Problem 1708: Largest Subarray Length K

Problem Information

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

Problem Description

An array

A

is larger than some array

B

if for the first index

i

where

A[i] != B[i]

,

A[i] > B[i]

.

For example, consider

0

-indexing:

[1,3,2,4] > [1,2,2,4]

, since at index

1

,

3 > 2

.

[1,4,4,4] < [2,1,1,1]

, since at index

0

,

1 < 2

.

A subarray is a contiguous subsequence of the array.

Given an integer array

nums

of

distinct

integers, return the

largest

subarray of

nums

of length

k

.

Example 1:

Input:

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

Output:

[5,2,3]

Explanation:

The subarrays of size 3 are: [1,4,5], [4,5,2], and [5,2,3]. Of these, [5,2,3] is the largest.

Example 2:

Input:

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

Output:

[4,5,2,3]

Explanation:

The subarrays of size 4 are: [1,4,5,2], and [4,5,2,3]. Of these, [4,5,2,3] is the largest.

Example 3:

Input:

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

Output:

[5]

Constraints:

1 <= k <= nums.length <= 10

5

1 <= nums[i] <= 10

9

All the integers of

nums

are

unique

.

Follow up:

What if the integers in

nums

are not distinct?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (largest-subarray nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
```

```
* 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> largestSubarray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
* 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[] largestSubarray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Subarray Length K
Difficulty: Easy
Tags: array, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
* 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 largestSubarray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
* 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 largestSubarray(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
* 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[] LargestSubarray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Largest Subarray Length K
* Difficulty: Easy
* Tags: array, greedy
*
* 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* largestSubarray(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Largest Subarray Length K
// Difficulty: Easy
// Tags: array, greedy
//
// 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 largestSubarray(nums []int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Subarray Length K
// Difficulty: Easy
// Tags: array, greedy
//
// 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 largest_subarray(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (largest-subarray nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```