Longest Palindromic Substring

Input: *str = “forgeeksskeegfor”*

Output: *“geeksskeeg”*

Explanation: *There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.*

Input: *str = “Geeks”*

Output: *“ee”*

Naive Approach for Longest Palindromic Substring:

Check each substring, if it is a palindrome or not and keep updating the longest palindromic substring found till now.

Follow the steps mentioned below to implement the idea:

- Generate all the substrings.
    - For each substring, check if it is palindrome or not.
    - If substring is Palindrome, then update the result on the basis of longest palindromic substring found till now.

```
#include <bits/stdc++.h>

using namespace std;

// Function to print a substring str[low..high]

void printSubStr(string str, int low, int high)

{

        for (int i = low; i <= high; ++i)

                cout << str[i];

}

// This function prints the longest palindrome substring

// It also returns the length of the longest palindrome

int longestPalSubstr(string str)

{

        // Get length of input string

        int n = str.size();

        // All substrings of length 1 are palindromes

        int maxLength = 1, start = 0;
```

```
    // Nested loop to mark start and end index
    for (int i = 0; i < str.length(); i++) {
        for (int j = i; j < str.length(); j++) {
            int flag = 1;

            // Check palindrome
            for (int k = 0; k < (j - i + 1) / 2; k++)
                if (str[i + k] != str[j - k])
                    flag = 0;

            // Palindrome
            if (flag && (j - i + 1) > maxLength) {
                start = i;
                maxLength = j - i + 1;
            }
        }
    }

    cout << "Longest palindrome substring is: ";
    printSubStr(str, start, start + maxLength - 1);

    // Return length of LPS
    return maxLength;
}

// Driver Code
int main()
{
    string str = "forgeeksskeegfor";
```

```
        cout << "\nLength is: " << longestPalSubstr(str);

        return 0;

}
```

Complexity Analysis:

- **Time complexity:** O(N3). Three nested loops are needed to find the longest palindromic substring in this approach.
- **Auxiliary complexity**: O(1). As no extra space is needed.

Approach 2:

The LPS is either of even length or odd length. So the idea is to traverse the input string and for each character check if this character can be the center of a palindromic substring of odd length or even length.

Follow the steps mentioned below to implement the idea:

- Use two pointers, **low** and **hi**, for the left and right end of the current palindromic substring being found.
- Then checks if the characters at **str[low]** and **str[hi]** are the same.
    - If they are, it expands the substring to the left and right by decrementing **low** and incrementing **hi**.
    - It continues this process until the characters at **str[low]** and **str[hi]** are unequal or until the indices are in bounds.
- If the length of the current palindromic substring becomes greater than the maximum length, it updates the maximum length.

```
// C++ code to implement the above idea


#include <bits/stdc++.h>

using namespace std;


// Function to print a substring str[low..high]

void printSubStr(string str, int low, int high)

{

        for (int i = low; i <= high; ++i)
```

```
                cout << str[i];
}

// Function to find the longest palindromic substring
int longestPalSubstr(string s)
{
        int n = s.length();
        int start = 0, end = 1;
        int low, hi;

        // Traverse the input string
        for (int i = 0; i < n; i++) {

                // Find longest palindromic substring of even size
                low = i - 1;
                hi = i;

                // Expand substring while it is palindrome
                // and in bounds
                while (low >= 0 && hi < n && s[low] == s[hi]) {

                        // Update maximum length and starting index
                        if (hi - low + 1 > end) {
                                start = low;
                                end = hi - low + 1;
                        }
                        low--;
                        hi++;
                }
        // Find longest palindromic substring of odd size
        low = i - 1;
        hi = i + 1;

        // Expand substring while it is palindrome
        // and in bounds
        while (low >= 0 && hi < n && s[low] == s[hi]) {

            // Update maximum length and starting index
            if (hi - low + 1 > end) {
                start = low;
                end = hi - low + 1;

            }
            low--;
            hi++;
        }
    }

    // Print the longest palindromic substring
    cout << "Longest palindrome substring is: ";
    printSubStr(s, start, start + end - 1);

    // Return output length
    return end;
}

// Driver code
```

```
int main()

{

        string str = "forgeeksskeegfor";


        // Function call

        cout << "\nLength is: " << longestPalSubstr(str);

        return 0;

}
```

Output

```
Longest palindrome substring is: geeksskeeg

Length is: 10
```

Time complexity: O(N^2), where N is the length of the input string
Auxiliary Space: O(1), No extra space used.