Ask a Question

Solution: Valid Palindrome

Let's solve the Valid Palindrome problem using the Two Pointers pattern.

We'll cover the following

Statement

Write a function that takes a string, `s`, as an input and determines whether or not it is a palindrome.

> **Note**: A **palindrome** is a word, phrase, or sequence of characters that reads the same backward as forward.

Constraints:

- $1 \leq$ `s.length` $\leq 2 \times 10^5$
- The string `s` will not contain any white space and will only consist of ASCII characters(digits and letters).

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

The naive approach to solve this problem is to reverse the string and then compare the reversed string with the original string. If they match, the original string is a valid palindrome. Although this solution has a linear time complexity, it requires extra space to store the reversed string, making it less efficient in terms of space complexity. Therefore, we can use an optimized approach to save extra space.

Optimized approach using two pointers

A **palindrome** is a word or phrase that reads the same way when it is reversed. This means the characters at both ends of the word or phrase should be the same.

The essence of this solution is the efficient use of two pointers converging from opposite ends toward the middle of the string. One pointer begins from the start and moves forward, while the other starts from the end and moves backward in the string. As they converge toward each other, they compare characters at each step to identify any mismatch. The string is a palindrome if these pointers converge in the middle without discovering mismatching characters.

This two pointers approach would allow us to solve this problem in linear time without any additional space complexity or the use of built-in functions. This is because we'll traverse the array from the start and the end simultaneously to reach the middle of the string.