palindrome.

Correct Approach:

We could see that the longest common substring method fails when there exists a reversed copy of a non-palindromic substring in some other part of S. To rectify this, each time we find a longest common substring candidate, we check if the substring's indices are the same as the reversed substring's original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.

This gives us an O(n^2) Dynamic Programming solution which uses O(n^2) space (could be improved to use O(n) space).

Dyanamic programming Approach: Dynamic programming solution is already discussed here in the previous post. The time complexity of the Dynamic Programming based solution is O(n^2) and it requires O(n^2) extra space. We can find the longest palindrome substring(LPS) in (n^2) time with O(1) extra space.

The algorithm below is very simple and easy to understand. The idea is to Fix a center and expand in both directions for longer palindromes and keep track of the longest palindrome seen so far.

ALGO:

1. Maintain a variable ' *maxLength = 1* ' (for storing LPS length) and ' start =0 ' (for storing starting index of LPS).
2. The idea is very simple, we will traverse through the entire string with i=0 to i<(length of string).
    1. while traversing, initialize 'low 'and 'high 'pointer such that low= i-1 and high=

    4. finally we will keep incrementing 'high' and decrementing 'low' until *str[low]==str[high]*.
    5. calculate length=high-low-1, if length > maxLength then maxLength = length and start = low+1 .
3. Print the LPS and return maxLength.

```
    string str = "forgeeksskeegfor";
    cout << "\nLength is: " << longestPalSubstr(str)
         << endl;
    return 0;
}

// This is code is contributed by saurabh yadav
```

C

```
// A O(n^2) time and O(1) space
// program to find the longest
// palindromic substring
#include <stdio.h>
#include <string.h>

// A utility function to print
// a substring str[low..high]
void printSubStr(char* str, int low, int high)
{
    for (int i = low; i <= high; ++i)
        printf("%c", str[i]);
}

// This function prints the longest
// palindrome substring (LPS)
// of str[]. It also returns the
// length of the longest palindrome
int longestPalSubstr(char* str)
{
    int n = strlen(str); // calculating size of string
    if (n < 2)
        return n; // if string is empty then size will be 0.
                  // if n==1 then, answer will be 1(single
                  // character will always palindrome)

    int maxLength = 1,start=0;
    int low, high;
    for (int i = 0; i < n; i++) {
        low = i - 1;
        high = i + 1;
        while ( high < n && str[high] == str[i]) //increment 'high'
            high++;

        while ( low >= 0 && str[low] == str[i]) // decrement 'low'
            low--;
        }

        int length = high - low - 1;
        if (maxLength < length) {
            maxLength = length;
              start=low+1;
        }
    }
    printf("Longest palindrome substring is: ");
      printSubStr(str,start,start+maxLength-1);
    return maxLength;
}

// Driver program to test above functions
int main()
{
    char str[] = "forgeeksskeegfor";
    printf("\nLength is: %d", longestPalSubstr(str));
    return 0;
}
// This is code is contributed by saurabh yadav
```

Java

```
// Java implementation of O(n^2)
// time and O(1) space method
// to find the longest palindromic substring
public class LongestPalinSubstring {

    // This function prints the
    // longest palindrome substring
    // (LPS) of str[]. It also
    // returns the length of the
    // longest palindrome
    static int longestPalSubstr(String str)
    {
        int n = str.length(); // calculcharAting size of string
        if (n < 2)
            return n; // if string is empty then size will be 0.
                  // if n==1 then, answer will be 1(single
                  // character will always palindrome)

        int maxLength = 1,start=0;
        int low, high;
        for (int i = 0; i < n; i++) {
            low = i - 1;
            high = i + 1;
```