LeetCode 5. Longest Palindromic Substring

1. Problem Title & Link

- **5. Longest Palindromic Substring**
- 🔗 https://leetcode.com/problems/longest-palindromic-substring/

2. Problem Statement (Short Summary)

Given a string s, return the **longest palindromic substring** in it.

A palindrome is a string that reads the same forward and backward (like “madam” or “abba”).

3. Examples (Input → Output)

Input: s = "babad"

Output: "bab"

Explanation: "aba" is also valid.

Input: s = "cbbd"

Output: "bb"

Input: s = "a"

Output: "a"

Input: s = "ac"

Output: "a"

4. Constraints

- 1 <= s.length <= 1000
- s consists of only digits and English letters.

5. Thought Process (Step by Step)

There are **two major approaches** here

🧠 Approach 1: Expand Around Center (Optimal for Interviews)

- Every palindrome has a **center** (can be 1 char or 2 chars).
- Expand outward while both sides match.
- Track the longest substring found.

For each index i:

- Expand with **odd center** → s[i] (for “aba”)
- Expand with **even center** → s[i] + s[i+1] (for “abba”)

🧮 Approach 2: Dynamic Programming (Tabulation)

- Let dp[i][j] = True if substring s[i..j] is palindrome.
- Base cases:
    - Single char → palindrome
    - Two equal chars → palindrome
- Transition:
  dp[i][j] = (s[i] == s[j]) and dp[i+1][j-1]
-
- Fill table bottom-up ($O(n^2)$ time, $O(n^2)$ space).
  ✅ Easier for visualization but slower and heavier in memory.

6. Pseudocode (Center Expansion)

```
for i in range(len(s)):
    expand_center(i, i)       # odd length
    expand_center(i, i + 1)   # even length

expand_center(l, r):
    while l >= 0 and r < n and s[l] == s[r]:
        if (r - l + 1) > max_len:
            start = l
            max_len = r - l + 1
        l -= 1
        r += 1
```

7. Code Implementation

✅ Python

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        start, max_len = 0, 0

        def expand(l, r):
            nonlocal start, max_len
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if r - l + 1 > max_len:
                    start = l
                    max_len = r - l + 1
                l -= 1
                r += 1
        for i in range(len(s)):
            expand(i, i)       # odd-length center
            expand(i, i + 1)   # even-length center

        return s[start:start + max_len]
```

✅ **Java**

```
class Solution {
    public String longestPalindrome(String s) {
        int start = 0, maxLen = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expand(s, i, i);     // odd
            int len2 = expand(s, i, i + 1); // even
            int len = Math.max(len1, len2);
            if (len > maxLen) {
                start = i - (len - 1) / 2;
                maxLen = len;
            }
        }
        return s.substring(start, start + maxLen);
    }

    private int expand(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) ==
s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
}
```

8. Time & Space Complexity

- **Time:** $O(n^2)$ — two expansions per center
- **Space:** $O(1)$ — no extra storage

(DP approach → $O(n^2)$ time & space)

9. Dry Run (Step-by-Step Execution)

👉 Input: s = "babad"

Center	Left	Right	Palindrome	Longest
0	b	b	"b"	"b"
1	a	a	"aba"	"aba"
2	b	b	"bab"	"bab"
3	a	d	❌	"bab"
Even centers	check “bb”, “aa”, “ad”	none	"bab"	

✅ Output: "bab" (or "aba")

10. Visual Insight (for teaching ❤️)

String: b a b a d

Centers: ↑ ↑ ↑ ↑ ↑

↖↘ ↖↘ ↖↘ ↖↘ ↖↘

Expand outward until mismatch

Track the longest mirrored span

11. Concept Insight Table

Core Concept	Common Use	Common Traps	Builds / Next Steps
Expand Around Center — every palindrome mirrors around its center.	- Palindromic substrings - Mirror-based problems - Symmetry-based	- Forgetting even-length centers - Expanding out of bounds - Not updating max length properly	🔷 Builds to **LC 647 (Count Palindromic Substrings)** 🔷 Leads into **DP table optimizations** 🔷 Foundation

12. Common Mistakes / Edge Cases

- Ignoring even-length centers (“abba” case).
- Returning the wrong substring slice (start:start+max_len).
- Empty string edge case → should return "".

13. Variations / Follow-Ups

- **LC 647:** Count all palindromic substrings.
- **LC 516:** Longest Palindromic Subsequence (DP-based).
- **Manacher’s Algorithm:** Advanced O(n) center expansion version.