Leetcode 5.
Longest Palindromic Substring

#String #Dynamic Programming

* Problem Description

Given a string s, return the longest palindromic substring in s

[DEF] A string is **palindrome** if it reads the same forward and backward.

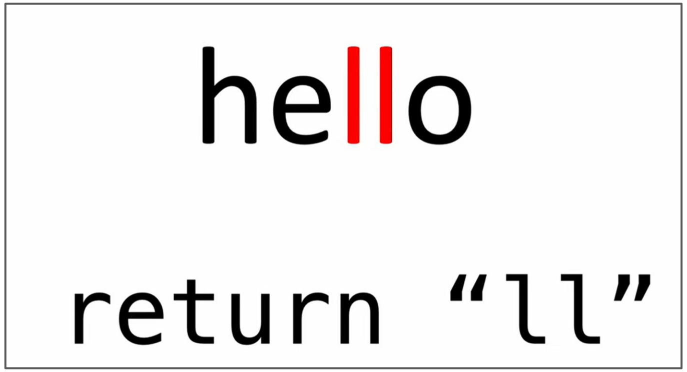
Example 1:

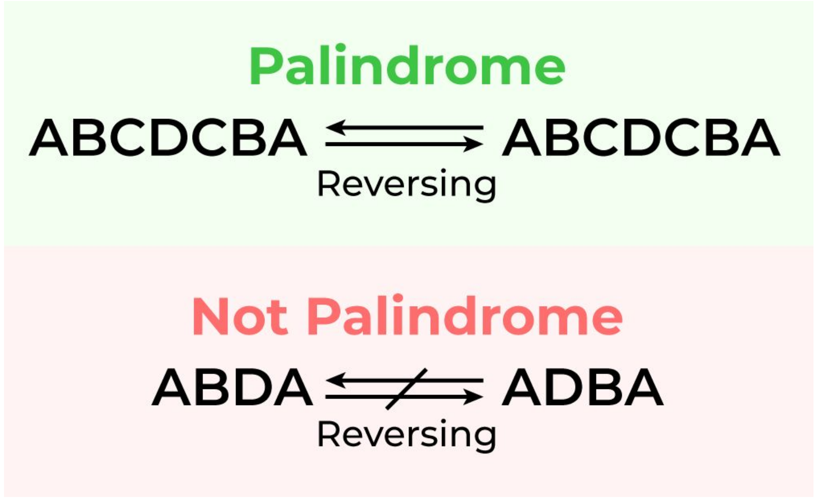
Input: s = "babad"
Output: "bab"

Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"
Output: "bb"



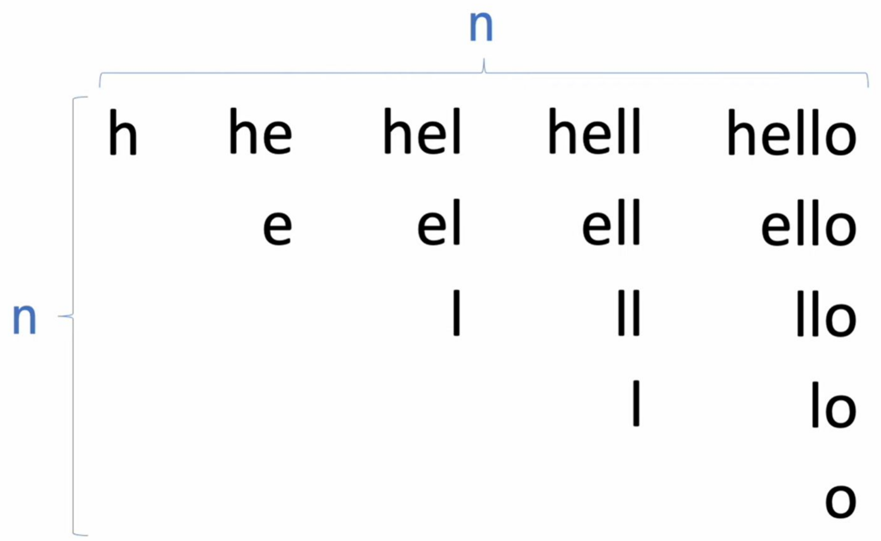


1 Brute Force Solution

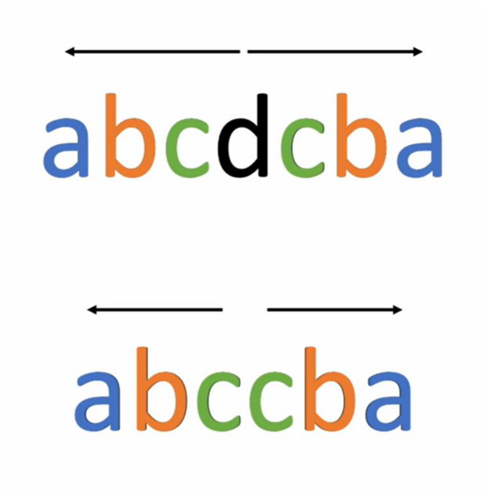
Step 1. Calculate the length of every possible substring

hello

return “ll”



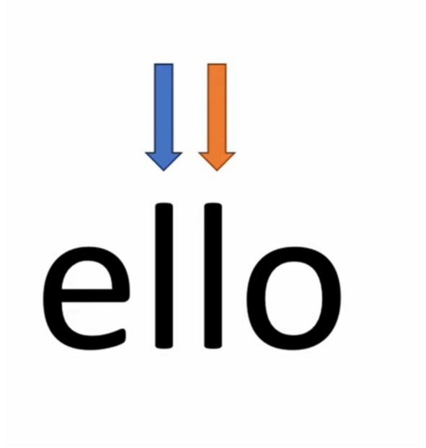
1 Brute Force Solution

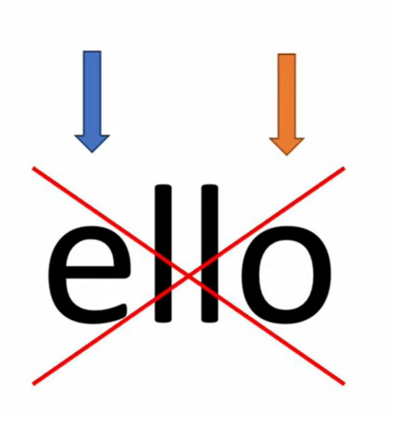


Step 2. Check if the substring is palindrome.

How?

→ Check the center of substring and work our way outwards, checking at each step to see if the characters match.





1 Brute Force Solution

Step 1. Calculate the length of every possible substring

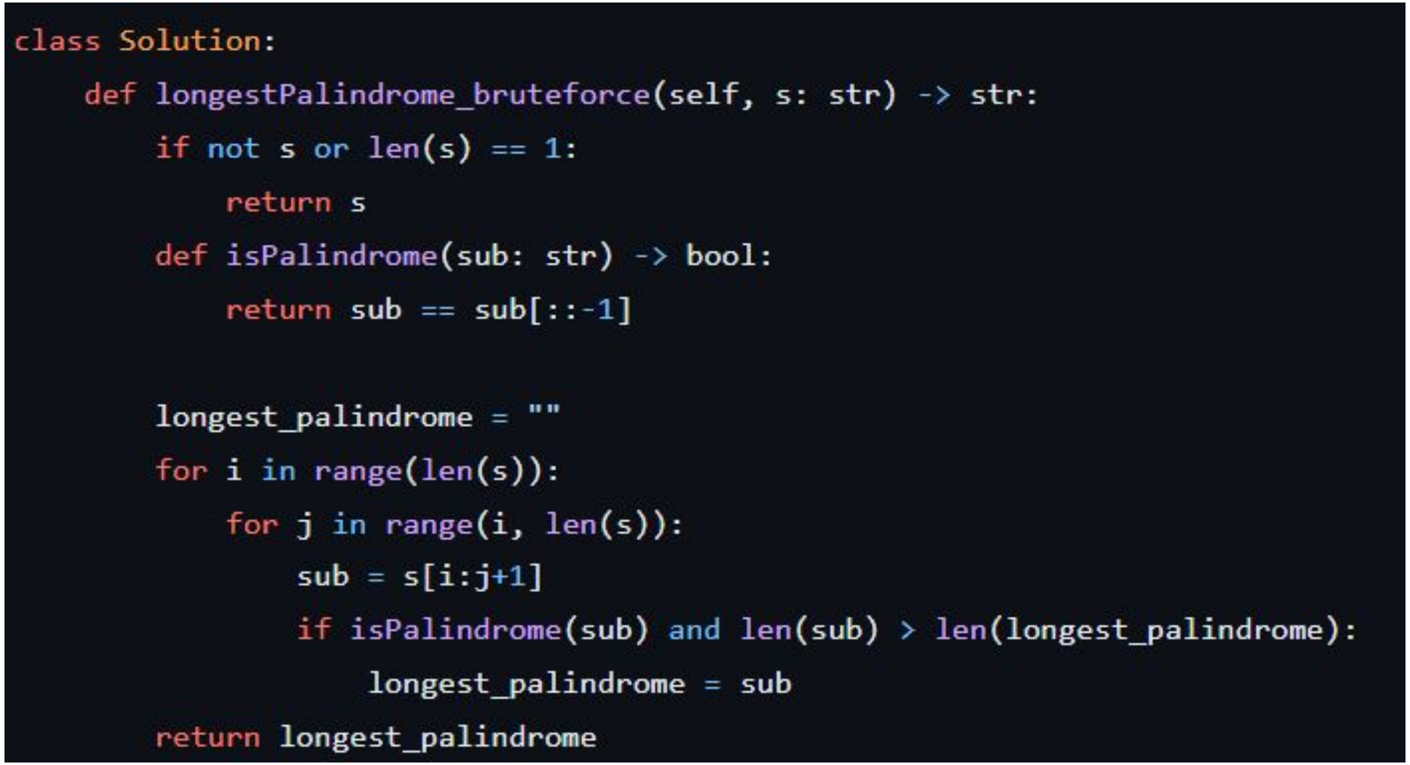
Step 2. Check if the substring is palindrome.

Time Complexity?

- Each check runs in O(n) time
- Must do this for each substring generated -> O(n^2) time

$\rightarrow$ **O(n^3) time!**

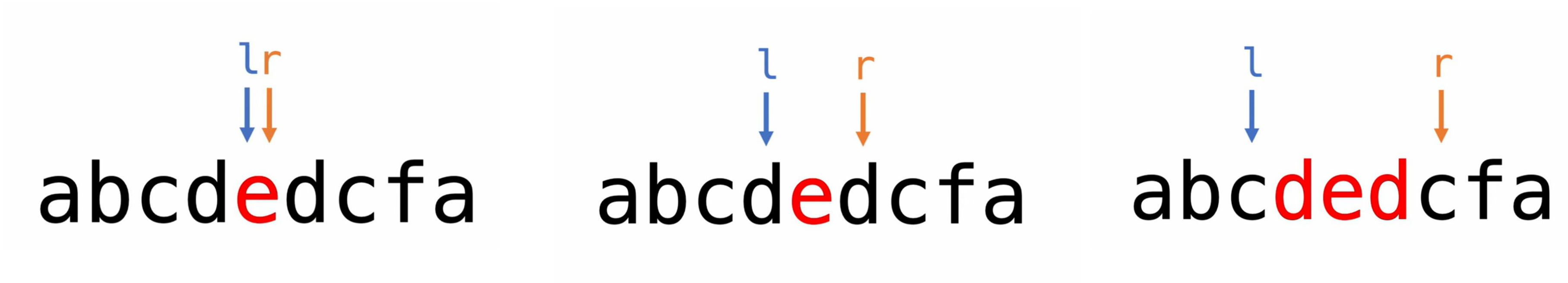
Code Solution



```python
class Solution:
    def longestPalindrome_bruteforce(self, s: str) -> str:
        if not s or len(s) == 1:
            return s
        def isPalindrome(sub: str) -> bool:
            return sub == sub[::-1]

        longest_palindrome = ""
        for i in range(len(s)):
            for j in range(i, len(s)):
                sub = s[i:j+1]
                if isPalindrome(sub) and len(sub) > len(longest_palindrome):
                    longest_palindrome = sub
        return longest_palindrome
```

2 Improved Solution with Time Complexity of O(n)



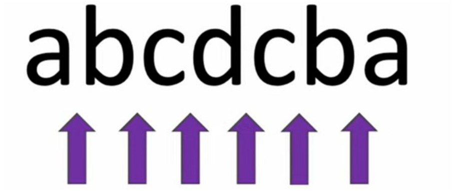
Key point: We can check ALL substrings with the SAME center in a single pass

Visiting every center: O(n)
Checking palindrome O(n)
-> Overall **O(n^2) time complexity.**

2 Improved Solution with Time Complexity of O(n)

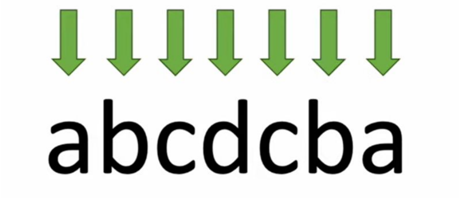
Q. How many different centers could the string have?

If len(s) is even:



The center could lie between any 2 characters

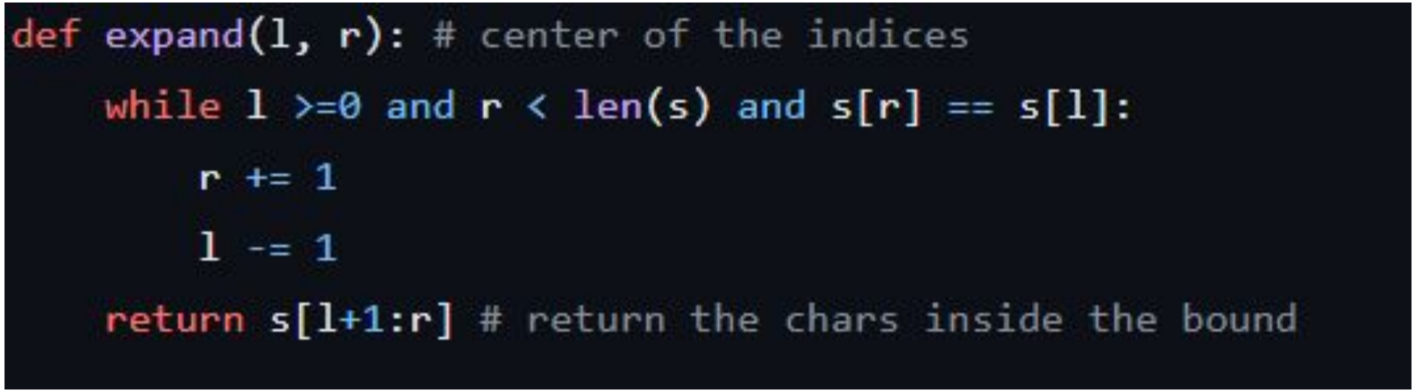
If len(s) is odd:



Any of the characters could be the center

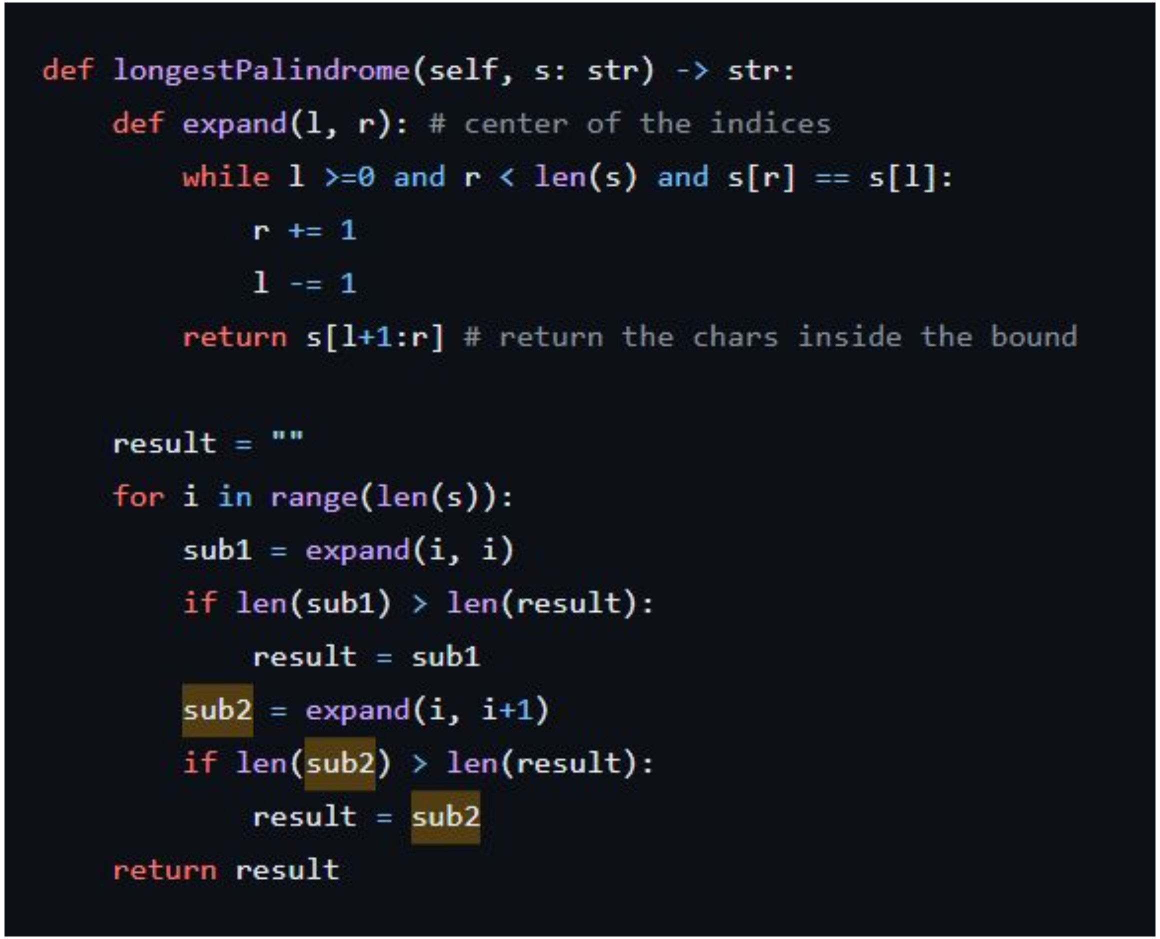
2

Improved Solution with Time Complexity of O(n)



```
def expand(l, r): # center of the indices
    while l >=0 and r < len(s) and s[r] == s[l]:
        r += 1
        l -= 1
    return s[l+1:r] # return the chars inside the bound
```

Code Solution

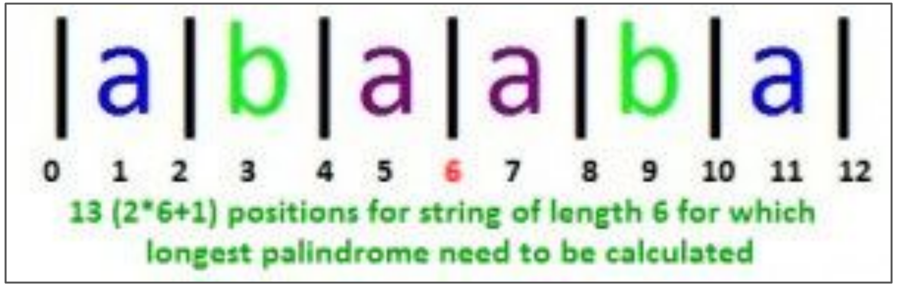


```
def longestPalindrome(self, s: str) -> str:
    def expand(l, r): # center of the indices
        while l >=0 and r < len(s) and s[r] == s[l]:
            r += 1
            l -= 1
        return s[l+1:r] # return the chars inside the bound

    result = ""
    for i in range(len(s)):
        sub1 = expand(i, i)
        if len(sub1) > len(result):
            result = sub1
        sub2 = expand(i, i+1)
        if len(sub2) > len(result):
            result = sub2
    return result
```

3 Manacher's Algorithm

Finding all sub-palindromes in **O(N)**



|a|b|a|a|b|a|
0 1 2 3 4 5 6 7 8 9 10 11 12
13 (2*6+1) positions for string of length 6 for which longest palindrome need to be calculated

|a|b|a|a|b|a|
0 1 2 3 4 5 6 7 8 9 10 11 12
13 (2*6+1) positions for string of length 6 for which longest palindrome need to be calculated

Key IDEA:

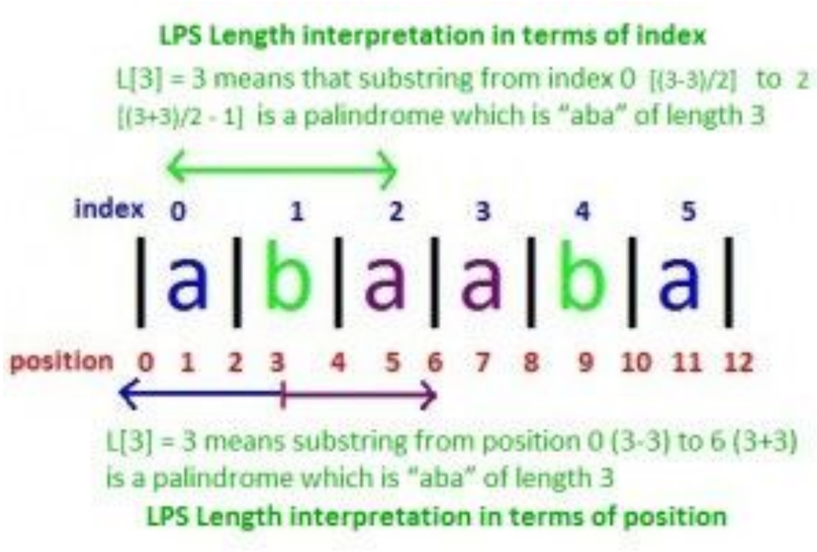
Note that left and right side of the center positions are symmetric. (bc the string is palindromic)
We need to calculate at each longest palindromic substring at 2*N + 1pos from left to right

If there is a palindrome of some length L centered at any position P, then we may not need to compare all characters in left and right side at position P+1

We already calculated LPS at positions before P and they can help to avoid some of the comparisons after position P.

3 Manacher's Algorithm

Finding all sub-palindromes in **O(N)**


LPS Length interpretation in terms of index

L[3] = 3 means that substring from index 0 [(3-3)/2] to 2 [(3+3)/2 - 1] is a palindrome which is "aba" of length 3

L[3] = 3 means substring from position 0 (3-3) to 6 (3+3) is a palindrome which is "aba" of length 3

LPS Length interpretation in terms of position

$$\dots\ \overbrace{s_{l+1}\ \dots\ \underbrace{s_{j-d_{odd}[j]+1}\ \dots\ s_j\ \dots\ s_{j+d_{odd}[j]-1}}_{\text{palindrome}}\ \dots\ \underbrace{s_{i-d_{odd}[j]+1}\ \dots\ s_i\ \dots\ s_{i+d_{odd}[j]-1}}_{\text{palindrome}}\ \dots\ s_{r-1}}^{\text{palindrome}}\ \dots$$

감사합니다!

THANK YOU