3254. Find the Power of K-Size Subarrays I

You are given an array of integers `nums` of length `n` and a *positive* integer `k`.

The **power** of an array is defined as:

- Its **maximum** element if *all* of its elements are **consecutive** and **sorted** in **ascending** order.
- -1 otherwise.

You need to find the **power** of all subarrays of `nums` of size `k`.
Return an integer array `results` of size `n - k + 1`, where `results[i]` is the *power* of `nums[i..(i + k - 1)]`.

Example 1:

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

Output: [3,4,-1,-1,-1]

Explanation:

There are 5 subarrays of `nums` of size 3:

- `[1, 2, 3]` with the maximum element 3.
- `[2, 3, 4]` with the maximum element 4.
- `[3, 4, 3]` whose elements are **not** consecutive.
- `[4, 3, 2]` whose elements are **not** sorted.
- `[3, 2, 5]` whose elements are **not** consecutive.

Example 2:

Input: nums = [2,2,2,2,2], k = 4

Output: [-1,-1]

Example 3:

Input: nums = [3,2,3,2,3,2], k = 2

Output: [-1,3,-1,3,-1]

Constraints:

- `1 <= n == nums.length <= 500`
- `1 <= nums[i] <=` 10^5
- `1 <= k <= n`

3254. Find the Power of K-Size Subarrays I

```
/*
   Brute Force
   Time complexity: O(n^2)
   Space complexity: O(1)
*/
typedef std::vector<int> vi;
class Solution {
   public:
      vi resultsArray(vi& nums, int k){
         int n=nums.size();

         // Check if all integer are consecutive in a given range
         // Time complexity: O(n)
         // Space complexity: O(1)
         auto is_consecutive=[&](int left,int right)->bool{
            for(int i=left;i<right;++i){
               if(nums[i+1]-nums[i]!=1) return false;
            }
            return true;
         };

         vi ans; // Store final answer
         // For each starting index i, we extract the subarray of elements from nums[i] to nums[i+k−1].
         // We then need to verify two conditions:
         // 1- the elements must be sorted in ascending order,
         // 2- and they must be consecutive integers.
         for(int i=0;i<=n-k;++i){
            int j=i+k-1;
            if(is_consecutive(i,j)) ans.push_back(nums[j]);
            else ans.push_back(-1);
         }
         return ans;
      }
   };
```

3254. Find the Power of K-Size Subarrays I

```
/*
    Sliding window
    Time complexity: O(n)
    Space complexity: O(1)
*/
typedef std::vector<int> vi;
class Solution {
    public:
        vi resultsArray(vi& nums, int k){
            int n=nums.size();

            // Check if all integer are consecutive in a given range
            // Time complexity: O(n)
            // Space complexity: O(1)
            auto is_consecutive=[&](int left,int right)->bool{
                for(int i=left;i<right;++i){
                    if(nums[i+1]-nums[i]!=1) return false;
                }
                return true;
            };

            vi ans; // Store final answer

            // Create a window of size k
            int win=0;
            for(int i=0;i<k;++i) win+=nums[i];

            // If the integers are NOT consecutive, add -1 to the answer
            if(!is_consecutive(0,k-1)) ans.push_back(-1);
            // Otherwise add the last element of the window to the answer
            else ans.push_back(nums[k-1]);
            // Slide the window
            int i=0;
            while(i<n-k){
                // Slide the window to the right
                win-=nums[i];
                win+=nums[i+k];

                // Check if the elements of the new window are consecutive
                bool is_consecutive_win=k==1 || nums[i+k]-nums[i+k-1]==1;

                i++;

                int a=nums[i]; // First element in the window
                int b=nums[i+k-1]; // Last element in the window

                // If the integers are consecutive, so mathematically
                // the sum should be a+(a+1)+(a+2)+...+b
                long long sum=(b*1ll*(b+1)*1ll/2)-((a-1)*1ll*a*1ll/2);
                // If the window's does not fit the mathematical sum
                // or the integer are NOT consecutive
                // means elements are NOT consecutive and NOT sorted in ascending order
                // Add -1 to the answer
                if(win!=sum || !is_consecutive_win) ans.push_back(-1);
                // Otherwise add the last element of the window to the answer
                else ans.push_back(nums[i+k-1]);
            }
            return ans;
        }
    };
```