🧩 Problem Understanding (LeetCode 125)

You are given a string s.

A string is a **palindrome** if:

- After converting all uppercase letters to lowercase
- Removing all **non-alphanumeric characters**
- It reads the **same forward and backward**

✅ **Valid characters:**

- Letters: a–z, A–Z
- Digits: 0–9

❌ **Ignore:**

- Spaces
- Punctuation
- Symbols

📌 **Example**

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

Output: true

Processed string: "amanaplanacanalpanama"

✅ Approach 1: StringBuilder Method

💡 **Idea**

1. Traverse the string
2. Keep **only alphanumeric characters**
3. Convert them to lowercase
4. Store them in a new string
5. Check whether this cleaned string is a palindrome

▯ **Step-by-Step Logic**

1. Create an empty StringBuilder
2. Loop through each character:

        - If it’s a letter or digit → add lowercase version

3. Convert builder to string
4. Use two pointers (or reverse) to check palindrome

⏱ **Complexity**

- **Time:** O(n)
- **Space:** O(n) (extra string)

☐ Java – StringBuilder Approach

```
class Solution {
  public boolean isPalindrome(String s) {
    StringBuilder sb = new StringBuilder();

    // Step 1: Filter and normalize
    for (char c : s.toCharArray()) {
      if (Character.isLetterOrDigit(c)) {
        sb.append(Character.toLowerCase(c));
      }
    }

    // Step 2: Check palindrome
    int left = 0, right = sb.length() - 1;
    while (left < right) {
      if (sb.charAt(left) != sb.charAt(right)) {
        return false;
      }
      left++;
      right--;
    }
    return true;
  }
}
```

□ C++ – StringBuilder (string) Approach

```
class Solution {
public:
    bool isPalindrome(string s) {
        string clean = "";

        // Step 1: Filter characters
        for (char c : s) {
            if (isalnum(c)) {
                clean += tolower(c);
            }
        }

        // Step 2: Check palindrome
        int left = 0, right = clean.size() - 1;
        while (left < right) {
            if (clean[left] != clean[right]) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
};
```

□ Python – StringBuilder (list) Approach

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

```
clean = []

# Step 1: Filter characters
for c in s:
    if c.isalnum():
        clean.append(c.lower())

# Step 2: Check palindrome
left, right = 0, len(clean) - 1
while left < right:
    if clean[left] != clean[right]:
        return False
    left += 1
    right -= 1

return True
```

✅ Approach 2: Two Pointer (Optimal – No Extra Space)

💡 Idea

Instead of creating a new string:

- Use **two pointers directly on the original string**
- Skip invalid characters on the fly
- Compare characters only when both are valid

▯ Step-by-Step Logic

1. Set left = 0, right = s.length - 1
2. While left < right:
    - Skip non-alphanumeric from left
    - Skip non-alphanumeric from right
    - Compare lowercase characters
3. If mismatch → return false

4. Else → move both pointers

⏱ **Complexity**

- **Time:** O(n)
- **Space:** O(1) ✅ BEST

□ Java – Two Pointer Approach

```java
class Solution {
    public boolean isPalindrome(String s) {
        int left = 0, right = s.length() - 1;

        while (left < right) {
            while (left < right && !Character.isLetterOrDigit(s.charAt(left))) {
                left++;
            }
            while (left < right && !Character.isLetterOrDigit(s.charAt(right))) {
                right--;
            }

            if (Character.toLowerCase(s.charAt(left)) !=
                Character.toLowerCase(s.charAt(right))) {
                return false;
            }

            left++;
            right--;
        }
        return true;
    }
}
```

□ C++ – Two Pointer Approach

```
class Solution {
public:
    bool isPalindrome(string s) {
        int left = 0, right = s.size() - 1;

        while (left < right) {
            while (left < right && !isalnum(s[left])) left++;
            while (left < right && !isalnum(s[right])) right--;

            if (tolower(s[left]) != tolower(s[right])) {
                return false;
            }

            left++;
            right--;
        }
        return true;
    }
};
```

□ Python – Two Pointer Approach

```
class Solution:
    def isPalindrome(self, s: str) -> bool:
        left, right = 0, len(s) - 1

        while left < right:
            while left < right and not s[left].isalnum():
                left += 1
            while left < right and not s[right].isalnum():
                right -= 1
            if s[left].lower() != s[right].lower():
                return False

            left += 1
            right -= 1

        return True
```

🔥 **Interview Summary (Must Say)**

Approach	**Extra Space**	**Preferred**
StringBuilder	O(n)	Easy to understand
Two Pointer	O(1)	✅ Best for interviews

📌 **Always prefer Two Pointer** in interviews unless asked otherwise.