Problem 813: Largest Sum of Averages

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. You can partition the array into

at most

k

non-empty adjacent subarrays. The

score

of a partition is the sum of the averages of each subarray.

Note that the partition must use every integer in

nums

, and that the score is not necessarily an integer.

Return

the maximum

score

you can achieve of all the possible partitions

. Answers within

10

-6

of the actual answer will be accepted.

Example 1:

Input:

nums = [9,1,2,3,9], k = 3

Output:

20.00000

Explanation:

The best choice is to partition nums into [9], [1, 2, 3], [9]. The answer is 9 + (1 + 2 + 3) / 3 + 9 = 20. We could have also partitioned nums into [9, 1], [2], [3, 9], for example. That partition would lead to a score of 5 + 2 + 6 = 13, which is worse.

Example 2:

Input:

nums = [1,2,3,4,5,6,7], k = 4

Output:

20.50000

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 10

4

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (largest-sum-of-averages nums k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Largest Sum of Averages
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
```

```
var largestSumOfAverages = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```c
/*
* Problem: Largest Sum of Averages
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```go
// Problem: Largest Sum of Averages
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func largestSumOfAverages(nums []int, k int) float64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Largest Sum of Averages
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (largest-sum-of-averages nums k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```