# 644. Maximum Average Subarray II 💆

July 15, 2017 | 15K views

Average Rating: 4.65 (26 votes)

Given an array consisting of n integers, find the contiguous subarray whose length is greater than or **equal to k** that has the maximum average value. And you need to output the maximum average value.

Example 1:

```
Input: [1,12,-5,-6,50,3], k = 4
 Output: 12.75
 Explanation:
 when length is 5, maximum average value is 10.8,
 when length is 6, maximum average value is 9.16667.
 Thus return 12.75.
Note:
```

## 1. 1 <= k <= n <= 10,000.

- Elements of the given array will be in range [-10,000, 10,000].
- 3. The answer with the calculation error less than 10<sup>-5</sup> will be accepted.

# Approach #1 Iterative method [Time Limit Exceeded]

Solution

### equal to k and to determine the maximum average from out of those. But, instead of finding out this sum in a naive manner for every subarray with length greater than or equal to k separately, we can do as follows.

between s and i is greater than the average found till now or not.

For every starting point, s, considered, we can iterate over the elements of nums starting from nums, and keep a track of the sum found till the current index(i). Whenever the index reached is such that the number of elements lying between s and i is greater than or equal to k, we can check if the average of the elements

One of the simplest solutions is to consider the sum of every possible subarray with length greater than or

Copy Copy Java 1 public class Solution { public double findMaxAverage(int[] nums, int k) { double res = Integer.MIN\_VALUE;

```
for (int s = 0; s < nums.length - k + 1; s++) {
                long sum = 0;
                for (int i = s; i < nums.length; i++) {
                   sum += nums[i];
                   if (i - s + 1 >= k)
                        res = Math.max(res, sum * 1.0 / (i - s + 1));
 10
 11
 12
             return res;
 13
         }
 14 }
 15
Complexity Analysis
  • Time complexity : O(n^2). Two for loops iterating over the whole length of nums with n elements.
   • Space complexity : O(1). Constant extra space is used.
```

- Approach #2 Using Binary Search [Accepted]
- Algorithm

# Firstly, we know that the value of the average could lie between the range (min, max). Here, min and

# max refer to the minimum and the maximum values out of the given nums array. This is because, the

average can't be lesser than the minimum value and can't be larger than the maximum value.

To understand the idea behind this method, let's look at the following points.

But, in this case, we need to find the maximum average of a subarray with atleast k elements. The idea in this method is to try to approximate(guess) the solution and to try to find if this solution really exists. If it exists, we can continue trying to approximate the solution even to a further precise value, but choosing a larger number as the next approximation. But, if the initial guess is wrong, and the initial maximum average

value(guessed) isn't possible, we need to try with a smaller number as the next approximate.

Now, instead of doing the guesses randomly, we can make use of Binary Search. With min and max as the initial numbers to begin with, we can find out the mid of these two numbers given by (min + max)/2. Now, we need to find if a subarray with length greater than or equal to k is possible with an average sum

say that  $(a_1 + a_2 + a_3... + a_j)/j \ge mid$  or  $(a_1 + a_2 + a_3... + a_j) \ge j * mid$  or

To determine if this is possible in a single scan, let's look at an observation. Suppose, there exist j elements,  $a_1, a_2, a_3..., a_i$  in a subarray within nums such that their average is greater than mid. In this case, we can

```
Thus, we can see that if after subtracting the mid number from the elements of a subarray with more than
k-1 elements, within nums, if the sum of elements of this reduced subarray is greater than 0, we can
```

element, making use of the following idea.

minimum  $sum_i$ , it can never be satisfied with a larger value.

such that  $j - i \ge k$ .

Java

9

10

11

12 13

O Previous

Comments: 15

achieve an average value greater than mid. Thus, in this case, we need to set the mid as the new minimum

element and continue the process.

greater than this mid value.

Otherwise, if this reduced sum is lesser than 0 for all subarrays with greater than or equal to 
$$k$$
 elements, we

 $(a_1 - mid) + (a_2 - mid) + (a_3 - mid) ... + (a_j - mid) \ge 0$ 

process. In order to determine if such a subarray exists in a linear manner, we keep on adding nums[i] - mid to the sum obtained till the  $i^{th}$  element while traversing over the nums array. If on traversing the first kelements, the sum becomes greater than or equal to 0, we can directly determine that we can increase the average beyond mid. Otherwise, we continue making additions to sum for elements beyond the  $k^{th}$ 

can't achieve mid as the average. Thus, we need to set mid as the new maximum element and continue the

If we know the cumulative sum upto indices i and j, say  $sum_i$  and  $sum_j$  respectively, we can determine the sum of the subarray between these indices(including j) as  $sum_j - sum_i$ . In our case, we want this difference between the cumulative sums to be greater than or equal to 0 as discusssed above. Further, for  $sum_i$  as the cumulative sum upto the current( $i^{th}$ ) index, all we need is  $sum_i - sum_i \ge 0$ 

To fulfil this, we make use of a prev variable which again stores the cumulative sums but, its current index(for cumulative sum) lies behind the current index for sum at an offset of k units. Thus, by finding the minimum out of prev and the last minimum value, we can easily find out the required minimum sum value.

To achive this, instead of checking with all possible values of  $sum_i$ , we can just consider the minimum

cumulative sum upto the index j-k. This is because if the required condition can't be sastisfied with the

Every time after checking the possiblility with a new mid value, at the end, we need to settle at some value as the average. But, we can observe that eventually, we'll reach a point, where we'll keep moving near some same value with very small changes. In order to keep our precision in control, we limit this process to  $10^-5$ 

precision, by making use of error and continuing the process till error becomes lesser than 0.00001.

1 public class Solution { public double findMaxAverage(int[] nums, int k) { double max\_val = Integer.MIN\_VALUE; double min\_val = Integer.MAX\_VALUE; for (int n: nums) { max\_val = Math.max(max\_val, n); 6 min\_val = Math.min(min\_val, n); 8

double prev\_mid = max\_val, error = Integer.MAX\_VALUE;

double mid = (max\_val + min\_val) \* 0.5;

while (error > 0.00001) {

if (check(nums, mid, k))

min\_val = mid;

**С**ору

Next

Sort By ▼

Post

```
14
 15
                   max_val = mid;
 16
               error = Math.abs(prev_mid - mid);
 17
               prev_mid = mid;
 18
 19
            return min_val;
 20
 21
        public boolean check(int[] nums, double mid, int k) {
 22
           double sum = 0, prev = 0, min_sum = 0;
 23
            for (int i = 0; i < k; i++)
 24
               sum += nums[i] - mid;
 25
            if (sum >= 0)
 26
             return true;
          for (int i = k; i < nums.length; i++) {
 27
               sum += nums[i] - mid.
Complexity Analysis
  • Time complexity: O(nlog(max\_val - min\_val)). check takes O(n) time and it is executed
     O(log(max\_val - min\_val)) times.
  • Space complexity : O(1). Constant Space is used.
Analysis written by: @vinod23
Rate this article: * * * *
```

The explanation is poorly written! Solution is awesome! 12 A V 🗗 Share 🦘 Reply

Preview

Type comment here... (Markdown is supported)

xshen93 🛊 -5 ② August 5, 2017 10:34 PM

@vinod23 Thanks. I got it.

1 A V C Share Share

SHOW 1 REPLY

SHOW 1 REPLY

haoyangfan \* 784 O December 19, 2018 1:24 AM

1 A V C Share Reply And why in the check function, the min\_sum should set to 0 not MAX\_VALUE? 1 A V C Share Share SHOW 1 REPLY xshen93 🖈 -5 ② August 4, 2017 10:53 PM

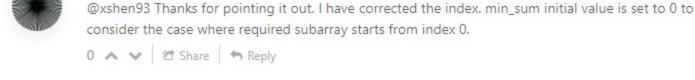
Why here "we can just consider the minimum cumulative sum upto the index j - i" is "j - i" not "j - k" ?

"min\_sum initial value is set to 0 to consider the case where required subarray starts from index 0"

I am still confused by why set initial value of min sum to be 0 instead of Double.MAX VALUE which is the technique often used when we want to keep track of min value of some set? Could anybody please explain the reason of using initial value of 0 instead of Double.MAX VALUE? I Read More 0 ∧ ∨ ☑ Share ¬ Reply

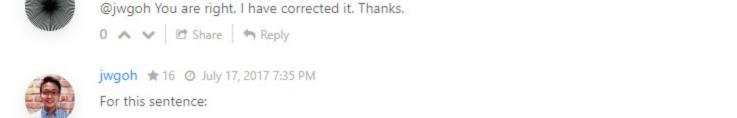
Could anyone can help explain this: for (int i = k; i < nums.length; i++) { sum += nums[i] - mid; Read More





vinod23 \* 425 July 18, 2017 8:50 AM

jedihy 🖈 1153 ② August 17, 2017 6:28 AM



Further, for sum\_i as the cumulative sum up to the current (i-th) index, all we need is sum\_i - sum\_j >= 0 such that j - i

Read More

0 ∧ ∨ ☑ Share 🦘 Reply 1 (2) (>)