Problem 3255: Find the Power of K-Size Subarrays II

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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 <= 10

5

1 <= nums[i] <= 10

6

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
vector<int> resultsArray(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int[] resultsArray(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def resultsArray(self, nums: List[int], k: int) -> List[int]:
```

Python:

```
class Solution(object):
def resultsArray(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var resultsArray = function(nums, k) {

};
```

TypeScript:

```
function resultsArray(nums: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] ResultsArray(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* resultsArray(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

```
func resultsArray(nums []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun resultsArray(nums: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func resultsArray(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn results_array(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
```

```
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def results_array(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function resultsArray($nums, $k) {

}
}
```

Dart:

```
class Solution {
List<int> resultsArray(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def resultsArray(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec results_array(nums :: [integer], k :: integer) :: [integer]
def results_array(nums, k) do

end
end
```

Erlang:

```
-spec results_array(Nums :: [integer()], K :: integer()) -> [integer()].
results_array(Nums, K) ->
.
```

Racket:

```
(define/contract (results-array nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> resultsArray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] resultsArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Power of K-Size Subarrays II
Difficulty: Medium
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def resultsArray(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def resultsArray(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
var resultsArray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function resultsArray(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
```

```
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] ResultsArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Power of K-Size Subarrays II
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* resultsArray(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find the Power of K-Size Subarrays II
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func resultsArray(nums []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun resultsArray(nums: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func resultsArray(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Power of K-Size Subarrays II
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn results_array(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
```

```
# @return {Integer[]}
def results_array(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function resultsArray($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> resultsArray(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def resultsArray(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec results_array(nums :: [integer], k :: integer) :: [integer]
def results_array(nums, k) do
```

```
end
end
```

Erlang Solution:

```
-spec results_array(Nums :: [integer()], K :: integer()) -> [integer()].
results_array(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (results-array nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```