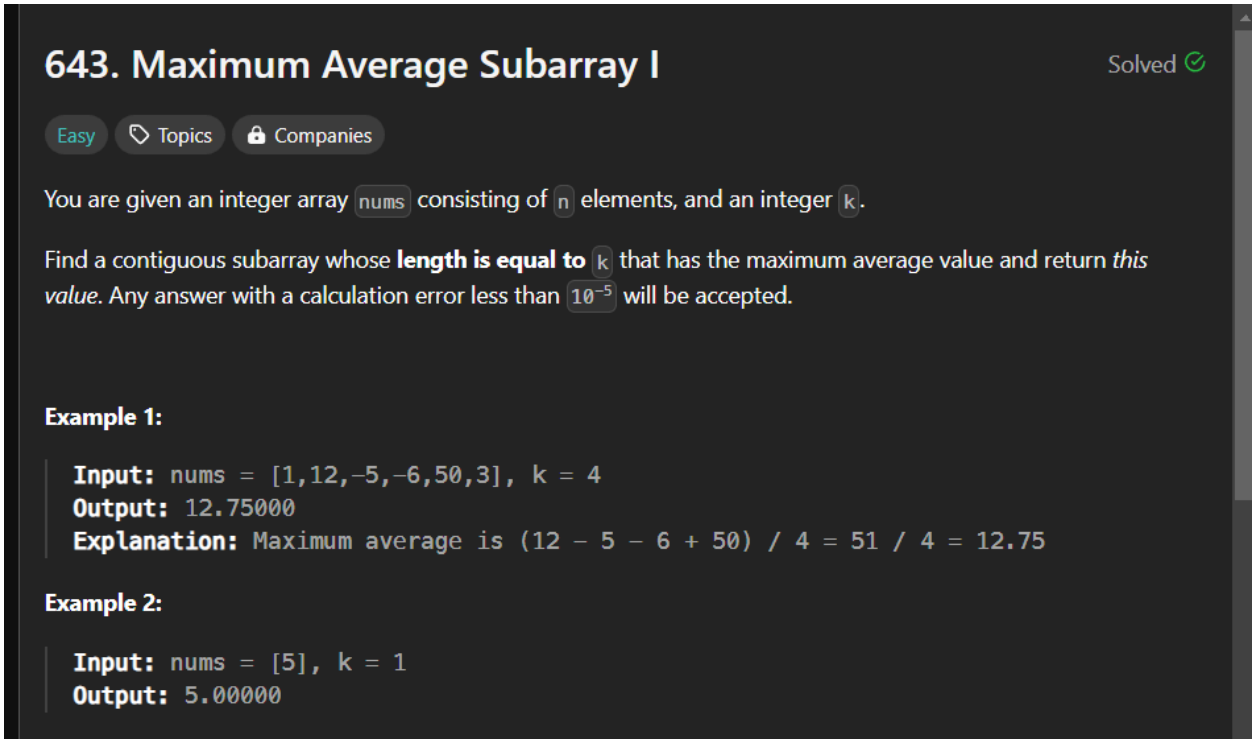
1: Leetcode 643: Maximum Average Subarray I

⇒ https://leetcode.com/problems/maximum-average-subarray-i/description/



643. Maximum Average Subarray I

Solved

Easy | Topics | Companies

You are given an integer array `nums` consisting of `n` elements, and an integer `k`.

Find a contiguous subarray whose **length is 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: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75
```

Example 2:

```
Input: nums = [5], k = 1
Output: 5.00000
```

A: PYTHON doubts:

→

- while loop → while:
- no ++ → use +=1
- For loop is equivalent to
  int sum = 0;
  for (int i = 0; i < k; i++)
  sum += nums[i];
  int maxSum = sum;

  →
  sum = sum(nums[:k])
  max_sum = sum

B: Code -

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

```
        currentSum = sum(nums[:k])
        maxSum = currentSum

        for i in range(k, len(nums)):
            currentSum += nums[i] - nums[i - k]
            maxSum = max(maxSum, currentSum)

        return maxSum / k
```

C: Written notes

Week-1 [sliding window]

Arrays-1 → Learn (4)

~~Sliding Window~~

1) 643: Maximum Average Subarray I

→ Brute force method: find all subarrays, calc sum for all.

for (i=0 to i=n-1)
for (j=i to j=n-1) → end index
for (k=i to k=n-1) → sum calc.

O(n³).

2) Sliding Window approach: [Nikhil Lohia YT].

⇒ { 1, 12, -5, -6, 50, 3, 0, -45, 23 }

sum = 2 2-1+50 = 51 → 51-12+3 = 42 ....

a) ~~Calc sum for start window (s=0, end=k-1)~~
b) Initialize window → start=0, end=k-1)
c) shift to right, s++, end++
d) sum - element gone + new element.
e) store the max sum b/w old & new.
f) return sum/k.

2) Kadane's algorithm: [Leetcode 10].

⇒ [-2, 1, -3, 4, -1, 2, 1, -5, 4]

-2 +1 -2× 4 → 3 → 5 → 6 → 1 → 5

a) if prev sum is -ve, ignore it, start new.
b) define / initialize: maxSum to 1st element, curSum to 0
c) Iterate through array → a) if curSum is -ve, set to 0.
b) add each element to curSum
c) set maxSum as max of maxSum & curSum
d) return maxSum / k