Villanova Programming Team

Agenda

- Sliding Window
- When to Use
- Fixed vs Variable Size Windows

Introduction to Sliding Window Technique

The Sliding Window Technique is a problem-solving method used for optimizing iterative operations over subarrays.

It helps reduce the time complexity by converting nested loops into a single loop.

Two Types

Fixed window – problems involve performing some operation on every subarray of size k within the original array.

Variable window – problems involve finding the optimal subarray of any length within the original array.

Fixed Sliding Window Example

Maximum Average Subarray

Find a contiguous subarray whose **length is equal to** k that has the maximum average value.

Example: nums = [1,12,-5,-6,50,3], k = 4

Algorithm

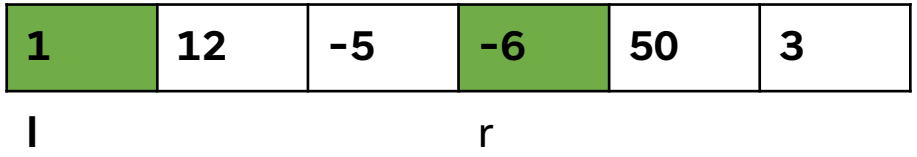
Use two pointers, l,r, to mark beginning and end of subarray.

Get the **sum** and **average** of the first subarray of size k.

To get the average of every subsequent subarray:

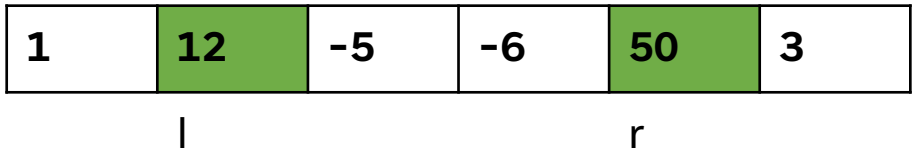
- Subtract the value at nums[l] from the sum and move pointer l over by 1
- Move pointer r over by 1 and add nums[r] to the sum.
- Calculate new average
- **Keep track of the new sum and maximum average.**

k=4



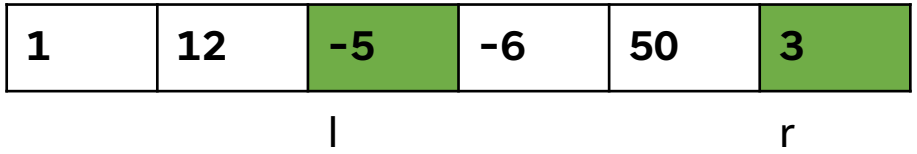
Get the average of the fist window marked by l and r.
sum = 1+12+-5+-6 = 2
average = sum/4 = 0.5
max average = 0.5

k=4



Get the average of the next window.
Don’t recompute the entire sum.
Take previous sum and subtract 1 and add 50.
sum = 2 – 1 + 50 = 49
average = 49/4 = 12.25
max average = 12.25

k=4



Take previous sum and subtract 12 and add 3.
sum = 49 - 12 + 3 = 40
average 40/4 = 10
max average = 12.5

Variable Sliding Window Example

Minimum Size Subarray Sum

Find the minimal length of a subarray whose sum is greater than or equal to a target.

Example: nums = [2,3,1,2,4,3], target = 7

Algorithm:

Use two pointers, l and r to keep track of window size.

Set initial window size to 1.

Repeat:

> Increase window size by 1 by moving r pointer over by one until the sum of the values in the window is greater than or equal to the target.

Store window size.

Decrease window size by 1 by moving l pointer to the right.

Continue Repeat step above.

Target = 7

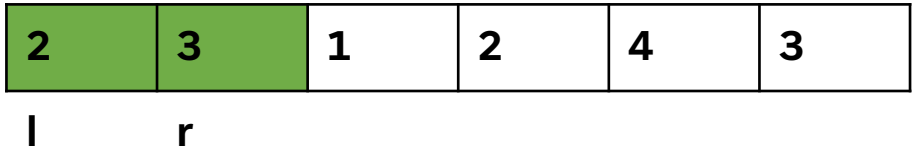
2	3	1	2	4	3

l,r

sum=2

Increase window size

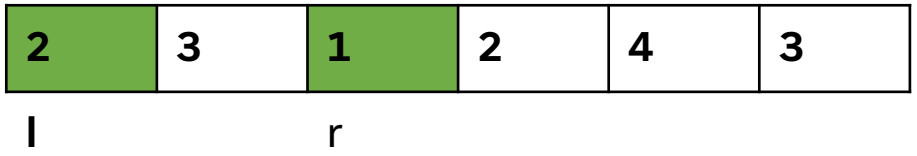
Target = 7



sum=5

Increase window size

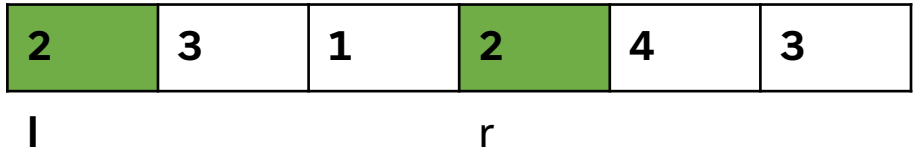
Target = 7



sum=6

Increase window size

Target = 7



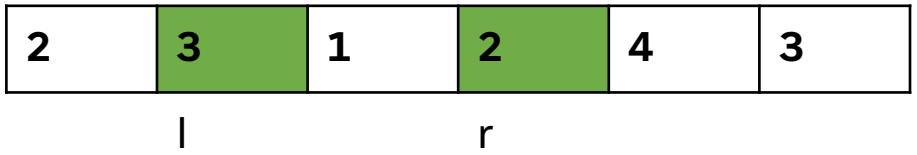
sum=8

Found first window with sum >= target

Minimum size window = 4

Move left pointer over by 1 and continue

Target = 7

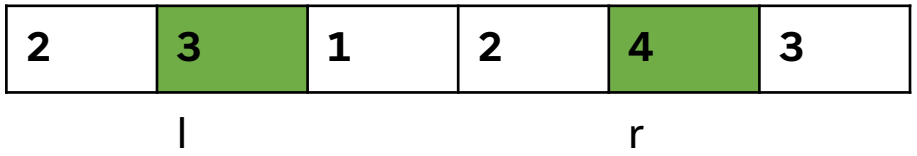


sum=6

Increase window size (move right pointer)

Minimum size window = 4

Target = 7



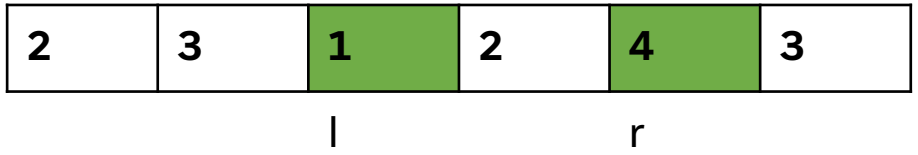
sum=10

Found second window with sum >= target

Minimum size window = 4

Move left pointer over by 1 and continue

Target = 7



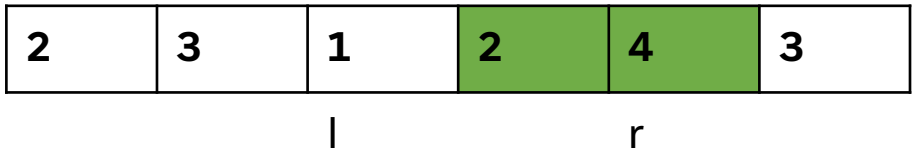
sum=7

Found third window with sum >= target

Minimum size window = 3

Move left pointer over by 1 and continue

Target = 7

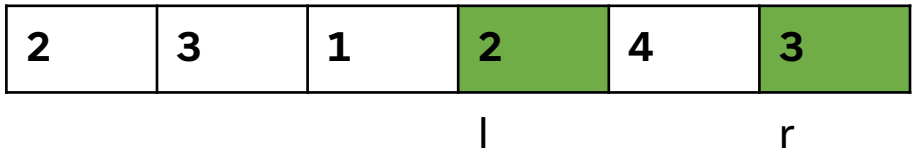


sum=6

Increase window size (move right pointer)

Minimum size window = 3

Target = 7



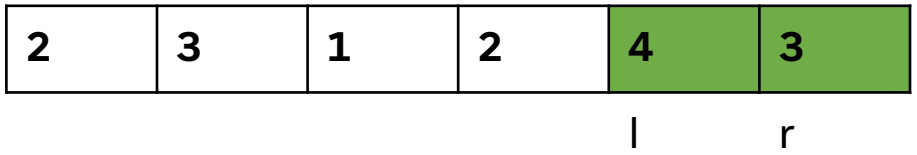
sum=9

Found fourth window with sum >= target

Minimum size window = 3

Move left pointer over by 1 and continue

Target = 7



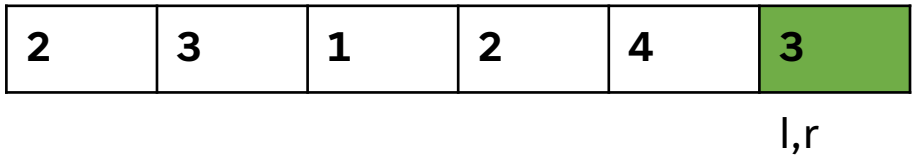
sum=7

Found fifth window with sum >= target

Minimum size window = 2

Move left pointer over by 1 and continue

Target = 7



sum=3

Minimum size window = 2

When to Use Sliding Window?

Running Values: Efficiently calculate moving values like sums or averages over a stream of data.

Adjacent Values: Process consecutive elements in an ordered structure.

Target Identification: Find specific values or subarrays meeting criteria.

Optimal Sequences: Identify longest, shortest, or most optimal sequences.

Summary

Sliding Window is a powerful technique for optimizing contiguous sequence problems.

Fixed windows are useful when the size is constant, while variable windows are flexible for dynamic conditions.

Mastering this technique helps in solving problems efficiently in competitive programming and technical interviews.