Find maximum average subarray of k length

Given an array with positive and negative numbers, find the maximum average subarray of given length.

Example:

```
Input:  arr[] = {1, 12, -5, -6, 50, 3}, k = 4
Output: Maximum average subarray of length 4 begins
        at index 1.
Maximum average is (12 - 5 - 6 + 50)/4 = 51/4
```

We strongly recommend you to minimize your browser and try this yourself first.

A **Simple Solution** is to run two loops. The outer loop picks starting point, the inner loop goes till length ‘k’ from the starting point and computes average of elements. Time complexity of this solution is O(n*k).

A **Better Solution** is to create an auxiliary array of size n. Store cumulative sum of elements in this array. Let the array be csum[]. csum[i] stores sum of elements from arr[0] to arr[i]. Once we have csum[] array with us, we can compute sum between two indexes in O(1) time.
Below is C++ implementation of this idea. One observation is, a subarray of given length has maximum average if it has maximum sum. So we can avoid floating point arithmetic by just comparing sum.

```
// C++ program to find maximum average subarray
// of given length.
#include<bits/stdc++.h>
using namespace std;

// Returns beginning index of maximum average
// subarray of length 'k'
int findMaxAverage(int arr[], int n, int k)
{
    // Check if 'k' is valid
    if (k > n)
        return -1;

    // Create and fill array to store cumulative
    // sum. csum[i] stores sum of arr[0] to arr[i]
    int *csum = new int[n];
    csum[0] = arr[0];
    for (int i=1; i<n; i++)
       csum[i] = csum[i-1] + arr[i];
```

```
    // Initialize max_sm as sum of first subarray
    int max_sum = csum[k-1], max_end = k-1;

    // Find sum of other subarrays and update
    // max_sum if required.
    for (int i=k; i<n; i++)
    {
        int curr_sum = csum[i] - csum[i-k];
        if (curr_sum > max_sum)
        {
            max_sum = curr_sum;
            max_end = i;
        }
    }

    delete [] csum; // To avoid memory leak

    // Return starting index
    return max_end - k + 1;
}

// Driver program
int main()
{
    int arr[] = {1, 12, -5, -6, 50, 3};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "The maximum average subarray of "
         "length "<< k << " begins at index "
         << findMaxAverage(arr, n, k);
    return 0;
}
```

Run on IDE

Output:

```
The maximum average subarray of length 4 begins at index 1
```

Time Complexity of above solution is O(n), but it requires O(n) auxiliary space.

We can avoid need of extra space by using below **Efficient Method**.

1) Compute sum of first ‘k’ elements, i.e., elements arr[0..k-1]. Let this sum be ‘sum’. Initialize ‘max_sum’ as ‘sum’

2) Do following for every element arr[i] where i varies from ‘k’ to ‘n-1′

…….a) Remove arr[i-k] from sum and add arr[i], i.e., do sum += arr[i] – arr[i-k]

…….b) If new sum becomes more than max_sum so far, update max_sum.

3) Return ‘max_sum’

```
// C++ program to find maximum average subarray
// of given length.
#include<bits/stdc++.h>
using namespace std;

// Returns beginning index of maximum average
// subarray of length 'k'
int findMaxAverage(int arr[], int n, int k)
{
```

```
    // Check if 'k' is valid
    if (k > n)
        return -1;

    // Compute sum of first 'k' elements
    int sum = arr[0];
    for (int i=1; i<k; i++)
        sum += arr[i];

    int max_sum = sum, max_end = k-1;

    // Compute sum of remaining subarrays
    for (int i=k; i<n; i++)
    {
        int sum = sum + arr[i] - arr[i-k];
        if (sum > max_sum)
        {
            max_sum = sum;
            max_end = i;
        }
    }

    // Return starting index
    return max_end - k + 1;
}

// Driver program
int main()
{
    int arr[] = {1, 12, -5, -6, 50, 3};
    int k = 4;
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "The maximum average subarray of "
         "length "<< k << " begins at index "
         << findMaxAverage(arr, n, k);
    return 0;
}
```

Run on IDE

Output:

```
The maximum average subarray of length 4 begins at index 1
```

Time complexity of this method is also O(n), but it requires constant extra space.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

11 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Convert array into Zig-Zag fashion

(Login to Rate and Mark)

2.8 Average Difficulty : **2.8/5.0**
Based on **5** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

11 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion

Raj Vardhan • a month ago

Do we have to make the default assumption in such cases that the subarray has to be contiguous?

• Reply • Share ›

Ratnesh Chandak • a month ago

//it gives the greatest integer of the resultant average, O(N) and constant space

```
#include<iostream>

using namespace std;

int main(){

int arr[6]={1,12,-5,-6,50,3};

int k=4;

int maxarr=0,currsum=0,start=0,end=5;

for(int i=0;i<6;i++){

maxarr=maxarr+arr[i];

}

currsum=maxarr;
```

see more

• Reply • Share ›

Billionaire • 2 months ago

Awesome solutions!

Prefix sum, and caterpillar method

1 • Reply • Share ›

Satabdi Aditya • 4 months ago

```
public class MaxSubarray {

public static void main(String[] args){

int[] arr = {1, 12, -5, -6, 50, 3};

int k = 4;
```

```
int max = maxSubArr(arr, k);

System.out.println("The maximum is :" + max);

}

public static int maxSubArr(int[] arr,int k){

int max = 0;

for(int i = 0; i<=(arr.length-k);i++){

int sum = 0;
```

see more

• Reply • Share ›

Nitin • 4 months ago

why do you need to declare "sum" as local variable too? I don't think it is required.

• Reply • Share ›

shivam • 4 months ago

testing

• Reply • Share ›

kaushal • 4 months ago

```
#include<iostream>

using namespace std;

int main()

{

int n,k,max=0,sum=0,index=0;;

cin>>n>>k;

int a[n];

for(int i=0;i<n;i++) cin=""">>a[i];

for(int i=0;i<k;i++) sum+="a[i];" max="sum;" for(int="" i="k;i&amp;lt;n;i++)" {="" sum+="a[i]" -a[i-k];="" if(sum="">max)

{

max=sum;
```

```
index=i-k+1;

}

}

cout<<max<<endl<<index; }="">
```

• Reply • Share ›

Abhimanyu Sharma • 4 months ago

I think we can sort as well and read last k no to calculate average.

• Reply • Share ›

TulsiRam Abhimanyu Sharma • 3 months ago

Nahi bhai: SUBARRAY poochi hai. . sorting se elements idhar udhar ho jayenge

• Reply • Share ›

sourabh dhanoa Abhimanyu Sharma • 4 months ago

no we cant do as we need subarray of lenght "k" ..bcz by sorting you are changing order of array which will give you average of maximum numbers not subarray

.

2 • Reply • Share ›

Anirban Acharya • 4 months ago

here's a python implementation of the problem in O(n) complexity : http://ideone.com/UvwqgE using sliding window technique

2 • Reply • Share ›