSum = 0;

        // Calculate the sum of the first 'k' elements
        for (int i = 0; i < k; i++) {
            currentSum += nums[i];
        }
        maxSum = currentSum;

        // Slide the window
        for (int i = k; i < n; i++) {
            currentSum += nums[i] - nums[i - k];
            maxSum = Math.max(maxSum, currentSum);
        }

        // Return the maximum average
        return maxSum / k;
    }
}
```

1176. Diet Plan Performance(P)

```
class Solution {
public:
    int dietPlanPerformance(vector<int>& calories, int k, int lower, int upper) {
    int n = calories.size();
        int s[n + 1];
        s[0] = 0;
        for (int i = 0; i < n; ++i) {
            s[i + 1] = s[i] + calories[i];
        }
        int ans = 0;
        for (int i = 0; i < n - k + 1; ++i) {
            int t = s[i + k] - s[i];
            if (t < lower) {
                --ans;
            } else if (t > upper) {
                ++ans;
            }
        }
        return ans;
    }
};
```

Java

```
lass Solution {
    public int dietPlanPerformance(int[] calories, int k, int lower, int upper) {
    int n = calories.length;
        int[] s = new int[n + 1];
        for (int i = 0; i < n; ++i) {
            s[i + 1] = s[i] + calories[i];
        }
        int ans = 0;
```

```
        for (int i = 0; i < n - k + 1; ++i) {
            int t = s[i + k] - s[i];
            if (t < lower) {
                --ans;
            } else if (t > upper) {
                ++ans;
            }
        }
        return ans;
    }
}
```

1652. Defuse the Bomb

```
class Solution {
public:
    vector<int> decrypt(vector<int>& code, int k) {
        int n = code.size();
        vector<int> ans(n);
        if (k == 0) {
            return ans;
        }
        for (int i = 0; i < n; ++i) {
            if (k > 0) {
                for (int j = i + 1; j < i + k + 1; ++j) {
                    ans[i] += code[j % n];
                }
            } else {
                for (int j = i + k; j < i; ++j) {
                    ans[i] += code[(j + n) % n];
                }
            }
        }
        return ans;
    }
};
```

Java

```
class Solution {
    public int[] decrypt(int[] code, int k) {
        int n = code.length;
        int[] ans = new int[n];
        if (k == 0) {
            return ans;
        }
        for (int i = 0; i < n; ++i) {
            if (k > 0) {
                for (int j = i + 1; j < i + k + 1; ++j) {
                    ans[i] += code[j % n];
                }
            } else {
                for (int j = i + k; j < i; ++j) {
                    ans[i] += code[(j + n) % n];
                }
            }
        }
        return ans;
    }
```

}

3. Minimum Sum of Subarray of Size K
Problem: Given an array of integers and a number k, find the minimum sum of any contiguous subarray of size k.
Input:
5
4 2 3 1 6
2 //k
Output:
3 (subarray [2, 1])

Input
6
2 1 5 1 3 2
3 //k
Output:
6
Explanation: The subarray [1, 3, 2] has the minimum sum 6.

Input
10
4 2 1 7 8 1 2 8 1 0
4 //k
Output:
11
Explanation: The subarray [2, 8, 1, 0] has the minimum sum, which is 11

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int findMinSumSubarray(const vector<int>& arr, int k) {
   int n = arr.size();

   int minSum = INT_MAX, windowSum = 0;

   // Calculate the sum of the first window
   for (int i = 0; i < k; ++i) {
      windowSum += arr[i];
   }

   minSum = windowSum;

   // Slide the window
   for (int i = k; i < n; ++i) {
      windowSum += arr[i] - arr[i - k];
      minSum = min(minSum, windowSum);
   }

   return minSum;
}

int main() {
   int n, k;
   //cout << "Enter the size of the array: ";
```

```
    cin >> n;

    vector<int> arr(n);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    //cout << "Enter the value of k: ";
    cin >> k;

    int result = findMinSumSubarray(arr, k);
    if (result != -1) {
        cout << result << endl;
    }

    return 0;
}

import java.util.Scanner;

class MinSumSubarray {
    public static int findMinSumSubarray(int[] arr, int k) {
        int n = arr.length;


        int minSum = Integer.MAX_VALUE, windowSum = 0;

        // Calculate the sum of the first window
        for (int i = 0; i < k; i++) {
            windowSum += arr[i];
        }

        minSum = windowSum;

        // Slide the window
        for (int i = k; i < n; i++) {
            windowSum += arr[i] - arr[i - k];
            minSum = Math.min(minSum, windowSum);
        }

        return minSum;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        //System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];
        //System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        //System.out.print("Enter the value of k: ");
```

```
        int k = scanner.nextInt();

        int result = findMinSumSubarray(arr, k);
        if (result != -1) {
            System.out.println(result);
        }

        scanner.close();
    }
}
```

4. Product of Subarrays of Size K
Problem: Find the product of all contiguous subarrays of size k.
Input:
4
2 3 4 5
2 //k
Output:
6 12 20

Explanation: The subarrays of size 2 are [2, 3], [3, 4], and [4, 5].
Their products are 2×3=6 3×4=12 and 4×5=20.

Input
5
1 2 3 4 5
3 //k
Output
6 24 60
Explanation: The subarrays of size 3 are [1, 2, 3], [2, 3, 4], and [3, 4, 5]. Their products are 1×2×3=6 2×3×4=24 and 3×4×5=60.

Input
4
10 5 2 6
2 //k
Output:
50 10 12

```
#include <iostream>
#include <vector>
using namespace std;

// Function to find the product of all subarrays of size k
vector<int> findSubarrayProducts(const vector<int>& arr, int k) {
    int n = arr.size();
    vector<int> result;

   int product = 1;

    // Compute the product for the first window
    for (int i = 0; i < k; ++i) {
        product *= arr[i];
    }
    result.push_back(product);
```

```
    // Slide the window
    for (int i = k; i < n; ++i) {
        product = product / arr[i - k] * arr[i];
        result.push_back(product);
    }

    return result;
}

int main() {
    int n, k;
    //cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    //cout << "Enter the value of k: ";
    cin >> k;

    vector<int> result = findSubarrayProducts(arr, k);

   for (int prod : result) {
          cout << prod << " ";
    }
        cout << endl;
    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class SubarrayProducts {

    // Function to find the product of all subarrays of size k
    public static List<Integer> findSubarrayProducts(int[] arr, int k) {
        int n = arr.length;
        List<Integer> result = new ArrayList<>();

        int product = 1;

        // Compute the product for the first window
        for (int i = 0; i < k; i++) {
            product *= arr[i];
        }
        result.add(product);

        // Slide the window
        for (int i = k; i < n; i++) {
            product = product / arr[i - k] * arr[i];
            result.add(product);
        }
```

```
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        //System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        // Input the elements of the array
        //System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input the value of k
       // System.out.print("Enter the value of k: ");
        int k = scanner.nextInt();

        // Find the products of subarrays of size k
        List<Integer> result = findSubarrayProducts(arr, k);

        // Output the result
        //System.out.println("Products of subarrays of size " + k + ": ");
        for (int prod : result) {
            System.out.print(prod + " ");
        }
        System.out.println();

        scanner.close();
    }
}
```

5. Count Subarrays of Size K with Even Sums
Problem: Count how many contiguous subarrays of size k have an even sum.
Input:
5
1 2 3 4 5
k = 2
Output:
0
Explanation
[1, 2] → Sum = 3 (odd)
[2, 3] → Sum = 5 (odd)
[3, 4] → Sum = 7 (odd)
[4, 5] → Sum = 9 (odd)

Input
4
2 4 6 8
2//k
Output
3

Explanation
2 4 Sum 6
4 6 Sum 10
6 8 Sum 14

Input
5
1 1 1 1 1
3
Output
0

```
#include <iostream>
#include <vector>
using namespace std;

// Function to count subarrays of size k with even sums
int countEvenSumSubarrays(const vector<int>& arr, int k) {
    int n = arr.size();
    int count = 0;
    int sum = 0;

    // Calculate the sum for the first window
    for (int i = 0; i < k; ++i) {
        sum += arr[i];
    }

    // Check if the first window sum is even
    if (sum % 2 == 0) {
        ++count;
    }

    // Slide the window
    for (int i = k; i < n; ++i) {
        sum += arr[i] - arr[i - k];
        if (sum % 2 == 0) {
            ++count;
        }
    }

    return count;
}

int main() {
    int n, k;

    //cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    //cout << "Enter the value of k: ";
```

```
    cin >> k;

    int result = countEvenSumSubarrays(arr, k);
    cout << result << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

public class EvenSumSubarrays {

    // Function to count subarrays of size k with even sums
    public static int countEvenSumSubarrays(int[] arr, int k) {
        int n = arr.length;
        int count = 0;
        int sum = 0;

        // Calculate the sum for the first window
        for (int i = 0; i < k; i++) {
            sum += arr[i];
        }

        // Check if the first window sum is even
        if (sum % 2 == 0) {
            count++;
        }

        // Slide the window
        for (int i = k; i < n; i++) {
            sum += arr[i] - arr[i - k]; // Update the window sum
            if (sum % 2 == 0) {
                count++;
            }
        }

        return count;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input array size
        System.out.print("Enter the size of the array: ");
        int n = scanner.nextInt();

        int[] arr = new int[n];

        // Input array elements
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input k
        System.out.print("Enter the value of k: ");
```

```
        int k = scanner.nextInt();

        // Call the function and display the result
        int result = countEvenSumSubarrays(arr, k);
        System.out.println("Count of subarrays of size " + k + " with even sums: " + result);

        scanner.close();
    }
}
```

6. Largest Sum of Subarray of Size K with Positive Numbers Only
Problem: Given an array, find the largest sum of a subarray of size k that contains only positive numbers.
Input:
7
3 -1 2 5 -2 7 1
2 //k
Output:
8 (subarray [7, 1])
Explanation
Subarray [2, 5]
Sum = 2+5=7 All positive → Valid.

Subarray [7, 1]
Sum = 7+1=8 All positive → Valid.

Input
6
1 2 -3 4 5 6
3 //k
Output
15
Explanation
Only [4, 5, 6] (sum = 15) is valid. The largest sum is 15

Input
7
10 -2 -1 3 4 -5 6
2 //k
Output
7
Explanation Only [3, 4] (sum = 7) is valid. The largest sum is 7.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Function to find the largest sum of subarray of size k with positive numbers only
int largestSumPositiveSubarray(const vector<int>& arr, int k) {
    int n = arr.size();
    int maxSum = INT_MIN;

    // Iterate over all possible subarrays of size k
    for (int i = 0; i <= n - k; ++i) {
        bool allPositive = true;
        int currentSum = 0;

        // Check if the subarray contains only positive numbers
```

```
        for (int j = i; j < i + k; ++j) {
            if (arr[j] <= 0) {
                allPositive = false;
                break;
            }
            currentSum += arr[j];
        }

        // Update maxSum if the subarray is valid
        if (allPositive) {
            maxSum = max(maxSum, currentSum);
        }
    }

    return maxSum == INT_MIN ? 0 : maxSum; // Return 0 if no valid subarray exists
}

int main() {
    int n, k;

    // Input size of the array
    //cout << "Enter the size of the array: ";
    cin >> n;

    vector<int> arr(n);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    // Input size of the subarray
    //cout << "Enter the value of k: ";
    cin >> k;

    // Find the largest sum of subarray of size k with positive numbers only
    int result = largestSumPositiveSubarray(arr, k);
    cout <<result << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;
class LargestSumPositiveSubarray {

    // Function to find the largest sum of subarray of size k with positive numbers only
    public static int largestSumPositiveSubarray(int[] arr, int k) {
        int n = arr.length;
        int maxSum = Integer.MIN_VALUE;

        // Iterate over all possible subarrays of size k
        for (int i = 0; i <= n - k; i++) {
            boolean allPositive = true;
            int currentSum = 0;

            // Check if the subarray contains only positive numbers
            for (int j = i; j < i + k; j++) {
```

```
            if (arr[j] <= 0) {
                allPositive = false;
                break;
            }
            currentSum += arr[j];
        }

        // Update maxSum if the subarray is valid
        if (allPositive) {
            maxSum = Math.max(maxSum, currentSum);
        }
    }

    return maxSum == Integer.MIN_VALUE ? 0 : maxSum; // Return 0 if no valid subarray exists
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Input the size of the array
    //System.out.print("Enter the size of the array: ");
    int n = scanner.nextInt();

    int[] arr = new int[n];

    // Input the elements of the array
    //System.out.println("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }

    // Input the size of the subarray
    //System.out.print("Enter the value of k: ");
    int k = scanner.nextInt();

    // Find the largest sum of subarray of size k with positive numbers only
    int result = largestSumPositiveSubarray(arr, k);

    // Print the result
    System.out.println(result);

    scanner.close();
}
}
```

7. Find the Subarray with Maximum Sum of Size K and Print It
Problem: Print the subarray of size k with the maximum sum.
Input:
6
2 1 5 1 3 2
3 //k
Output:
5 1 3

Input
7
1 9 2 4 7 3 6

2 //k
Output
9 2

Input
8
-1 4 -2 6 3 -5 8 2
4 //k
Output
6 3 -5 8

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Function to find the subarray of size k with the maximum sum
vector<int> maxSumSubarray(const vector<int>& arr, int k) {
    int n = arr.size();
    int maxSum = INT_MIN, currentSum = 0;
    int startIndex = 0;

    // Calculate the sum of the first window
    for (int i = 0; i < k; ++i) {
        currentSum += arr[i];
    }
    maxSum = currentSum;

    // Slide the window
    for (int i = k; i < n; ++i) {
        currentSum += arr[i] - arr[i - k];
        if (currentSum > maxSum) {
            maxSum = currentSum;
            startIndex = i - k + 1;
        }
    }

    return vector<int>(arr.begin() + startIndex, arr.begin() + startIndex + k);
}

int main() {
    int n, k;
    cin >> n;

    vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cin >> k;

    vector<int> result = maxSumSubarray(arr, k);
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;
```

```
    return 0;
}
```

Java

```
import java.util.*;
import java.lang.*;
import java.io.*;

class MaxSumSubarray {

    // Function to find the subarray of size k with the maximum sum
    public static int[] maxSumSubarray(int[] arr, int k) {
        int n = arr.length;
        int maxSum = Integer.MIN_VALUE, currentSum = 0;
        int startIndex = 0;

        // Calculate the sum of the first window
        for (int i = 0; i < k; i++) {
            currentSum += arr[i];
        }
        maxSum = currentSum;

        // Slide the window
        for (int i = k; i < n; i++) {
            currentSum += arr[i] - arr[i - k];
            if (currentSum > maxSum) {
                maxSum = currentSum;
                startIndex = i - k + 1;
            }
        }

        // Return the subarray
        int[] result = new int[k];
        for (int i = 0; i < k; i++) {
            result[i] = arr[startIndex + i];
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        int k = scanner.nextInt();

        int[] result = maxSumSubarray(arr, k);
        for (int num : result) {
            System.out.print(num + " ");
        }
        System.out.println();

        scanner.close();
```

```
    }
}
```

8. Count Subarrays of Size K with Distinct Elements
Problem: Count all subarrays of size k that have distinct elements (no duplicates).
Input:
6
1 2 3 4 5 6
3
Output:
4
Explanation
The subarrays [1, 2, 3], [2, 3, 4], [3, 4, 5], and [4, 5, 6] all have distinct elements.

Input
7
1 2 2 3 4 5 6
3
Output
3
Explanation: The subarrays [2, 3, 4], [3, 4, 5], and [4, 5, 6] have distinct elements.

Input
5
1 1 1 1 1
2
Output
0
Explanation: No subarrays of size 2 have distinct elements.

```
#include <iostream>
#include <vector>
using namespace std;

// Function to count subarrays of size k with distinct elements
int countDistinctSubarrays(const vector<int>& arr, int k) {
    int n = arr.size();
    int count = 0;

    for (int i = 0; i <= n - k; ++i) {
        bool hasDuplicate = false;

        // Check for duplicates in the subarray
        for (int j = i; j < i + k; ++j) {
            for (int l = j + 1; l < i + k; ++l) {
                if (arr[j] == arr[l]) {
                    hasDuplicate = true;
                    break;
                }
            }
            if (hasDuplicate) break;
        }

        // Increment count if no duplicates were found
        if (!hasDuplicate) {
            ++count;
        }
    }
```

```
    return count;
}

int main() {
    int n, k;
    cin >> n;

    vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cin >> k;

    cout << countDistinctSubarrays(arr, k) << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

class DistinctSubarrays {

    // Function to count subarrays of size k with distinct elements
    public static int countDistinctSubarrays(int[] arr, int k) {
        int n = arr.length;
        int count = 0;

        for (int i = 0; i <= n - k; i++) {
            boolean hasDuplicate = false;

            // Check for duplicates in the subarray
            for (int j = i; j < i + k; j++) {
                for (int l = j + 1; l < i + k; l++) {
                    if (arr[j] == arr[l]) {
                        hasDuplicate = true;
                        break;
                    }
                }
                if (hasDuplicate) break;
            }

            // Increment count if no duplicates were found
            if (!hasDuplicate) {
                count++;
            }
        }

        return count;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input array size
```

```
        int n = scanner.nextInt();
        int[] arr = new int[n];

        // Input array elements
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input k
        int k = scanner.nextInt();

        // Output the count of distinct subarrays of size k
        System.out.println(countDistinctSubarrays(arr, k));

        scanner.close();
    }
}
```

Variable Size window Dynamic Window
Problem: Given an array, find the subarray of size k with the maximum number of odd numbers.
Input
6
1 2 3 4 5 6
3 //k
Output
1 2 3
Explanation:
The subarray [1, 3, 1] has a sum ≤5.

Input
7
2 4 6 8 1 3 5
2 //k
Output
1 3

Input
5
1 1 1 1 2
4 //k
Output
1 1 1 1

```
#include <iostream>
#include <vector>
using namespace std;

// Function to find the subarray of size k with the maximum number of odd numbers
vector<int> maxOddNumbersSubarray(const vector<int>& arr, int k) {
    int n = arr.size();
    int maxOddCount = 0, currentOddCount = 0;
    int startIndex = 0;

    // Count the number of odd numbers in the first window
```

```
    for (int i = 0; i < k; ++i) {
        if (arr[i] % 2 != 0) {
            currentOddCount++;
        }
    }
    maxOddCount = currentOddCount;

    // Slide the window
    for (int i = k; i < n; ++i) {
        // Remove the effect of the element going out of the window
        if (arr[i - k] % 2 != 0) {
            currentOddCount--;
        }
        // Add the effect of the element coming into the window
        if (arr[i] % 2 != 0) {
            currentOddCount++;
        }

        // Update the maxOddCount and startIndex if needed
        if (currentOddCount > maxOddCount) {
            maxOddCount = currentOddCount;
            startIndex = i - k + 1;
        }
    }

    // Return the subarray with the maximum number of odd numbers
    return vector<int>(arr.begin() + startIndex, arr.begin() + startIndex + k);
}

int main() {
    int n, k;
    cin >> n;

    vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cin >> k;

    vector<int> result = maxOddNumbersSubarray(arr, k);
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}

Java
import java.util.Scanner;

class MaxOddNumbersSubarray {

    // Function to find the subarray of size k with the maximum number of odd numbers
    public static int[] maxOddNumbersSubarray(int[] arr, int k) {
        int n = arr.length;
```

```
    int maxOddCount = 0, currentOddCount = 0;
    int startIndex = 0;

    // Count the number of odd numbers in the first window
    for (int i = 0; i < k; i++) {
        if (arr[i] % 2 != 0) {
            currentOddCount++;
        }
    }
    maxOddCount = currentOddCount;

    // Slide the window
    for (int i = k; i < n; i++) {
        // Remove the effect of the element going out of the window
        if (arr[i - k] % 2 != 0) {
            currentOddCount--;
        }
        // Add the effect of the element coming into the window
        if (arr[i] % 2 != 0) {
            currentOddCount++;
        }

        // Update the maxOddCount and startIndex if needed
        if (currentOddCount > maxOddCount) {
            maxOddCount = currentOddCount;
            startIndex = i - k + 1;
        }
    }

    // Create the resulting subarray
    int[] result = new int[k];
    for (int i = 0; i < k; i++) {
        result[i] = arr[startIndex + i];
    }
    return result;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Input array size
    int n = scanner.nextInt();
    int[] arr = new int[n];

    // Input array elements
    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }

    // Input k
    int k = scanner.nextInt();

    // Find the subarray with the maximum number of odd numbers
    int[] result = maxOddNumbersSubarray(arr, k);

    // Print the result
    for (int num : result) {
```

```
            System.out.print(num + " ");
        }
        System.out.println();

        scanner.close();
    }
}
```

QSmallest Subarray with Sum Greater Than K

Problem:

Given an array of integers and a number k, find the length of the smallest subarray whose sum is greater than k.

Input
6
2 3 1 2 4 3
7 //k

Output
3
Explanation subarray 2 4 3

Input
7
1 1 1 1 1 1 8
4//k
Output
1
Explanation subarray 8

Input
6
1 4 45 6 0 19
51 //k
Output
3
Explanation subarray 4 45 6

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Function to find the length of the smallest subarray with a sum greater than k
int smallestSubarrayWithSum(const vector<int>& arr, int k) {
    int n = arr.size();
    int minLength = INT_MAX;
    int currentSum = 0;
    int start = 0;

    for (int end = 0; end < n; ++end) {
        currentSum += arr[end];

        // Shrink the window as long as the sum is greater than k
```

```
        while (currentSum > k) {
            minLength = min(minLength, end - start + 1);
            currentSum -= arr[start];
            start++;
        }
    }

    return minLength == INT_MAX ? 0 : minLength;
}

int main() {
    int n, k;
    cin >> n;

    vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cin >> k;

    int result = smallestSubarrayWithSum(arr, k);
    cout << result << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

class SmallestSubarrayWithSum {

    // Function to find the length of the smallest subarray with a sum greater than k
    public static int smallestSubarrayWithSum(int[] arr, int k) {
        int n = arr.length;
        int minLength = Integer.MAX_VALUE;
        int currentSum = 0;
        int start = 0;

        for (int end = 0; end < n; end++) {
            currentSum += arr[end];

            // Shrink the window as long as the sum is greater than k
            while (currentSum > k) {
                minLength = Math.min(minLength, end - start + 1);
                currentSum -= arr[start];
                start++;
            }
        }

        return minLength == Integer.MAX_VALUE ? 0 : minLength;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
```

```
        int n = scanner.nextInt();
        int[] arr = new int[n];

        // Input the elements of the array
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input the value of k
        int k = scanner.nextInt();

        // Find and print the result
        int result = smallestSubarrayWithSum(arr, k);
        System.out.println(result);

        scanner.close();
    }
}
```

Modified Question
QSmallest Subarray with Sum Greater Than K
Problem:
Given an array of integers and a number k, return the smallest subarray whose sum is greater than k.

Input
6
2 3 1 2 4 3
7 //k

Output
2 4 3

Input
7
1 1 1 1 1 1 8
4//k
Output
8

Input
6
1 4 45 6 0 19
51 //k
Output
4 45 6

```
#include <iostream>
#include <vector>
#include <climits>
```

```
using namespace std;

// Function to find the smallest subarray with a sum greater than k
vector<int> smallestSubarrayWithSum(const vector<int>& arr, int k) {
   int n = arr.size();
   int minLength = INT_MAX;
   int currentSum = 0;
   int start = 0;
   vector<int> result;

   for (int end = 0; end < n; ++end) {
      currentSum += arr[end];

      // Shrink the window as long as the sum is greater than k
      while (currentSum > k) {
         // Update the result if a smaller subarray is found
         if (end - start + 1 < minLength) {
            minLength = end - start + 1;
            result.assign(arr.begin() + start, arr.begin() + end + 1);
         }
         currentSum -= arr[start];
         start++;
      }
   }

   return result; // Return the subarray
}

int main() {
   int n, k;
   cin >> n;

   vector<int> arr(n);
   for (int i = 0; i < n; ++i) {
      cin >> arr[i];
   }

   cin >> k;

   vector<int> result = smallestSubarrayWithSum(arr, k);

     for (int num : result) {
         cout << num << " ";
      }
      cout << endl;
   return 0;
}


Java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class SmallestSubarrayWithSum {

   // Function to find the smallest subarray with a sum greater than k
```

```
    public static List<Integer> smallestSubarrayWithSum(int[] arr, int k) {
        int n = arr.length;
        int minLength = Integer.MAX_VALUE;
        int currentSum = 0;
        int start = 0;
        List<Integer> result = new ArrayList<>();

        for (int end = 0; end < n; end++) {
            currentSum += arr[end];

            // Shrink the window as long as the sum is greater than k
            while (currentSum > k) {
                // Update the result if a smaller subarray is found
                if (end - start + 1 < minLength) {
                    minLength = end - start + 1;
                    result = new ArrayList<>();
                    for (int i = start; i <= end; i++) {
                        result.add(arr[i]);
                    }
                }
                currentSum -= arr[start];
                start++;
            }
        }

        return result; // Return the subarray
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        int n = scanner.nextInt();
        int[] arr = new int[n];

        // Input the elements of the array
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Input the value of k
        int k = scanner.nextInt();

        // Find and print the result
        List<Integer> result = smallestSubarrayWithSum(arr, k);

        System.out.println(result);

        scanner.close();
    }
}
```

Q**Longest Subarray with Even Numbers**

Problem:

Given an array of integers, find the longest subarray that contains only even numbers.

Input:
7
1 2 4 6 3 8 10
Output:
2 4 6
Explanation:
The subarray [2, 4, 6] contains only even numbers, and its length is 3.

Input
4
7 3 5 11
Output
No even subarray

Input
7
2 4 6 8 10 1 12
Output
2 4 6 8 10

```
#include <iostream>
#include <vector>
using namespace std;

// Function to find the longest subarray with only even numbers
vector<int> longestEvenSubarray(const vector<int>& arr) {
   int n = arr.size();
   int maxLength = 0, currentLength = 0, start = 0, maxStart = -1;

   for (int i = 0; i < n; ++i) {
      if (arr[i] % 2 == 0) {
         currentLength++;
         if (currentLength > maxLength) {
            maxLength = currentLength;
            maxStart = i - maxLength + 1;
         }
      } else {
         currentLength = 0;
      }
   }

   if (maxStart == -1) {
      return {}; // No even subarray found
   }

   return vector<int>(arr.begin() + maxStart, arr.begin() + maxStart + maxLength);
}

int main() {
   int n;
   cin >> n;

   vector<int> arr(n);
   for (int i = 0; i < n; ++i) {
      cin >> arr[i];
```

```
    }

    vector<int> result = longestEvenSubarray(arr);

    if (!result.empty()) {
        for (int num : result) {
            cout << num << " ";
        }
        cout << endl;
    } else {
        cout << "No even subarray found" << endl;
    }

    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class LongestEvenSubarray {

    // Function to find the longest subarray with only even numbers
    public static List<Integer> longestEvenSubarray(int[] arr) {
        int n = arr.length;
        int maxLength = 0, currentLength = 0, start = 0, maxStart = -1;

        for (int i = 0; i < n; i++) {
            if (arr[i] % 2 == 0) {
                currentLength++;
                if (currentLength > maxLength) {
                    maxLength = currentLength;
                    maxStart = i - maxLength + 1;
                }
            } else {
                currentLength = 0;
            }
        }

        if (maxStart == -1) {
            return new ArrayList<>(); // No even subarray found
        }

        List<Integer> result = new ArrayList<>();
        for (int i = maxStart; i < maxStart + maxLength; i++) {
            result.add(arr[i]);
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        int n = scanner.nextInt();
        int[] arr = new int[n];
```

```
        // Input the elements of the array
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Find and print the result
        List<Integer> result = longestEvenSubarray(arr);

        if (!result.isEmpty()) {
            for (int num : result) {
                System.out.print(num + " ");
            }
            System.out.println();
        } else {
            System.out.println("No even subarray");
        }

        scanner.close();
    }
}
```

4. Maximum Sum of Subarray with at Most K Distinct Integers
Problem:
Given an array of integers and a number k, find the maximum sum of a subarray containing at most k distinct integers.
Input:
5
4 2 2 1 2
2 //k
Output:
9
Explanation:
The subarray [2, 2, 1, 2] has 2 distinct integers and a sum of 9.

---

5. Longest Subarray with Sum Divisible by K
Problem:
Given an array of integers and a number k, find the length of the longest subarray whose sum is divisible by k.
Input:
arr = [2, 3, 4, 6]
k = 5
Output:
3
Explanation:
The subarray [3, 4, 6] has a sum 3+4+6=133 + 4 + 6 = 133+4+6=13, and 13%5=013 \% 5 = 013%5=0.

---

6. Count Subarrays with Sum Exactly Equal to K
Problem:
Given an array of integers and a number kkk, count the number of subarrays with a sum equal to kkk.
Example Input:
makefile
Copy code
arr = [1, 2, 3]
k = 3
Output:
Copy code

2
Explanation:
The subarrays [1, 2] and [3] have sums equal to k=3k = 3k=3.

---

7. Maximum Product of a Subarray
Problem:
Given an array of positive integers, find the maximum product of any subarray.
Example Input:
css
Copy code
arr = [1, 2, 3, 4]
Output:
Copy code
24
Explanation:
The subarray [1, 2, 3, 4] has a product of 24, which is the maximum.