

■ Check Palindrome String Using Recursion

Same letters from both ends — the mirror test

● Recursion

■ EASY

■ What You Will Learn

- Applying the 'shrinking boundaries' pattern to strings
- How recursion naturally checks symmetry
- Base cases for even and odd length strings
- Why this is more intuitive than iterative for beginners

■ Intuition First

'racecar' — is it a palindrome? Check first and last letters: $r==r$ ✓. Now check the inner string 'aceca': $a==a$ ✓. Inner 'cec': $c==c$ ✓. Inner 'e': single letter — always a palindrome! Each step, you peel one letter from each end and check if the inside is still a palindrome.

■ Recursive Palindrome Logic

`isPalin(s, left, right)`: if `s[left] != s[right]` → return false. Otherwise return `isPalin(s, left+1, right-1)`. Base case: if `left >= right` → return true (empty or single char is always palindrome).

■ Visual — 'racecar'

```
'r a c e c a r'
0 1 2 3 4 5 6
L           R
```

```
Step 1: s[0]='r' == s[6]='r' ✓ → check inside
Step 2: s[1]='a' == s[5]='a' ✓ → check inside
Step 3: s[2]='c' == s[4]='c' ✓ → check inside
Step 4: L=3, R=3 → L >= R → return true ✓
```

```
'r a c e b a r' (NOT palindrome)
Step 1: r==r ✓
Step 2: a==a ✓
Step 3: c != b ✗ → return false immediately
```

■ Java Code

```
1 class Solution {
2     public boolean isPalindrome(String s, int left, int right) {
```

```

3 // BASE CASE: empty middle or single char
4 if (left >= right) return true;
5
6 // If outer chars don't match → not palindrome
7 if (s.charAt(left) != s.charAt(right)) return false;
8
9 // Check if the inner substring is palindrome
10 return isPalindrome(s, left + 1, right - 1);
11 }
12
13 // Wrapper call:
14 // isPalindrome(s, 0, s.length()-1)
15 }

```

■ Time & Space Complexity

Type	Value	What It Means
Time	O(n)	Check n/2 pairs — each char visited once
Space	O(n)	Recursion stack depth up to n/2

■ Common Mistakes

■ Mistake

Returning false before checking the recursive call. The function must first verify the outer chars, THEN return the result of the inner check — not both simultaneously.

■ Pro Tips

■ Early Exit

If `s.charAt(left) != s.charAt(right)`, you can immediately return false without checking anything else. This makes it fast in practice — mismatches often appear early.

■ Practice Problems

#	Problem	Diff
1	Valid Palindrome (LeetCode 125)	Easy
2	Palindrome Number (LeetCode 9)	Easy
3	Check Palindrome Recursively	Easy
4	Palindromic Substrings (LeetCode 647)	Medium
5	Longest Palindromic Substring (LeetCode 5)	Medium

■ Quick Revision

✓	Base: $\text{left} \geq \text{right} \rightarrow \text{return true}$
✓	Mismatch: $s[\text{left}] \neq s[\text{right}] \rightarrow \text{return false}$
✓	Recurse: $\text{isPalin}(s, \text{left}+1, \text{right}-1)$
✓	Call with $\text{left}=0, \text{right}=s.\text{length}()-1$
✓	Time $O(n)$, Space $O(n/2)$ stack

■ Visual Thinking

Imagine folding the string in half like a piece of paper. If it's a palindrome, every character on the left lines up perfectly with its mirror on the right. Recursion is literally the act of folding: peel the outermost layer, check if they match, fold again. If any layer doesn't match, the paper tears — not a palindrome.