Maximum Sum Subarray of Size K (easy)

We'll cover the following ⌃

- Problem Statement
- Try it yourself
- Solution
- Code
  - A better approach
  - Time Complexity
  - Space Complexity

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].
```

Try it yourself

Try solving this question here:

☕ Java	🐍 Python3	JS JS	Ⓒ C++

```
1  def max_sub_array_of_size_k(k, arr):
2    # TODO: Write your code here
3    return -1
4
```

Solution

A basic brute force solution will be to calculate the sum of all 'k' sized subarrays of the given array, to find the subarray with the highest sum. We can start from every index of the given array and add the next 'k' elements to find the sum of the subarray. Following is the visual

array and add the next 'K' elements to find the sum of the subarray. Following is the visual representation of this algorithm for Example-1:
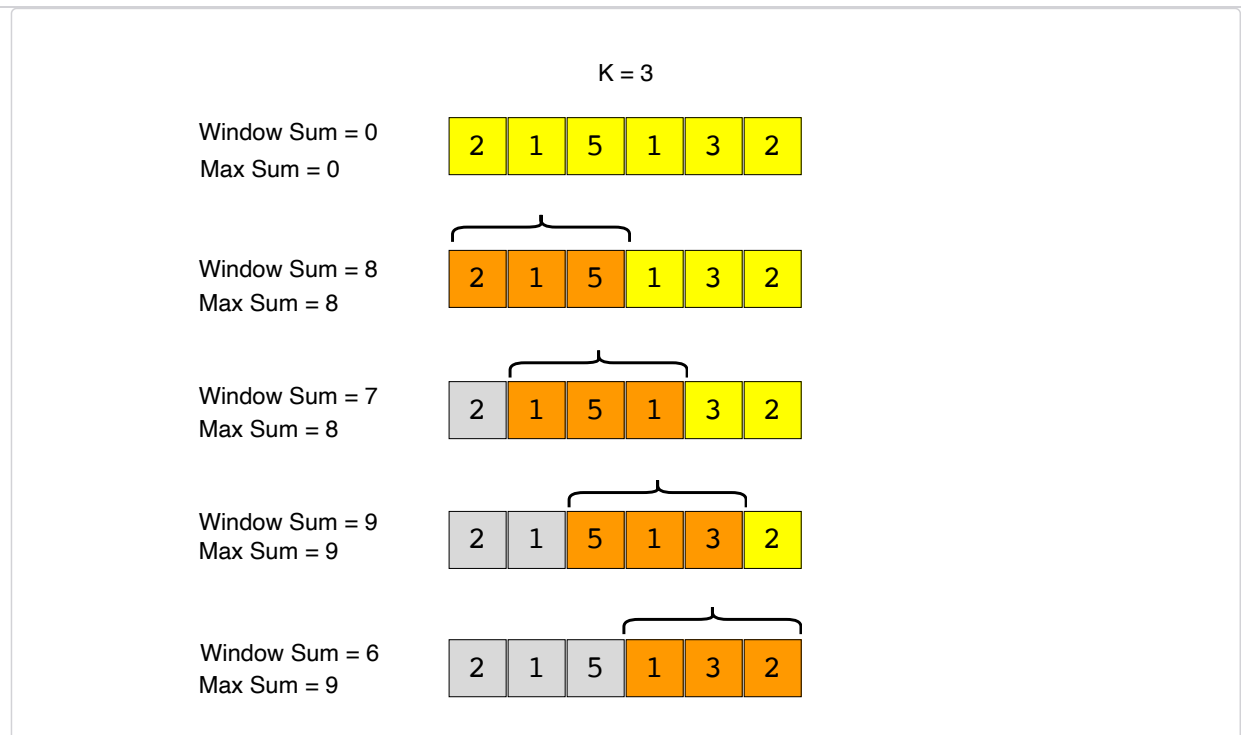
Code

Here is what our algorithm will look like:

| ☕ Java | 🐍 Python3 | C++ | JS JS |

```python
1   def max_sub_array_of_size_k(k, arr):
2     max_sum = 0
3     window_sum = 0
4
5     for i in range(len(arr) - k + 1):
6       window_sum = 0
7       for j in range(i, i+k):
8         window_sum += arr[j]
9       max_sum = max(max_sum, window_sum)
10    return max_sum
11
12
13  def main():
14    print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_size_k(3, [2, 1, 5,
15    print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_size_k(2, [2, 3, 4,
16
17
18  main()
19
```

The time complexity of the above algorithm will be $O(N * K)$, where 'N' is the total number of elements in the given array. Is it possible to find a better algorithm than this?

A better approach #

If you observe closely, you will realize that to calculate the sum of a contiguous subarray we

ahead by one element. So to slide the window forward and calculate the sum of the new position of the sliding window, we need to do two things:

1. Subtract the element going out of the sliding window i.e., subtract the first element of the window.

2. Add the new element getting included in the sliding window i.e., the element coming right after the end of the window.

This approach will save us from re-calculating the sum of the overlapping part of the sliding window. Here is what our algorithm will look like:

Java	Python3	C++	JS

```python
1   def max_sub_array_of_size_k(k, arr):
2     max_sum , window_sum = 0, 0
3     window_start = 0
4
5     for window_end in range(len(arr)):
6       window_sum += arr[window_end]  # add the next element
7       # slide the window, we don't need to slide if we've not hit the required window size o
8       if window_end >= k-1:
9         max_sum = max(max_sum, window_sum)
10        window_sum -= arr[window_start]  # subtract the element going out
11        window_start += 1  # slide the window ahead
12    return max_sum
13
14
15  def main():
16    print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_size_k(3, [2, 1, 5,
17    print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_size_k(2, [2, 3, 4,
18
19  main()
20
```

Time Complexity #

The time complexity of the above algorithm will be $O(N)$.

Space Complexity #

The algorithm runs in constant space $O(1)$.

✓ COMPLETED

← Back

Next →

Introduction

Smallest Subarray with a given sum (e...