Problem 644: Maximum Average Subarray II

Problem Information

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

Problem Description

You are given an integer array

nums

consisting of

n

elements, and an integer

k

.

Find a contiguous subarray whose

length is greater than or equal to

k

that has the maximum average value and return

this value

. Any answer with a calculation error less than

10

-5

will be accepted.

Example 1:

Input:

nums = [1,12,-5,-6,50,3], k = 4

Output:

12.75000

Explanation:

- When the length is 4, averages are [0.5, 12.75, 10.5] and the maximum average is 12.75 - When the length is 5, averages are [10.4, 10.8] and the maximum average is 10.8 - When the length is 6, averages are [9.16667] and the maximum average is 9.16667 The maximum average is when we choose a subarray of length 4 (i.e., the sub array [12, -5, -6, 50]) which has the max average 12.75, so we return 12.75 Note that we do not consider the subarrays of length < 4.

Example 2:

Input:

nums = [5], k = 1

Output:

5.00000

Constraints:

n == nums.length

1 <= k <= n <= 10

4

-10

4

<= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findMaxAverage(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: float
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
double findMaxAverage(int* nums, int numsSize, int k) {

}
```

Go:

```
func findMaxAverage(nums []int, k int) float64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn find_max_average(nums: Vec<i32>, k: i32) -> f64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-max-average nums k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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:
double findMaxAverage(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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 double findMaxAverage(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Average Subarray II
```

```
Difficulty: Hard
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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 findMaxAverage = function(nums, k) {


};
```

TypeScript Solution:

```
/**
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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 findMaxAverage(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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 double FindMaxAverage(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Average Subarray II
* Difficulty: Hard
* Tags: array, search
*
* 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
*/

double findMaxAverage(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Average Subarray II
// Difficulty: Hard
// Tags: array, search
//
// 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 findMaxAverage(nums []int, k int) float64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Average Subarray II
// Difficulty: Hard
// Tags: array, search
//
// 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 find_max_average(nums: Vec<i32>, k: i32) -> f64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-max-average nums k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```