Ask a Question

Introduction to Sliding Window

Let’s go over the Sliding Window pattern, its real-world applications, and some problems we can solve with it.

We'll cover the following...

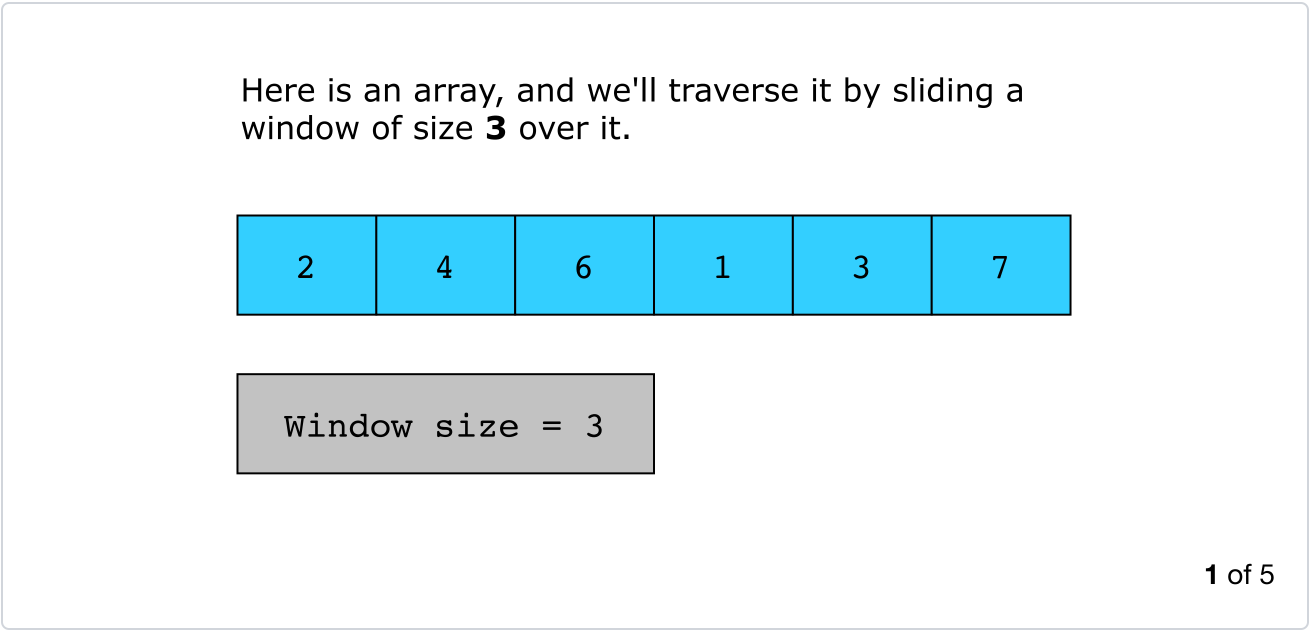
About the pattern

The **sliding window** pattern is used to process sequential data, arrays, and strings, for example, to efficiently solve subarray or substring problems. It involves maintaining a dynamic window that slides through the array or string, adjusting its boundaries as needed to track relevant elements or characters. The window is used to slide over the data in chunks corresponding to the window size, and this can be set according to the problem’s requirements. It may be viewed as a variation of the two pointers pattern, with the pointers being used to set the window bounds.

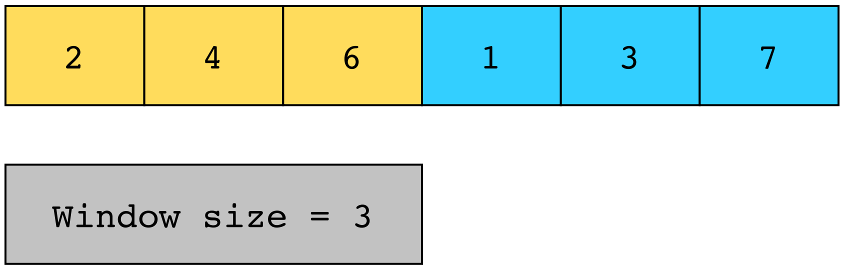
Imagine you’re in a long hallway lined with paintings, and you’re looking through a narrow frame that only reveals a portion of this hallway at any time. As you move the frame along the hallway, new paintings come into view while others leave the frame. This process of moving and adjusting what’s visible through the frame is akin to how the sliding window technique operates over data.

Why is this method more efficient? Consider we need to find k consecutive integers with the largest sum in an array. The time complexi of the naive solution to this problem would be $O(kn)$, because we nee to compute sums of all subarrays of size k. On the other hand, if we employ the sliding window pattern, instead of computing the sum of all elements in the window, we can just subtract the element exiting the window, add the element entering the window, and update the maximum sum accordingly. In this way, we can update the sums in constant time, yielding an overall time complexity of $O(n)$. To summarize, generally, the computations performed every time the window moves should take $O(1)$ time or a slow-growing function, such as the log of a small variable.

The following illustration shows a possibility of how a window could move along an array:

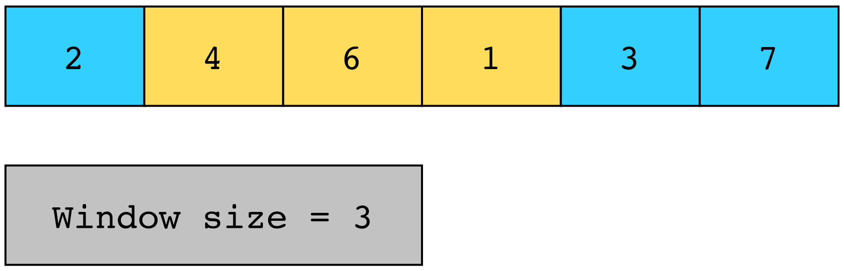

Here is an array, and we'll traverse it by sliding a window of size **3** over it.
2
4
6
1
3
7
Window size = 3
1 of 5

This is the first set of elements that will be considered by the sliding window.



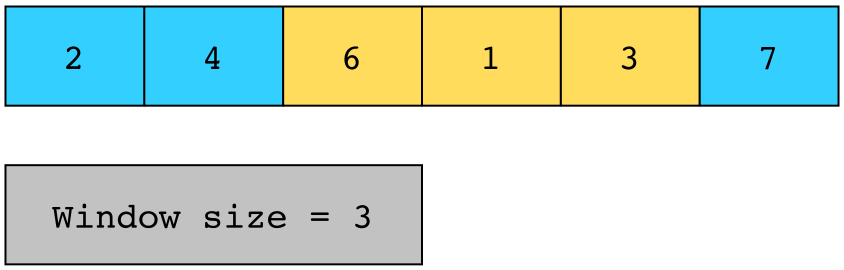
2 of 5

In the next step, the leftmost element of the window leaves it, and the next immediate element in the array enters the window.



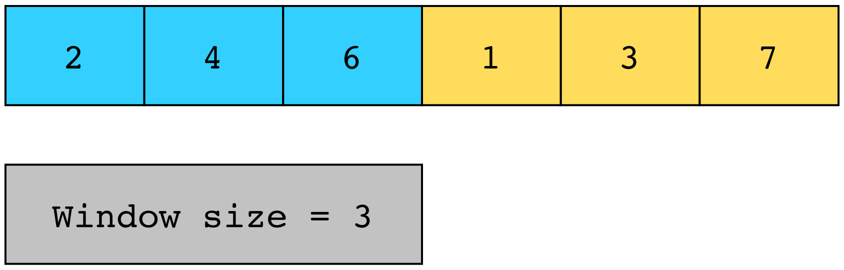
3 of 5

In the same manner, we move the window forward to consider the next group of **3** elements.



4 of 5

Finally, the window is moved to process the last **3** elements of the array.

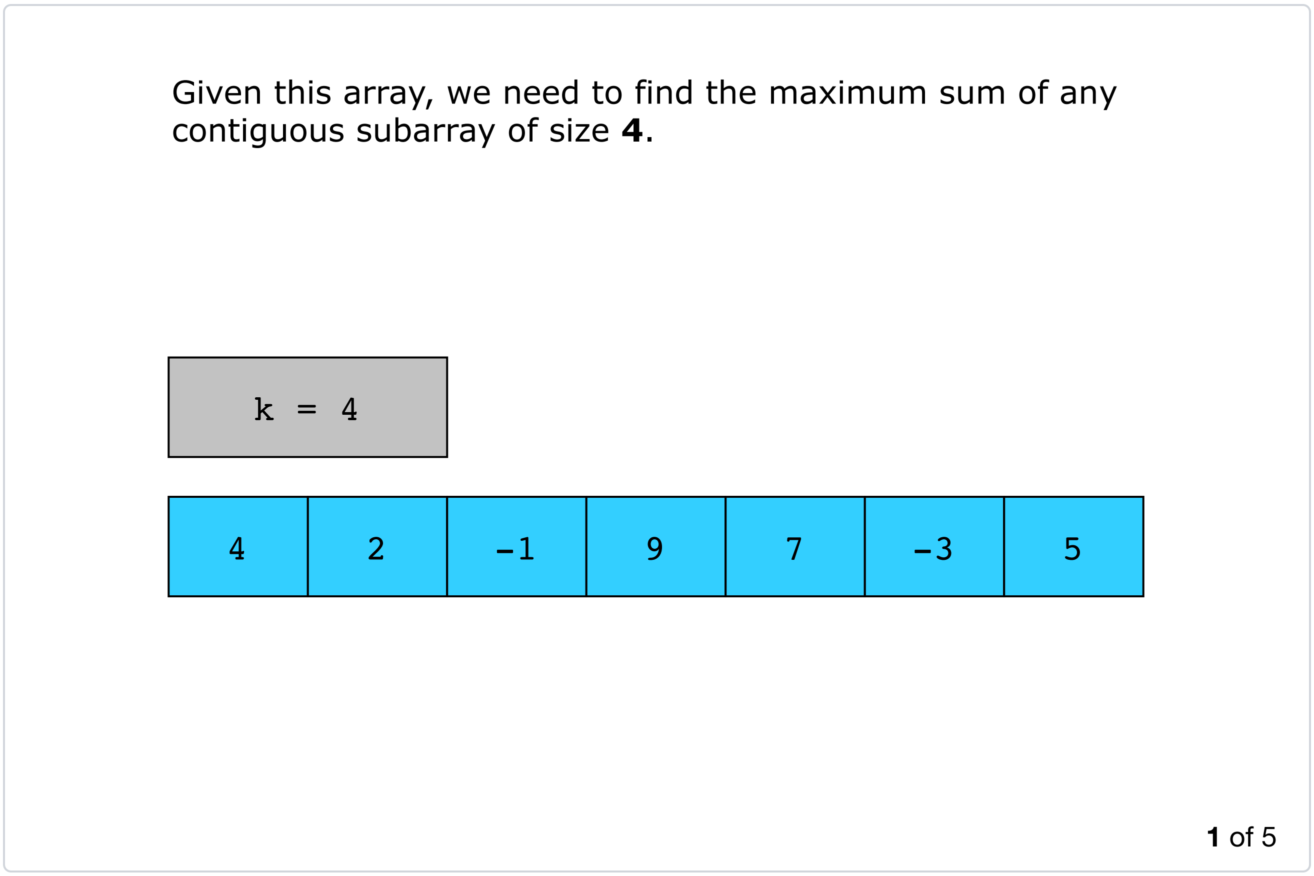


5 of 5

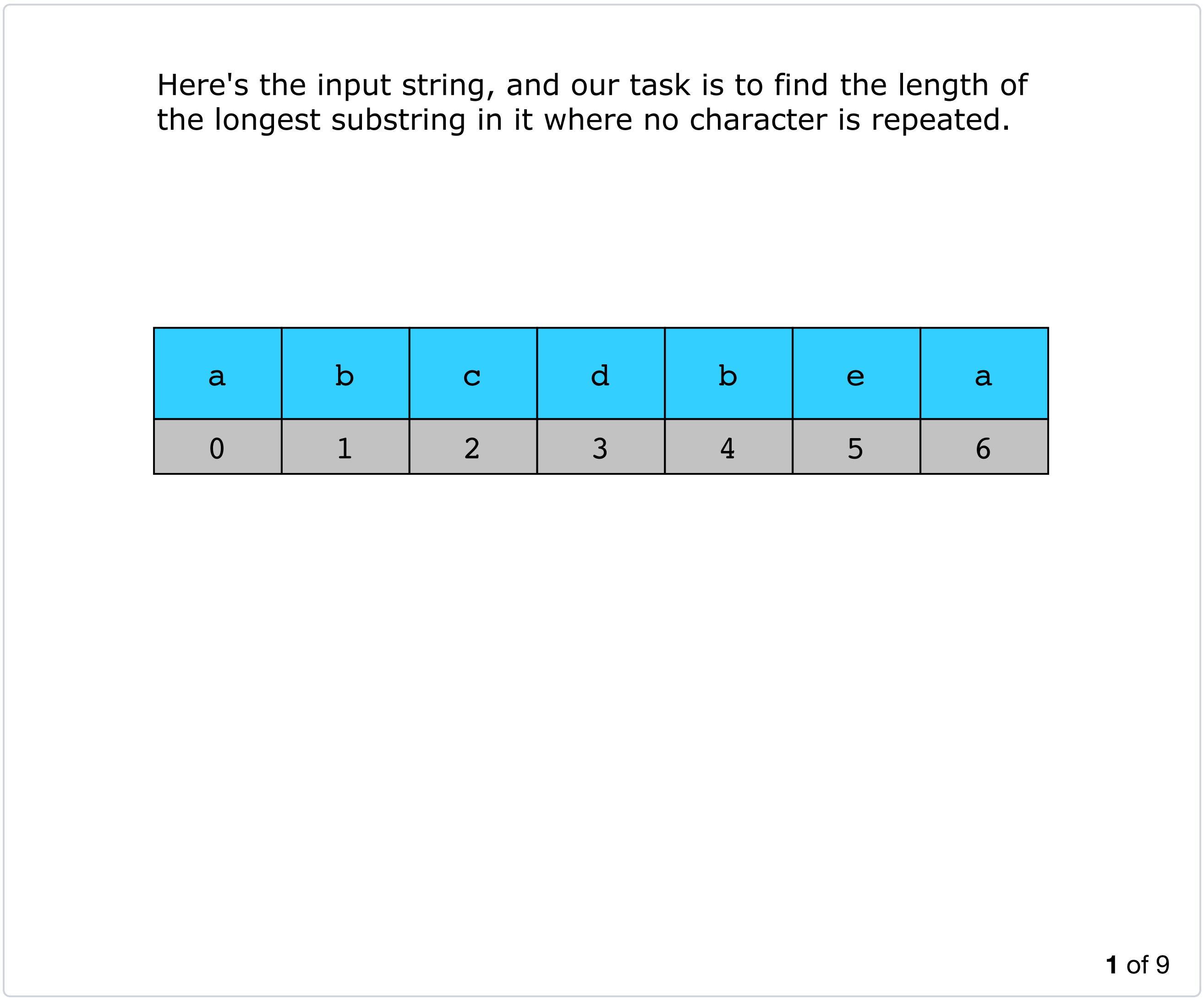
Examples

The following examples illustrate some problems that can be solved with this approach:

1. **Maximum sum subarray of size K:** Given an array of integers and a positive integer k, find the maximum sum of any contiguous subarray of size k.


Given this array, we need to find the maximum sum of any contiguous subarray of size **4**.

2. **Longest substring without repeating characters:** Given a string, find the length of the longest substring without repeating characters.



Here's the input string, and our task is to find the length of the longest substring in it where no character is repeated.

a	b	c	d	b	e	a
0	1	2	3	4	5	6

Does your problem match this pattern?

Yes, if all of these conditions are fulfilled:

- **Contiguous data:** The input data is stored in a contiguous manner, such as an array or string.

- **Processing subsets of elements:** The problem requires repeated computations on a contiguous subset of data elements (a subarray or a substring), such that the window moves across the input array from one end to the other. The size of the window may be fixed or variable, depending on the requirements of the problem.
- **Efficient computation time complexity:** The computations performed every time the window moves take constant or very small time.

Real-world problems

Many problems in the real world use the sliding window pattern. Let's look at some examples.

- **Telecommunications:** Find the maximum number of users connected to a cellular network's base station in every k-millisecond sliding window.
- **Video streaming:** Given a stream of numbers representing the number of buffering events in a given user session, calculate the median number of buffering events in each one-minute interval.
- **Social media content mining:** Given the lists of topics that two users have posted about, find the shortest sequence of posts by one user that includes all the topics that the other user has posted about.

Strategy time!

Match the problems that can be solved using the sliding window pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side

Locate the 6^{th} largest element in an array.	Sliding Window