Longest Palindromic Substring - GeeksforGeeks

Source: https://www.geeksforgeeks.org/longest-palindromic-substring/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Longest Palindromic Substring Last Updated : 3 Oct, 2025 Given a string s , find the longest substring which is a palindrome. If there are multiple answers, then find the first appearing substring. Examples: Input: s = "forgeeksskeegfor" Output: "geeksskeeg" Explanation: The longest substring that reads the same forward and backward is "geeksskeeg". Other palindromes like "kssk" or "eeksskee" are shorter. Input: s = "Geeks" Output: "ee" Explanation: The substring "ee" is the longest palindromic part in "Geeks". All others are shorter single characters. Input: s = "abc" Output: "a" Explanation: No multi-letter palindromes exist. So the first character "a" is returned as the longest palindromic substring. Try it on GfG Practice Table of Content [Naive Approach] Generating all sub-strings - O(n^3) time and O(1) space [Better Approach - 1] Using Dynamic Programming - O(n^2) time and O(n^2) space [Better Approach - 2] Using Expansion from center - O(n^2) time and O(1) space [Expected Approach] Using Manacher's Algorithm - O(n) time and O(n) space [Naive Approach] Generating all sub-strings - O(n 3) time and O(1) space Generate all possible substrings of the given string. For each substring, check if it is a palindrome. If it is, update the result if its length is greater than the longest palindrome found so far. C++ #include <iostream> using namespace std ; // function to check if a substring // s[low..high] is a palindrome bool checkPal (string & s , int low , int high) { while (low < high) { if (s [low] != s [high]) return false ; low ++ ; high -- ; } return true ; } // function to find the longest palindrome substring string getLongestPal (string & s) { int n = s . size (); // all substrings of length 1 are palindromes int maxLen = 1 , start = 0 ; // nested loop to mark start and end index for (int i = 0 ; i < n ; i ++) { for (int j = i ; j < n ; j ++) { // check if the current substring is // a palindrome if (checkPal (s , i , j) && (j - i + 1) > maxLen) { start = i ; maxLen = j - i + 1 ; } } } return s . substr (start , maxLen); } int main () { string s = "forgeeksskeegfor" ; cout << getLongestPal (s) << endl ; return 0 ; } Java class GfG { // function to check if a substring // s[low..high] is a palindrome static boolean checkPal (String s , int low , int high) { while (low < high) { if (s . charAt (low) != s . charAt (high)) return false ; low ++ ; high -- ; } return true ; } // function to find the longest palindrome substring static String getLongestPal (String s) { int n = s . length (); // all substrings of length 1 are palindromes int maxLen = 1 , start = 0 ; // nested loop to mark start and end index for (int i = 0 ; i < n ; i ++) { for (int j = i ; j < n ; j ++) { // check if the current substring is // a palindrome if (checkPal (s , i , j) && (j - i + 1) > maxLen) { start = i ; maxLen = j - i + 1 ; } } } return s . substring (start , start + maxLen); } public static void main (String [] args) { String s = "forgeeksskeegfor" ; System . out . println (getLongestPal (s)); } } Python # function to check if a substring # s[low..high] is a palindrome def checkPal (str , low , high): while low < high : if str [low] != str [high]: return False low += 1 high -= 1 return True # function to find the longest palindrome substring def getLongestPal (s): n = len (s) # all substrings of length 1 are palindromes maxLen = 1 start = 0 # nested loop to mark start and end index for i in range (n): for j in range (i , n): # check if the current substring is # a palindrome if checkPal (s , i , j) and (j - i + 1) > maxLen : start = i maxLen = j - i + 1 return s [start : start + maxLen] if __name__ == "__main__" : s = "forgeeksskeegfor" print (getLongestPal (s)) C# using System ; class GfG { // function to check if a substring // s[low..high] is a palindrome static bool checkPal (string s , int low , int high) { while (low < high) { if (s [low] != s [high]) return false ; low ++ ; high -- ; } return true ; } // function to find the longest palindrome substring static string getLongestPal (string s) { int n = s . Length ; // all substrings of length 1 are palindromes int maxLen = 1 , start = 0 ; // nested loop to mark start and end index for (int i = 0 ; i < n ; i ++) { for (int j = i ; j < n ; j ++) { // check if the current substring is // a palindrome if (checkPal (s , i , j) && (j - i + 1) >

maxLen) { start = i ; maxLen = j - i + 1 ; } } } return s . Substring (start , maxLen); } static void Main (string [] args) { string s = "forgeeksskeegfor" ; Console . WriteLine (getLongestPal (s)); } } JavaScript // function to check if a substring // s[low..high] is a palindrome function checkPal (s , low , high) { while (low < high) { if (s [low] !== s [high]) return false ; low ++ ; high -- ; } return true ; } // function to find the longest palindrome substring function getLongestPal (s) { const n = s . length ; // all substrings of length 1 are palindromes let maxLen = 1 , start = 0 ; // nested loop to mark start and end index for (let i = 0 ; i < n ; i ++) { for (let j = i ; j < n ; j ++) { // check if the current substring is // a palindrome if (checkPal (s , i , j) && (j - i + 1) > maxLen) { start = i ; maxLen = j - i + 1 ; } } } return s . substring (start , start + maxLen); } // Driver Code const s = "forgeeksskeegfor" ; console . log (getLongestPal (s)); Output geeksskeeg [Better Approach - 1] Using Dynamic Programming - O(n 2) time and O(n 2) space The idea is to use Dynamic Programming to store the status of smaller substrings and use these results to check if a longer substring forms a palindrome. The main idea behind the approach is that if we know the status (i.e., palindrome or not) of the substring ranging [i, j], we can find the status of the substring ranging [i-1, j+1] by only matching the character str[i-1] and str[j+1]. If the substring from i to j is not a palindrome, then the substring from i-1 to j+1 will also not be a palindrome. Otherwise, it will be a palindrome only if str[i-1] and str[j+1] are the same. Base on this fact, we can create a 2D table (say table[][] which stores status of substring str[i . . . j]), and check for substrings with length from 1 to n. For each length find all the substrings starting from each character i and find if it is a palindrome or not using the above idea. The longest length for which a palindrome formed will be the required answer. Note: Refer to Longest Palindromic Substring using Dynamic Programming for detailed approach. C++ #include <iostream> #include <vector> using namespace std ; string getLongestPal (string s) { int n = s . size (); vector < vector < bool >> dp (n , vector < bool > (n , false)); // dp[i][j] if the substring from [i to j] // is a palindrome or not int start = 0 , maxLen = 1 ; // all substrings of length 1 are palindromes for (int i = 0 ; i < n ; ++ i) dp [i][i] = true ; // check for substrings of length 2 for (int i = 0 ; i < n - 1 ; ++ i) { if (s [i] == s [i + 1]) { dp [i][i + 1] = true ; if (maxLen == 1){ start = i ; maxLen = 2 ; } } } // check for substrings of length 3 and more for (int len = 3 ; len <= n ; ++ len) { for (int i = 0 ; i <= n - len ; ++ i) { int j = i + len - 1 ; // if s[i] == s[j] then check for // i [i+1 --- j-1] j if (s [i] == s [j] && dp [i + 1][j -1]) { dp [i][j] = true ; if (len > maxLen){ start = i ; maxLen = len ; } } } } return s . substr (start , maxLen); } int main () { string s = "forgeeksskeegfor" ; cout << getLongestPal (s) << endl ; } Java class GfG { public static String getLongestPal (String s) { int n = s . length (); boolean [][] dp = new boolean [n][n] ; // dp[i][j] if the substring from [i to j] // is a palindrome or not int start = 0 , maxLen = 1 ; // all substrings of length 1 are palindromes for (int i = 0 ; i < n ; ++ i) dp [i][i] = true ; // check for substrings of length 2 for (int i = 0 ; i < n - 1 ; ++ i) { if (s . charAt (i) == s . charAt (i + 1)) { dp [i][i + 1] = true ; if (maxLen == 1){ start = i ; maxLen = 2 ; } } } // check for substrings of length 3 and more for (int len = 3 ; len <= n ; ++ len) { for (int i = 0 ; i <= n - len ; ++ i) { int j = i + len - 1 ; // if s[i] == s[j] then check for // i [i+1 --- j-1] j if (s . charAt (i) == s . charAt (j) && dp [i + 1][j - 1]) { dp [i][j] = true ; if (len > maxLen){ start = i ; maxLen = len ; } } } } return s . substring (start , start + maxLen); } public static void main (String [] args) { String s = "forgeeksskeegfor" ; System . out . println (getLongestPal (s)); } } Python def getLongestPal (s): n = len (s) dp = [[False] * n for _ in range (n)] # dp[i][j] if the substring from [i to j] is a palindrome or not start = 0 maxLen = 1 # all substrings of length 1 are palindromes for i in range (n): dp [i][i] = True # check for substrings of length 2 for i in range (n - 1): if s [i] == s [i + 1]: dp [i][i + 1] = True if maxLen == 1 : start = i maxLen = 2 # check for substrings of length 3 and more for length in range (3 , n + 1): for i in range (n - length + 1): j = i + length - 1 # if s[i] == s[j] then check for [i+1 .. j-1] if s [i] == s [j] and dp [i + 1][j - 1]: dp [i][j] = True if length > maxLen : start = i maxLen = length return s [start : start + maxLen] if __name__ == "__main__" : s = "forgeeksskeegfor" print (getLongestPal (s)) C# using System ; class GfG { static string getLongestPal (string s) { int n = s . Length ; bool [,] dp = new bool [n , n]; // dp[i][j] if the substring from [i to j] // is a palindrome or not int start = 0 , maxLen = 1 ; // all substrings of length 1 are palindromes for (int i = 0 ; i < n ; ++ i) dp [i , i] = true ; // check for substrings of length 2 for (int i = 0 ; i < n - 1 ; ++ i) { if (s [i] == s [i + 1]) { dp [i , i + 1] = true ; if (maxLen == 1) { start = i ; maxLen = 2 ; } } } // check for substrings of length 3 and more for (int len = 3 ; len <= n ; ++ len) { for (int i = 0 ; i <= n - len ; ++ i) { int j = i + len - 1 ; // if s[i] == s[j] then check for // i [i+1 --- j-1] j if (s [i] == s [j] && dp [i + 1 , j - 1]) { dp [i , j] = true ; if (len > maxLen) { start = i ; maxLen = len ; } } } } return s . Substring (start , maxLen); } public static void Main (string [] args) { string s = "forgeeksskeegfor" ; Console . WriteLine (getLongestPal (s)); } } JavaScript function getLongestPal (s) { let n = s . length ; const dp = Array . from ({ length : n }, () => new Uint8Array (n)); // dp[i][j] if the substring from [i to j] // is a palindrome or not let start = 0 , maxLen = 1 ; // all substrings of length 1 are palindromes for (let i = 0 ; i < n ; ++ i) dp [i][i] = true ; // check for substrings of length 2

for (let i = 0 ; i < n - 1 ; ++ i) { if (s [i] === s [i + 1]) { dp [i][i + 1] = true ; if (maxLen === 1) { start = i ; maxLen = 2 ; } } } // check for substrings of length 3 and more for (let len = 3 ; len <= n ; ++ len) { for (let i = 0 ; i <= n - len ; ++ i) { let j = i + len - 1 ; // if s[i] == s[j] then check for // i [i+1 --- j-1] j if (s [i] === s [j] && dp [i + 1][j - 1]) { dp [i][j] = true ; if (len > maxLen) { start = i ; maxLen = len ; } } } } return s . substring (start , start + maxLen); } // Driver Code let s = "forgeeksskeegfor" ; console . log (getLongestPal (s)); Output geeksskeeg [Better Approach - 2] Using Expansion from center - O(n 2) time and O(1) space The idea is to traverse each character in the string and treat it as a potential center of a palindrome, trying to expand around it in both directions while checking if the expanded substring remains a palindrome. => For each position, we check for both odd-length palindromes (where the current character is the center) and even-length palindromes (where the current character and the next character together form the center). => As we expand outward from each center, we keep track of the start position and length of the longest palindrome found so far, updating these values whenever we find a longer valid palindrome. Step-by-step approach: Iterate through each character in the string, treating it as the center of a potential palindrome. For each center, expand in two ways: one for odd-length palindromes (center at index i) and one for even-length palindromes (center between indices i and i+1) Use two pointers low and high to track the left and right boundaries of the current palindrome. While low and high are in bounds and s[low] == s[high], expand outward. If the current palindrome length (high - low + 1) is greater than the previous maximum, update the starting index and max length. After checking all centers, return the substring starting at start with length maxLen. C++ #include <iostream> using namespace std ; string getLongestPal (string & s) { int n = s . length (); int start = 0 , maxLen = 1 ; for (int i = 0 ; i < n ; i ++) { // this runs two times for both odd and even // length palindromes. // j = 0 means odd and j = 1 means even length for (int j = 0 ; j <= 1 ; j ++) { int low = i ; int high = i + j ; // expand substring while it is a palindrome // and in bounds while (low >= 0 && high < n && s [low] == s [high]) { int currLen = high - low + 1 ; if (currLen > maxLen) { start = low ; maxLen = currLen ; } low -- ; high ++ ; } } } return s . substr (start , maxLen); } int main () { string s = "forgeeksskeegfor" ; cout << getLongestPal (s) << endl ; return 0 ; } Java class GfG { static String getLongestPal (String s) { int n = s . length (); int start = 0 , maxLen = 1 ; for (int i = 0 ; i < n ; i ++) { // this runs two times for both odd and even // length palindromes. // j = 0 means odd and j = 1 means even length for (int j = 0 ; j <= 1 ; j ++) { int low = i ; int high = i + j ; // expand substring while it is a palindrome // and in bounds while (low >= 0 && high < n && s . charAt (low) == s . charAt (high)) { int currLen = high - low + 1 ; if (currLen > maxLen) { start = low ; maxLen = currLen ; } low -- ; high ++ ; } } } return s . substring (start , start + maxLen); } public static void main (String [] args) { String s = "forgeeksskeegfor" ; System . out . println (getLongestPal (s)); } } Python def getLongestPal (s): n = len (s) start , maxLen = 0 , 1 for i in range (n): # this runs two times for both odd and even # length palindromes. # j = 0 means odd and j = 1 means even length for j in range (2): low , high = i , i + j # expand substring while it is a palindrome # and in bounds while low >= 0 and high < n and s [low] == s [high]: currLen = high - low + 1 if currLen > maxLen : start = low maxLen = currLen low -= 1 high += 1 return s [start : start + maxLen] if __name__ == "__main__" : s = "forgeeksskeegfor" print (getLongestPal (s)) C# using System ; class GfG { static string getLongestPal (string s) { int n = s . Length ; int start = 0 , maxLen = 1 ; for (int i = 0 ; i < n ; i ++) { // this runs two times for both odd and even // length palindromes. // j = 0 means odd and j = 1 means even length for (int j = 0 ; j <= 1 ; j ++) { int low = i ; int high = i + j ; // expand substring while it is a palindrome // and in bounds while (low >= 0 && high < n && s [low] == s [high]) { int currLen = high - low + 1 ; if (currLen > maxLen) { start = low ; maxLen = currLen ; } low -- ; high ++ ; } } } return s . Substring (start , maxLen); } static void Main (string [] args) { string s = "forgeeksskeegfor" ; Console . WriteLine (getLongestPal (s)); } } JavaScript function getLongestPal (s) { const n = s . length ; let start = 0 , maxLen = 1 ; for (let i = 0 ; i < n ; i ++) { // this runs two times for both odd and even // length palindromes. // j = 0 means odd and j = 1 means even length for (let j = 0 ; j <= 1 ; j ++) { let low = i ; let high = i + j ; // expand substring while it is a palindrome // and in bounds while (low >= 0 && high < n && s [low] === s [high]) { const currLen = high - low + 1 ; if (currLen > maxLen) { start = low ; maxLen = currLen ; } low -- ; high ++ ; } } } return s . substring (start , start + maxLen); } // Driver Code const s = "forgeeksskeegfor" ; console . log (getLongestPal (s)); Output geeksskeeg [Expected Approach] Using Manacher’s Algorithm - O(n) time and O(n) space The idea is to use Manacher’s algorithm , which transforms the input string by inserting separators (#) and sentinels to handle both even and odd-length palindromes uniformly. For each position in the transformed string, we expand the longest possible palindrome centered there using mirror symmetry and previously computed values. This expansion is bounded efficiently using the rightmost known palindrome range to avoid redundant checks. Whenever a longer palindrome is found,

we update its length and starting index in the original string. C++ #include <iostream> #include <vector> #include <string> using namespace std ; class manacher { public : // p[i] stores the radius of the palindrome // centered at position i in ms vector < int > p ; // transformed string with sentinels // and separators string ms ; manacher (string & s) { // left sentinel to avoid bounds check ms = "@" ; for (char c : s) { // insert '#' between every character ms += "#" + string (1 , c); } // right sentinel ms += "#$" ; runManacher (); } void runManacher () { int n = ms . size (); p . assign (n , 0); int l = 0 , r = 0 ; for (int i = 1 ; i < n - 1 ; ++ i) { // initialize p[i] based on its mirror // and current [l, r] range if (r + l - i >= 0 && r + l - i < n) p [i] = max (0 , min (r - i , p [r + l - i])); // try expanding around center i while (ms [i + 1 + p [i]] == ms [i - 1 - p [i]]) { ++ p [i]; } // update [l, r] if the new palindrome goes // beyond current right boundary if (i + p [i] > r) { l = i - p [i]; r = i + p [i]; } } } // return the radius of the longest palindrome // centered at original index 'cen' int getLongest (int cen , int odd) { int pos = 2 * cen + 2 + ! odd ; return p [pos]; } // checks whether the substring // s[l..r] is a palindrome bool check (int l , int r) { int res = getLongest ((r + l) / 2 , (r - l + 1) % 2); return (r - l + 1) <= res ; } }; // finds and returns the longest palindromic substring in s string getLongestPal (string & s) { int n = s . size (), maxLen = 1 , start = 0 ; manacher M (s); for (int i = 0 ; i < n ; ++ i) { int oddLen = M . getLongest (i , 1); if (oddLen > maxLen) { // update start for odd-length palindrome start = i - (oddLen - 1) / 2 ; } int evenLen = M . getLongest (i , 0); if (evenLen > maxLen) { // update start for even-length palindrome start = i - (evenLen - 1) / 2 ; } maxLen = max (maxLen , max (oddLen , evenLen)); } return s . substr (start , maxLen); } int main () { string s = "forgeeksskeegfor" ; cout << getLongestPal (s) << endl ; return 0 ; } Java class Manacher { // p[i] stores the radius of the palindrome // centered at position i in ms int [] p ; // transformed string with sentinels // and separators String ms ; public Manacher (String s) { // left sentinel to avoid bounds check StringBuilder sb = new StringBuilder ("@"); for (char c : s . toCharArray ()) { // insert '#' between every character sb . append ("#"). append (c); } // right sentinel sb . append ("#$"); ms = sb . toString (); runManacher (); } private void runManacher () { int n = ms . length (); p = new int [n] ; int l = 0 , r = 0 ; for (int i = 1 ; i < n - 1 ; ++ i) { int mirror = l + r - i ; if (mirror >= 0 && mirror < n) { p [i] = Math . max (0 , Math . min (r - i , p [mirror])); } else { p [i] = 0 ; } // try expanding around center i while ((i + 1 + p [i]) < n && (i - 1 - p [i]) >= 0 && ms . charAt (i + 1 + p [i]) == ms . charAt (i - 1 - p [i])) { ++ p [i] ; } // update [l, r] if the new palindrome goes // beyond current right boundary if (i + p [i] > r) { l = i - p [i] ; r = i + p [i] ; } } } // return the radius of the longest palindrome // centered at original index 'cen' public int getLongest (int cen , int odd) { int pos = 2 * cen + 2 + (odd == 0 ? 1 : 0); return p [pos] ; } // checks whether the substring // s[l..r] is a palindrome public boolean check (int l , int r) { int res = getLongest ((r + l) / 2 , (r - l + 1) % 2); return (r - l + 1) <= res ; } } class GfG { // finds and returns the longest // palindromic substring in s public static String getLongestPal (String s) { int n = s . length (), maxLen = 1 , start = 0 ; Manacher M = new Manacher (s); for (int i = 0 ; i < n ; ++ i) { int oddLen = M . getLongest (i , 1); if (oddLen > maxLen) { // update start for odd-length palindrome start = i - (oddLen - 1) / 2 ; } int evenLen = M . getLongest (i , 0); if (evenLen > maxLen) { // update start for even-length palindrome start = i - (evenLen - 1) / 2 ; } maxLen = Math . max (maxLen , Math . max (oddLen , evenLen)); } return s . substring (start , start + maxLen); } public static void main (String [] args) { String s = "forgeeksskeegfor" ; System . out . println (getLongestPal (s)); } } Python class manacher : # p[i] stores the radius of the palindrome # centered at position i in ms def __init__ (self , s): # transformed string with sentinels # and separators self . ms = "@" for c in s : self . ms += "#" + c self . ms += "#$" self . p = [0] * len (self . ms) self . runManacher () def runManacher (self): n = len (self . ms) l = r = 0 for i in range (1 , n - 1): mirror = l + r - i if 0 <= mirror < n : self . p [i] = max (0 , min (r - i , self . p [mirror])) else : self . p [i] = 0 # try expanding around center i while (i + 1 + self . p [i] < n and i - 1 - self . p [i] >= 0 and self . ms [i + 1 + self . p [i]] == self . ms [i - 1 - self . p [i]]): self . p [i] += 1 # update [l, r] if the new palindrome goes # beyond current right boundary if i + self . p [i] > r : l = i - self . p [i] r = i + self . p [i] # return the radius of the longest palindrome # centered at original index 'cen' def getLongest (self , cen , odd): pos = 2 * cen + 2 + (0 if odd else 1) return self . p [pos] # checks whether the substring # s[l..r] is a palindrome def check (self , l , r): length = r - l + 1 return length <= self . getLongest ((l + r) // 2 , length % 2) # finds and returns the longest # palindromic substring in s def getLongestPal (s): n = len (s) maxLen = 1 start = 0 M = manacher (s) for i in range (n): oddLen = M . getLongest (i , 1) if oddLen > maxLen : # update start for odd-length palindrome start = i - (oddLen - 1) // 2 evenLen = M . getLongest (i , 0) if evenLen > maxLen : # update start for even-length palindrome start = i - (evenLen - 1) // 2 maxLen = max (maxLen , max (oddLen , evenLen)) return s [start : start + maxLen] if __name__ == "__main__" : s = "forgeeksskeegfor" print (getLongestPal (s)) C# using System ; class manacher { // p[i] stores the radius of the palindrome // centered at position i in ms public int [] p ; // transformed string with sentinels // and separators public

```
string ms ; public manacher ( string s ) { ms = "@" ; foreach ( char c in s ) { ms += "#" + c ; } ms += "#$" ; runManacher (); } void runManacher () { int n = ms . Length ; p = new int [ n ]; int l = 0 , r = 0 ; for ( int i = 1 ; i < n - 1 ; ++ i ) { int mirror = l + r - i ; if ( mirror >= 0 && mirror < n ) p [ i ] = Math . Max ( 0 , Math . Min ( r - i , p [ mirror ])); else p [ i ] = 0 ; // try expanding around center i while (( i + 1 + p [ i ]) < n && ( i - 1 - p [ i ]) >= 0 && ms [ i + 1 + p [ i ]] == ms [ i - 1 - p [ i ]]) { ++ p [ i ]; } // update [l, r] if the new palindrome goes // beyond current right boundary if ( i + p [ i ] > r ) { l = i - p [ i ]; r = i + p [ i ]; } } } // return the radius of the longest palindrome // centered at original index 'cen' public int getLongest ( int cen , int odd ) { int pos = 2 * cen + 2 + ( odd == 0 ? 1 : 0 ); return p [ pos ]; } // checks whether the substring // s[l..r] is a palindrome public bool check ( int l , int r ) { int res = getLongest (( l + r ) / 2 , ( r - l + 1 ) % 2 ); return ( r - l + 1 ) <= res ; } } class GfG { // finds and returns the longest // palindromic substring in s public static string getLongestPal ( string s ) { int n = s . Length , maxLen = 1 , start = 0 ; manacher M = new manacher ( s ); for ( int i = 0 ; i < n ; ++ i ) { int oddLen = M . getLongest ( i , 1 ); if ( oddLen > maxLen ) { // update start for odd-length palindrome start = i - ( oddLen - 1 ) / 2 ; } int evenLen = M . getLongest ( i , 0 ); if ( evenLen > maxLen ) { // update start for even-length palindrome start = i - ( evenLen - 1 ) / 2 ; } maxLen = Math . Max ( maxLen , Math . Max ( oddLen , evenLen )); } return s . Substring ( start , maxLen ); } public static void Main ( string [] args ) { string s = "forgeeksskeegfor" ; Console . WriteLine ( getLongestPal ( s )); } } JavaScript class Mancjher { // p[i] stores the radius of the palindrome // centered at position i in ms constructor ( s ) { this . ms = "@" ; for ( let c of s ) { this . ms += "#" + c ; } this . ms += "#$" ; // right sentinel this . p = new Array ( this . ms . length ). fill ( 0 ); this . runManacher (); } runManacher () { let n = this . ms . length ; let l = 0 , r = 0 ; for ( let i = 1 ; i < n - 1 ; ++ i ) { let mirror = l + r - i ; if ( mirror >= 0 && mirror < n ) { this . p [ i ] = Math . max ( 0 , Math . min ( r - i , this . p [ mirror ])); } else { this . p [ i ] = 0 ; } // try expanding around center i while ( ( i + 1 + this . p [ i ]) < n && ( i - 1 - this . p [ i ]) >= 0 && this . ms [ i + 1 + this . p [ i ]] === this . ms [ i - 1 - this . p [ i ]] ) { ++ this . p [ i ]; } // update [l, r] if new palindrome goes beyond right if ( i + this . p [ i ] > r ) { l = i - this . p [ i ]; r = i + this . p [ i ]; } } } // return the radius of the longest palindrome // centered at original index 'cen' getLongest ( cen , odd ) { let pos = 2 * cen + 2 + ( odd === 0 ? 1 : 0 ); return this . p [ pos ]; } // checks whether the substring // s[l..r] is a palindrome check ( l , r ) { let len = r - l + 1 ; return len <= this . getLongest (( l + r ) >> 1 , len % 2 ); } } // finds and returns the longest // palindromic substring in s function getLongestPal ( s ) { let n = s . length , maxLen = 1 , start = 0 ; const M = new Mancjher ( s ); for ( let i = 0 ; i < n ; ++ i ) { let oddLen = M . getLongest ( i , 1 ); if ( oddLen > maxLen ) { start = i - Math . floor (( oddLen - 1 ) / 2 ); } let evenLen = M . getLongest ( i , 0 ); if ( evenLen > maxLen ) { start = i - Math . floor (( evenLen - 1 ) / 2 ); } maxLen = Math . max ( maxLen , Math . max ( oddLen , evenLen )); } return s . substring ( start , start + maxLen ); } // Driver Code const s = "forgeeksskeegfor" ; console . log ( getLongestPal ( s )); Output geeksskeeg Comment Article Tags: Article Tags: Strings Dynamic Programming DSA Microsoft Amazon Groupon Qualcomm Samsung Accolite Zoho MakeMyTrip Visa strings + 9 More
```