[B - Largest Palindromic Substring](https://vjudge.net/problem/HackerRank-si-largest-palindromic-substring" \t "_blank)

Given a string, find the length of the largest palindromic substring.

**Input Format**

First line of input contains T - number of test cases. It’s followed by 2T lines. The first line contains N - the size of the string and the second line contains a string of size N, containing only lowercase English alphabets.

**Constraints**

30 points  
1 <= T <= 200  
1 <= len(S) <= 102  
'a' <= S[i] <= 'z'

70 points  
1 <= T <= 200  
1 <= len(S) <= 103  
'a' <= S[i] <= 'z'

**Output Format**

For each test case, print the length of the largest palindromic substring, separated by newline.

**Sample Input 0**

5

8

pfyafafd

9

sllwffoqq

6

yoogvb

4

hcch

23

mzmqnnrkurfmmfrukrnnqsm

**Sample Output 0**

3

2

2

4

18

Brute Force

Generate every substring of EVEN as well as ODD length and check if it is palindrome or not.  If it is palindrome and longer than previous, this is the answer.

bool isPalindrome(string s, int startIdx, int endIdx)  
    // easy to write in (endIdx - startIdx + 1) complexity  
  
int longestPalindromicSubstringLength(string s)  
    int n = s.length();  
    int longestAns = 0;  
    **for startIdx = [0, n-1]  
        for endIdx = [startIdx, n-1]**  
            if(**isPalindrome**(s, startIdx, endIdx))  
                longestAns = max(longestAns, endIdx - startIdx + 1);  
    return longestAns;

Time Complexity:  for i = 0, inner loop runs n times.   for i = 1, inner loop runs (n-1) times and so on.....

Time Complexity:  n(n+1)/2 \* [endIdx-startIdx+1] = O(n3)

Space Complexity:  O(1)

Verdict:  TLE

Expand Around Centers

Using two pointers technique,

**for ODD length,**

**first use 0 as center index,** left = 0, right = 0.  At each step if s[left] == s[right] keep expanding the range of left to right.

then use 1 as center index, left = 1, right = 1, and continue to expand

then use index 2 as center index, left = 2, right = 2 and continue to expand as long as characters s[left]==s[right]

...

then use index  s.length()-1 as center index, left = s.length()-1, right = s.length()-1 and continue to expand.

**for EVEN length,**

**first use [0, 1] as center two indices**, left = 0, right = 1.  At each step if s[left]==s[right] keep expanding the range of left and right.

then use [1, 2] as center two indices

then use [2, 3] as center two indices

...

then use [s.length()-2, s.length()-1] s center two indices and keep expanding.

We can manage expansion for both ODD and EVEN length with a single function

int **longestPalindromeSubstringLengthByExpandingAroundCenter**(string originalString, int l, int r)  
     int n = originalString.length();  
      // Keep moving l left and r right (expanding)  
      while (l >= 0 && r <= n-1 && originalString[l] == originalString[r]) {  
        l--;  
        r++;  
      }  
      return r-l-1; **// substring of length r-l-1**

Now use above function for both even and odd lengths to get the answer.

int longestPalindrome(string s)  
      int n = s.length();  
      if (n == 0) return 0;  
      int **longestLength** = 1; // a single char itself is a palindrome  
      for i = [0, n-1)  
        int oddLength = **longestPalindromeSubstringLengthByExpandingAroundCenter**(s, i, i);  
        **longestLength** = max(longestLength, oddLength);  
     int evenLength = **longestPalindromeSubstringLengthByExpandingAroundCenter**(s, i, i+1);  
        **longestLength** = max(longestLength, oddLength);  
      return **longestLength**;

Time Complexity:  (n-1) \* [ (r-l+1) + (r-l+1)] = O(n \* 2n) = O(n^2)

Space Complexity:  O(1)

Verdict:  ACCEPTED