Problem 239: Sliding Window Maximum

Problem Information

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

Problem Description

You are given an array of integers

nums

, there is a sliding window of size

k

which is moving from the very left of the array to the very right. You can only see the

k

numbers in the window. Each time the sliding window moves right by one position.

Return

the max sliding window

.

Example 1:

Input:

nums = [1,3,-1,-3,5,3,6,7], k = 3

Output:

[3,3,5,5,6,7]

Explanation:

Window position Max --------------- ----- [1 3 -1] -3 5 3 6 7

3

1 [3 -1 -3] 5 3 6 7

3

1 3 [-1 -3 5] 3 6 7

5

1 3 -1 [-3 5 3] 6 7

5

1 3 -1 -3 [5 3 6] 7

6

1 3 -1 -3 5 [3 6 7]

7

Example 2:

Input:

nums = [1], k = 1

Output:

[1]

Constraints:

1 <= nums.length <= 10

5

-10

4

<= nums[i] <= 10

4

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-sliding-window nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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> maxSlidingWindow(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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[] maxSlidingWindow(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sliding Window Maximum
Difficulty: Hard
Tags: array, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 maxSlidingWindow = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 maxSlidingWindow(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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[] MaxSlidingWindow(int[] nums, int k) {

}

}
```

C Solution:

```
/*
* Problem: Sliding Window Maximum
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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* maxSlidingWindow(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sliding Window Maximum
// Difficulty: Hard
// Tags: array, queue, heap
//
// 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 maxSlidingWindow(nums []int, k int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sliding Window Maximum
// Difficulty: Hard
// Tags: array, queue, heap
//
// 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 max_sliding_window(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-sliding-window nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```