Maximum Sum Subarray of Size K (easy)

We'll cover the following

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 | C++

```python
from typing import List


def max_sub_array_of_size_k(k: int, arr: List) -> int:
    winSum, winStart = 0, 0
    max_ans = -1
    for i, num in enumerate(arr):
        winSum += num
        if i >= k-1:
            max_ans = max(max_ans, winSum)
            winSum -= arr[winStart]
            winStart += 1
    return max_ans
```

Show Results Show Console

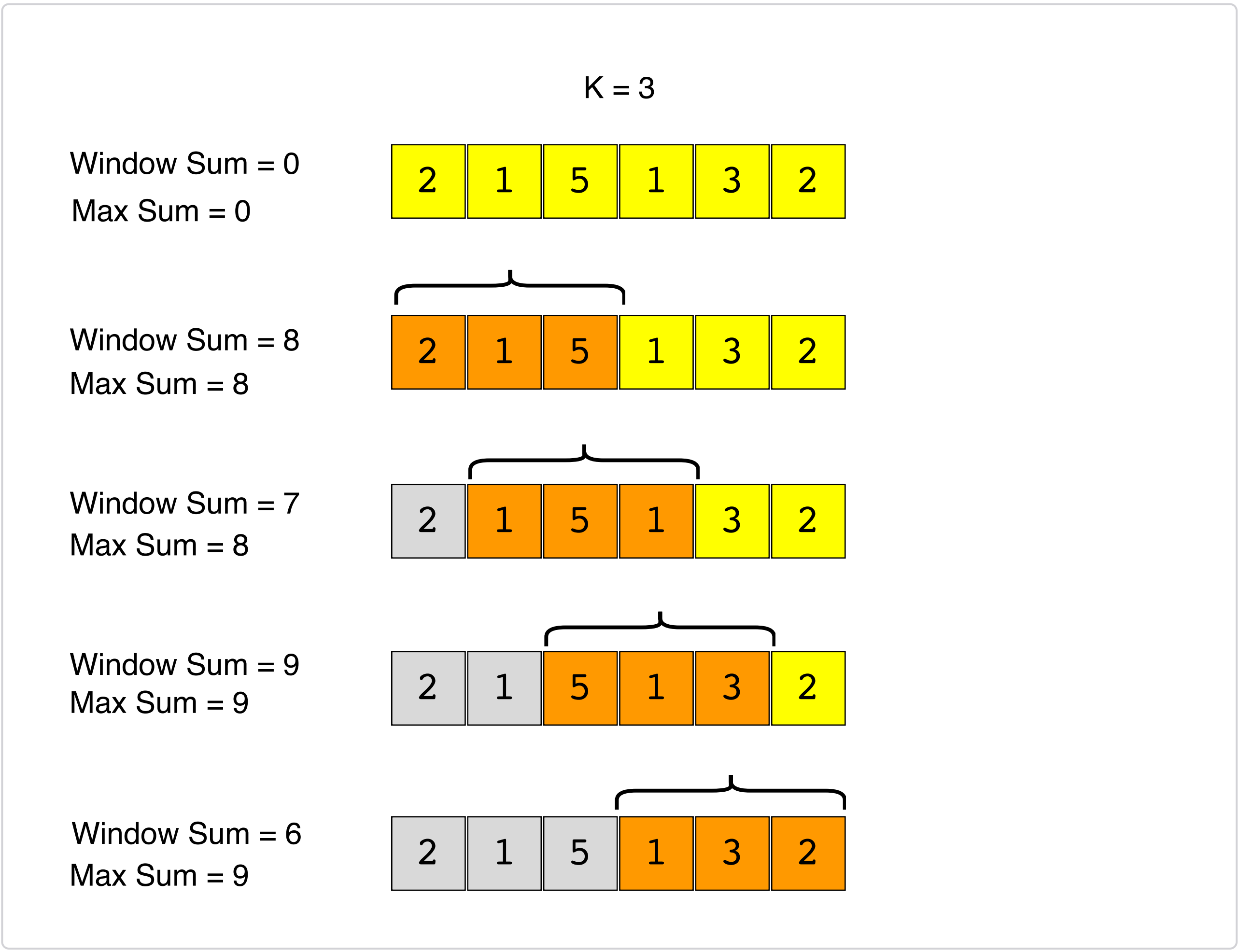
0.83s

2 of 2 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓	max_sub_array_of_size_k(3, [2, 1, 5, 1, …	9	9	Succeeded
✓	max_sub_array_of_size_k(2, [2, 3, 4, 1, …	7	7	Succeeded

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 subarray’s sum. Following is the visual representation of this algorithm for Example-1:



Code#

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
def max_sub_array_of_size_k(k, arr):
  max_sum = 0
  window_sum = 0

  for i in range(len(arr) - k + 1):
    window_sum = 0
    for j in range(i, i+k):
      window_sum += arr[j]
    max_sum = max(max_sum, window_sum)
  return max_sum


def main():
  print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_siz
  print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_siz


main()

```

Output 0.88s

```
Maximum sum of a subarray of size K: 9
Maximum sum of a subarray of size K: 7
```

The above algorithm's time complexity 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 can utilize the sum of the previous subarray. For this, consider each subarray as a **Sliding Window** of size 'k.' To calculate the sum of the next subarray, we need to slide the window 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
def max_sub_array_of_size_k(k, arr):
  max_sum , window_sum = 0, 0
  window_start = 0

  for window_end in range(len(arr)):
    window_sum += arr[window_end]  # add the next element
    # slide the window, we don't need to slide if we've not hit the requir
    if window_end >= k-1:
      max_sum = max(max_sum, window_sum)
      window_sum -= arr[window_start]  # subtract the element going out
      window_start += 1  # slide the window ahead
  return max_sum


def main():
  print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_siz
  print("Maximum sum of a subarray of size K: " + str(max_sub_array_of_siz

main()

```

Output 0.93s

```
Maximum sum of a subarray of size K: 9
Maximum sum of a subarray of size K: 7
```

Time Complexity#

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

Space Complexity#

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

← Back

Introduction

Next →

Smallest Subarray with a given sum (e...

✓ Mark as Completed

Report an Issue