

Maximum Sum Subarray of Size K

Σ SR Score	35306
🔗 Link	https://www.educative.io/courses/grokking-the-coding-interview/JPKr0kqLGNP
📅 Last Reviewed	@March 25, 2022
# Time	4
# Score	4
≡ DS	arrays
≡ Algo	sliding window
▼ Stated	easy
▼ Perceived	easy
▼ List	DONE
☑ Needs Review	<input type="checkbox"/>
☑ Repeat Offender	<input type="checkbox"/>
☑ Confident	<input checked="" type="checkbox"/>
Σ C_Date	17253
Σ C_Solution	2
Σ C_Time	400
▼ Frequency	

▼ Problem Statement

Problem Statement

Given an array of positive numbers and a positive number 'k,' find the **maximum sum of any contiguous subarray of size 'k'**.

Example 1:

Input: [2, 1, 5, 1, 3, 2], k=3

Output: 9

Explanation: Subarray with maximum sum is [5, 1, 3].

Example 2:

Input: [2, 3, 4, 1, 5], k=2

Output: 7

Explanation: Subarray with maximum sum is [3, 4].

▼ Intuition

- this is the type of sliding window problem where the window size is given & fixed $\rightarrow k$
- so instead of doing double for loops to go thru every combination of k length subarray, we can have two pointers which always create a window of k numbers
- we then add to a sum variable until the window size hits k for the first time
- then once $windowSize == k$, we move the `windowStart` pointer each iteration to shrink the subarray & move the `windowEnd` pointer up by 1 to grow the subarray back to size k
- each time the $windowSize == k$, we compare our current running sum against a ***maxSum*** variable which is ultimately what we will return at the end
 - this first happens when the subarray size reaches k , and every single loop iteration thereafter since we are maintaining the subarray size to be k by manipulating the `windowStart` & `windowEnd` pointers

▼ Time & Space Considerations

- Time: $O(2n) \rightarrow O(n)$
 - technically `windowEnd` pointer iterates thru end of array and `windowStart` iterates thru `len(arr) - 3`, so it's like $O(2n - 3)$
- Space: $O(1)$

- fixed # of variables created; doesn't depend on input size of arr

▼ Review Notes

▼ [3/2/22]

- had no idea how to do it, looked at full solution

▼ [3/25/22]

- can't recall, but def went better than first go-round

▼ [4/5/22]

- got optimal solution in ~5 min
- had to refresh for a sec abt how sliding window worked
- fumbled around with whether the question was asking for the maxSum or max length of window... which doesn't make sense since the problem is based around a fixed window size k
- still, overall encouraging that you got the basic idea down after not revisiting for a bit

▼ Tracking

Scores

Attempt #	Date	Time	Score
<u>3</u>	@April 5, 2022	5	5
<u>2</u>	@March 25, 2022	2	2
<u>1</u>	@March 2, 2022	1	1

▼ Solutions

```
# Attempt 3: 4/5/22
#time: O(n)
#space: O(1)
def max_sub_array_of_size_k(k, arr):
    windowStart = sum = maxSum = 0
    for windowEnd in range(len(arr)):
        sum += arr[windowEnd]
        if windowEnd >= k - 1:
            maxSum = max(maxSum, sum)
            sum -= arr[windowStart]
            windowStart += 1

    return maxSum
# -----
# Attempt 2: 3/25/22
# time: O(n)
# space: O(1)
# filled in retrospect, can't recall but def better than first time
def max_sub_array_of_size_k(k, arr):
    maxSum = currSum = windowStart = 0
    for windowEnd in range(len(arr)):
        num = arr[windowEnd]
        currSum += num
        if windowEnd >= k - 1:
            maxSum = max(maxSum, currSum)
            leftNum = arr[windowStart]
            currSum -= leftNum
            windowStart += 1

    return maxSum
# -----
# Attempt 1: 3/2/22
# time: O(n)
# space: O(1)
# had no idea, looked at solution
def max_sub_array_of_size_k(k, arr):
    max_sum = sum = window_start = 0
    for window_end in range(len(arr)):
        sum += arr[window_end]
        if window_end >= k - 1:
            max_sum = max(max_sum, sum)
            sum -= arr[window_start]
            window_start += 1

    return max_sum
```


▼ Resources

Maximum Sum Subarray of Size K (easy) - Grokking the Coding Interview: Patterns for Coding Questions

<https://www.educative.io/courses/grokking-the-coding-interview/JPKr0kqLGNP>

▼ GitHub

GCI-master-list/Pattern 1 - Sliding Window/Maximum Sum Subarray of Size K at main · psdev30/GCI-master-list
Contribute to psdev30/GCI-master-list development by creating an account on GitHub.

 <https://github.com/psdev30/GCI-master-list/tree/main/Pattern%201%20-%20Sliding%20Window/Maximum%20Sum%20Subarray%20of%20Size%20K>