PROBLEMS OF THE DAY – 3

1.K Sized Subarray Maximum

Given an array arr[] and an integer k. Find the maximum for each and every contiguous subarray of size k.

Examples:

Input: k = 3, arr[] = [1, 2, 3, 1, 4, 5, 2, 3, 6]

Output: [3, 3, 4, 5, 5, 5, 6]

Explanation:

1st contiguous subarray = [1 2 3] max = 3

2nd contiguous subarray = [2 3 1] max = 3

3rd contiguous subarray = [3 1 4] max = 4

4th contiguous subarray = [1 4 5] max = 5

5th contiguous subarray = [4 5 2] max = 5

6th contiguous subarray = [5 2 3] max = 5

7th contiguous subarray = [2 3 6] max = 6

Input: k = 4, arr[] = [8, 5, 10, 7, 9, 4, 15, 12, 90, 13]

Output: [10, 10, 10, 15, 15, 90, 90]

Explanation:

1st contiguous subarray = [8 5 10 7], max = 10

2nd contiguous subarray = [5 10 7 9], max = 10

3rd contiguous subarray = [10 7 9 4], max = 10

4th contiguous subarray = [7 9 4 15], max = 15

5th contiguous subarray = [9 4 15 12], max = 15

6th contiguous subarray = [4 15 12 90], max = 90

7th contiguous subarray = {15 12 90 13}, max = 90

Expected Time Complexity: **O(n)**

Expected Auxiliary Space**: O(k)**

2.Indexes of Subarray Sum

Given an unsorted array arr of size n that contains only non negative integers, find a sub-array (continuous elements) that has sum equal to s. You mainly need to return the left and right indexes(1-based indexing) of that subarray.

In case of multiple subarrays, return the subarray indexes which come first on moving from left to right. If no such subarray exists return an array consisting of element -1.

Examples:

Input: arr[] = [1,2,3,7,5], n = 5, s = 12

Output: 2 4

Explanation: The sum of elements from 2nd to 4th position is 12.

Input: arr[] = [1,2,3,4,5,6,7,8,9,10], n = 10, s = 15,

Output: 1 5

Explanation: The sum of elements from 1st to 5th position is 15.

Input: arr[] = [7,2,1], n = 3, s = 2

Output: 2 2

Explanation: The sum of elements from 2nd to 2nd position is 2.

Input: arr[] = [5,3,4], n = 3, s = 2

Output: -1

Explanation: There is no subarray with sum 2

Expected Time Complexity: **O(n)**

Expected Auxiliary Space: **O(1)**

3.Not a subset sum

Given a sorted array arr[] of positive integers, find the smallest positive integer such that it cannot be represented as the sum of elements of any subset of the given array set.

Examples:

Input: arr[] = [1, 2, 3]

Output: 7

Explanation: 7 is the smallest positive number for which no subset is there with sum 7.

Input: arr[] = [3, 6, 9, 10, 20, 28]

Output: 1

Explanation: 1 is the smallest positive number for which no subset is there with sum 1.

Expected Time Complexity: **O(n)**

Expected Auxiliary Space: **O(1)**

4.Next Greater Element

Given an array arr[ ] of n elements, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Examples

Input: arr[] = [1 3 2 4], n = 4

Output: 3 4 4 -1

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Input: arr[] [6 8 0 1 3], n = 5

Output: 8 -1 1 3 -1

Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Input: arr[] [10, 20, 30, 50], n = 4

Output: 20 30 50 -1

Explanation: For a sorted array, the next element is next greater element also exxept for the last element.

Input: arr[] [50, 40, 30, 10], n = 4

Output: -1 -1 -1 -1

Explanation: For a reverse sorted array, the next greater element is always 1.

Expected Time Complexity : **O(n)**

Expected Auxiliary Space : **O(n)**

5.Minimum Jumps

Given an array arr[] of non-negative integers. Each array element represents the maximum length of the jumps that can be made forward from that element. This means if arr[i] = x, then we can jump any distance y such that y ≤ x.

Find the minimum number of jumps to reach the end of the array starting from the first element. If an element is 0, then you cannot move through that element.

Note: Return -1 if you can't reach the end of the array.

Examples :

Input: arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}

Output: 3

Explanation:First jump from 1st element to 2nd element with value 3. From here we jump to 5th element with value 9, and from here we will jump to the last.

Input: arr = {1, 4, 3, 2, 6, 7}

Output: 2

Explanation: First we jump from the 1st to 2nd element and then jump to the last element.

Input: arr = {0, 10, 20}

Output: -1

Explanation: We cannot go anywhere from the 1st element.

Expected Time Complexity: **O(n)**

Expected Space Complexity: **O(1)**