Longest Palindromic Substring

Problem Statement: Given a string STR, we are required to find the longest palindromic substring. A **substring** is a contiguous sequence of characters within a string. For example, in the string "minor": "in", "ino", "min",...etc are substrings, but not "mr". Also, a palindrome is a word that reads the same backward as forward. Examples include abba, aaaa, hannah. If there exist multiple such strings we return the substring with the minimum starting index.

Example:

Given String STR is abccbc.
Then the substring bccb is the longest palindromic substring of the given string.
For string "abccbc" there are multiple palindromic substrings like "a", "b", "c", "cc", "bccb", "cbc". But "bccb" is of the longest length. Thus, the answer is "bccb".

Brute Force

- Generate substrings of the given string such that substring having greater length will be generated first.
- To do this, run a loop where the iterator 'LEN' will go from N to 1, where N is the length of the given string.
- Run a nested loop and fix an iterator 'j' that will point at the starting index of the substring.
- Get the substring from 'j' to 'j' + 'LEN'.
- If the substring is a palindrome, return the substring (as the substring will be of the longest length and minimum starting index).

```
/*
        function to check if the given string sub is a palindrome or not
*/
function isPalindrome(sub)
        n = len(sub)
        i = 0
        j = n - 1
        while i < j
                if(sub[i] != sub[j])
                        return False

                i += 1
                j -= 1

        return True
```

```
/*
        Function to find the longest palindromic substring of the given string
*/

function longestPalinSubstring(str)
        n = len(str)
        if n < 1
                return ""

        //   Start finding palindromes having maximum length.
        for length in range(n, 0, -1)
                for j in range(n - length + 1)

                        //   Get the input starting from j and having length 'length'.
                        input = str[j  j + length]
                        if (isPalindrome(input))
                                return input

        return ""
```

Time Complexity : O(N ^ 3), where N is the length of the given string. We are creating every substring which takes N ^ 2 time. Checking whether the substring is palindromic takes O(length of substring) time, which can be maximum O(N). Thus, the total time complexity is O(N ^ 3).

Space Complexity : O(N), where N is the length of the given string. We are storing substring in a variable string.