Valid Palindrome

Description:

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a **palindrome**, or false otherwise.

Example 1:

Input: s = "A man, a plan, a canal: Panama"

Output: true

Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: s = "race a car"

Output: false

Explanation: "raceacar" is not a palindrome.

Example 3:

Input: s = " "

Output: true

Explanation: s is an empty string "" after removing non-alphanumeric characters.

Since an empty string reads the same forward and backward, it is a palindrome.

Constraints:

- $1 <= s.length <= 2 * 10^5$
- s consists only of printable ASCII characters.

Algorithm:

1.) **Filter and Normalize the Input**: Create a new string string by iterating through each character in the input string s. Only include alphanumeric characters (letters and digits) and convert them to lowercase.

2.) **Initialize Pointers**: Set two pointers, left at the start (index 0) and right at the end (index len(string) - 1) of the string.

3.) **Check for Palindrome**: Enter a while loop that continues as long as left < right. At each iteration, compare the characters at the left and right pointers.

4.) **Mismatch Handling**: If the characters at string[left] and string[right] are not equal, return False as the string is not a palindrome.

5.) **Move Pointers**: If the characters match, increment the left pointer and decrement the right pointer to continue checking the next characters.

6.) **Return True**: If the loop completes without mismatches, return True, indicating the string is a palindrome.

Pseudocode:

```
function isPalindrome(s):
    string = empty string
    // Step 1: Filter and normalize input
    for each character in s:
        if character is alphanumeric:
            add character in lowercase to string
    // Step 2: Initialize pointers
    left = 0
    right = length of string - 1
    // Step 3: Check for palindrome
    while left < right:
        // Step 4: Mismatch handling
        if string[left] != string[right]:
            return False
        // Step 5: Move pointers
        left = left + 1
```

```
    right = right – 1
  // Step 6: Return True if palindrome
  return True
```

Code:

```
class Solution:
  def isPalindrome(self, s: str) -> bool:

    string = ''.join(char.lower() for char in s if char.isalnum())

    left, right = 0, len(string) - 1

    while left < right:
      if string[left] != string[right]:
        return False
      left += 1
      right -= 1

    return True
```

Conclusion

The algorithm efficiently checks if a given string is a palindrome by first normalizing the input string (removing non-alphanumeric characters and converting to lowercase) and then using a two-pointer approach to compare characters from both ends of the string. If any mismatch is found, the function returns False; otherwise, it returns True once all characters are verified. This approach runs in linear time, O(n), where n is the length of the input string, making it an optimal solution for palindrome checking.