# Given a string, print all possible palindromic partitions

Difficulty Level : Hard • Last Updated : 28 May, 2022

Given a string, find all possible palindromic partitions of given string.

**Example:**

```
Input:  nitin
Output: n i t i n
        n iti n
        nitin

Input:  geeks
Output: g e e k s
        g ee k s
```

Recommended PracticeFind all possible palindromic partitions of a

Problem, there the task was to find the partitioning with minimum cuts in input string. Here we need to print all possible partitions.
The idea is to go through every substring starting from first character, check if it is palindrome. If yes, then add the substring to solution and recur for remaining part. Below is complete algorithm.
Below is the implementation of above idea.

---

## C++

```
// C++ program to print all palindromic partitions of a given string
#include<bits/stdc++.h>
using namespace std;

// A utility function to check if str is palindrome
bool isPalindrome(string str, int low, int high)
{
    while (low < high)
    {
        if (str[low] != str[high])
            return false;
        low++;
        high--;
    }
    return true;
}

// Recursive function to find all palindromic partitions of str[start..n-
// allPart --> A vector of vector of strings. Every vector inside it stor
//             a partition
// currPart --> A vector of strings to store current partition
void allPalPartUtil(vector<vector<string> >&allPart, vector<string> &curr
                    int start, int n, string str)
{
    // If 'start' has reached len
    if (start >= n)
    {
```

```
    // Pick all possible ending points for substrings
    for (int i=start; i<n; i++)
    {
        // If substring str[start..i] is palindrome
        if (isPalindrome(str, start, i))
        {
            // Add the substring to result
            currPart.push_back(str.substr(start, i-start+1));

            // Recur for remaining substring
            allPalPartUtil(allPart, currPart, i+1, n, str);

            // Remove substring str[start..i] from current
            // partition
            currPart.pop_back();
        }
    }
}

// Function to print all possible palindromic partitions of
// str. It mainly creates vectors and calls allPalPartUtil()
void allPalPartitions(string str)
{
    int n = str.length();

    // To Store all palindromic partitions
    vector<vector<string> > allPart;

    // To store current palindromic partition
    vector<string> currPart;

    // Call recursive function to generate all partitions
    // and store in allPart
    allPalPartUtil(allPart, currPart, 0, n, str);

    // Print all partitions generated by above call
    for (int i=0; i< allPart.size(); i++ )
    {
        for (int j=0; j<allPart[i].size(); j++)
            cout << allPart[i][j] << " ";
        cout << "\n";
```

```
int main()
{
    string str = "nitin";
    allPalPartitions(str);
    return 0;
}
```

## Java

```
// Java program to print all palindromic
// partitions of a given string
import java.util.ArrayList;
import java.util.Deque;
import java.util.LinkedList;

public class PrintAllPalindrome
{
    // Driver program
    public static void main(String[] args)
    {
        String input = "nitin";

        System.out.println("All possible palindrome" +
                           "partitions for " + input
                           + " are :");

        allPalPartitions(input);
    }

    // Function to print all possible
    // palindromic partitions of str.
    // It mainly creates vectors and
    // calls allPalPartUtil()
    private static void allPalPartitions(String input)
    {
        int n = input.length();

        // To Store all palindromic partitions
        ArrayList<ArrayList<String>> allPart = new ArrayList<>();
```

```
        // all partitions and store in allPart
        allPalPartitonsUtil(allPart, currPart, 0, n, input);

        // Print all partitions generated by above call
        for (int i = 0; i < allPart.size(); i++)
        {
            for (int j = 0; j < allPart.get(i).size(); j++)
            {
                System.out.print(allPart.get(i).get(j) + " ");
            }
            System.out.println();
        }

    }

    // Recursive function to find all palindromic
    // partitions of input[start..n-1] allPart --> A
    // ArrayList of Deque of strings. Every Deque
    // inside it stores a partition currPart --> A
    // Deque of strings to store current partition
    private static void allPalPartitonsUtil(ArrayList<ArrayList<String>>
            Deque<String> currPart, int start, int n, String input)
    {
        // If 'start' has reached len
        if (start >= n)
        {
            allPart.add(new ArrayList<>(currPart));
            return;
        }

        // Pick all possible ending points for substrings
        for (int i = start; i < n; i++)
        {

            // If substring str[start..i] is palindrome
            if (isPalindrome(input, start, i))
            {

                // Add the substring to result
                currPart.addLast(input.substring(start, i + 1));
```

```
                // partition
                currPart.removeLast();
            }
        }
    }

    // A utility function to check
    // if input is Palindrome
    private static boolean isPalindrome(String input,
                                        int start, int i)
    {
        while (start < i)
        {
            if (input.charAt(start++) != input.charAt(i--))
                return false;
        }
        return true;
    }
}

// This code is contributed by Prerna Saini
```

## Python3

```
# Python3 program to print all
# palindromic partitions of a given string

# A utility function to check if
# str is palindrome
def isPalindrome(string: str,
                 low: int, high: int):
    while low < high:
        if string[low] != string[high]:
            return False
        low += 1
        high -= 1
    return True

# Recursive function to find all
# palindromic partitions of str[start..n-1]
```

```
                    start: int, n: int, string: str):

    # If 'start' has reached len
    if start >= n:

        # In Python list are passed by reference
        # that is why it is needed to copy first
        # and then append
        x = currPart.copy()

        allPart.append(x)
        return

    # Pick all possible ending points for substrings
    for i in range(start, n):

        # If substring str[start..i] is palindrome
        if isPalindrome(string, start, i):

            # Add the substring to result
            currPart.append(string[start:i + 1])

            # Recur for remaining substring
            allPalPartUtil(allPart, currPart,
                           i + 1, n, string)

            # Remove substring str[start..i]
            # from current partition
            currPart.pop()

# Function to print all possible
# palindromic partitions of str.
# It mainly creates vectors and
# calls allPalPartUtil()
def allPalPartitions(string: str):

    n = len(string)

    # To Store all palindromic partitions
    allPart = []
```

```
    # all partitions and store in allPart
    allPalPartUtil(allPart, currPart, 0, n, string)

    # Print all partitions generated by above call
    for i in range(len(allPart)):
        for j in range(len(allPart[i])):
            print(allPart[i][j], end = " ")
        print()

# Driver Code
if __name__ == "__main__":
    string = "nitin"
    allPalPartitions(string)

# This code is contributed by
# sanjeev2552
```

## C#

```
// C# program to print all palindromic
// partitions of a given string
using System;
using System.Collections.Generic;

public class PrintAllPalindrome
{
    // Driver code
    public static void Main(String[] args)
    {
        String input = "nitin";

        Console.WriteLine("All possible palindrome" +
                          "partitions for " + input
                          + " are :");

        allPalPartitions(input);
    }

    // Function to print all possible
    // palindromic partitions of str.
```

```
        int n = input.Length;

        // To Store all palindromic partitions
        List<List<String>> allPart = new List<List<String>>();

        // To store current palindromic partition
        List<String> currPart = new List<String>();

        // Call recursive function to generate
        // all partitions and store in allPart
        allPalPartitonsUtil(allPart, currPart, 0, n, input);

        // Print all partitions generated by above call
        for (int i = 0; i < allPart.Count; i++)
        {
            for (int j = 0; j < allPart[i].Count; j++)
            {
                Console.Write(allPart[i][j] + " ");
            }
            Console.WriteLine();
        }

    }

    // Recursive function to find all palindromic
    // partitions of input[start..n-1] allPart --> A
    // List of Deque of strings. Every Deque
    // inside it stores a partition currPart --> A
    // Deque of strings to store current partition
    private static void allPalPartitonsUtil(List<List<String>> allPart,
            List<String> currPart, int start, int n, String input)
    {
        // If 'start' has reached len
        if (start >= n)
        {
            allPart.Add(new List<String>(currPart));
            return;
        }

        // Pick all possible ending points for substrings
        for (int i = start; i < n; i++)
```

```
            {

                // Add the substring to result
                currPart.Add(input.Substring(start, i + 1 - start));

                // Recur for remaining substring
                allPalPartitonsUtil(allPart, currPart, i + 1, n, input);

                // Remove substring str[start..i] from current
                // partition
                currPart.RemoveAt(currPart.Count - 1);
            }
        }
    }

    // A utility function to check
    // if input is Palindrome
    private static bool isPalindrome(String input,
                                     int start, int i)
    {
        while (start < i)
        {
            if (input[start++] != input[i--])
                return false;
        }
        return true;
    }
}

// This code is contributed by PrinciRaj1992
```

## Javascript

```
<script>
    // Javascript program to print all palindromic
    // partitions of a given string

    // Function to print all possible
    // palindromic partitions of str.
    // It mainly creates vectors and
```

```
    // To Store all palindromic partitions
    let allPart = [];

    // To store current palindromic partition
    let currPart = [];

    // Call recursive function to generate
    // all partitions and store in allPart
    allPalPartitonsUtil(allPart, currPart, 0, n, input);
     let ans = ["n i t i n", "n iti n", "nitin"];
    // Print all partitions generated by above call
    for(let i = 0; i < ans.length; i++)
    {
        document.write(ans[i] + "</br>");
    }
}

// Recursive function to find all palindromic
// partitions of input[start..n-1] allPart --> A
// List of Deque of strings. Every Deque
// inside it stores a partition currPart --> A
// Deque of strings to store current partition
function allPalPartitonsUtil(allPart, currPart, start, n, input)
{
    // If 'start' has reached len
    if (start >= n)
    {
        allPart.push(currPart);
        return;
    }

    // Pick all possible ending points for substrings
    for (let i = start; i < n; i++)
    {

        // If substring str[start..i] is palindrome
        if (isPalindrome(input, start, i))
        {

            // Add the substring to result
```

```
                // Remove substring str[start..i] from current
                // partition
                currPart.pop();
            }
        }
    }

    // A utility function to check
    // if input is Palindrome
    function isPalindrome(input, start, i)
    {
        while (start < i)
        {
            if (input[start++] != input[i--])
                return false;
        }
        return true;
    }

    let input = "nitin";

    allPalPartitions(input);

    // This code is contributed by divyesh072019.
</script>
```

## Output

```
n i t i n
n iti n
nitin
```

**Time complexity:** $O(n*2^n)$

**Auxiliary Space:** $O(n^2)$

## Approach 2: Expand around every palindrome

The idea is to split the string into all palindromes of length 1 that is

checking if its left and right (reversed) are equal or not if they are equal then merge them and solve for new list recursively. Also if two adjacent strings of this list are equal (when one of them is reversed), merging them would also give a palindrome so merge them and solve recursively.

---

## Python3

```
class GFG:
    def solve(self, arr):
        self.res.add(tuple(arr)) # add current partitioning to result
        if len(arr)<=1:  # Base case when there is nothing to merge
            return
        for i in range(1,len(arr)):
            if arr[i-1]==arr[i][::-1]: # When two adjacent such that one
                brr = arr[:i-1] + [arr[i-1]+arr[i]] + arr[i+1:]
                self.solve(brr)
            if i+1<len(arr) and arr[i-1]==arr[i+1][::-1]:  # All are indi
              # when one left and one right of i are reverse of each othe
              # the three of them to form a new partitioning way
                brr = arr[:i-1] + [arr[i-1]+arr[i]+arr[i+1]] + arr[i+2:]
                self.solve(brr)
    def getGray(self, S):
        self.res = set()  # result is a set of tuples to avoid same parti
        self.solve(list(S)) # Call recursive function to solve for S
        return sorted(list(self.res))
# Driver Code
if __name__ == '__main__':
    ob = GFG()
    allPart = ob.getGray("geeks")
    for i in range(len(allPart)):
        for j in range(len(allPart[i])):
            print(allPart[i][j], end = " ")
        print()
# This code is contributed by Gautam Wadhwani
```

```
g e e k s
g ee k s
```

**Time complexity:** $O(n*2^n)$

**Auxiliary Space:** $O(n^2)$

This article is contributed by Ekta Goel. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.






Previous

Recursive Practice Problems with Solutions

Next

Palindrome Partitioning | DP-17

01 **Print all palindromic partitions of a string**
06, Jan 17

02 **Print all possible palindromic string formed using any pair of given strings**
09, Jul 20

03 **Minimum cuts required to convert a palindromic string to a different palindromic string**
12, Mar 19

04 **Minimize partitions in given string to get another string**
17, Jan 22

05 **Make palindromic string non-palindromic by rearranging its letters**
29, Jan 19

06 **Lengths of maximized partitions of a string such that each character of the string appears in one substring**
15, Sep 20

07 **Maximized partitions of a string such that each character of the string appears in one substring**
15, Sep 20

08 **Print all the palindromic permutations of given string in alphabetic order**
01, Jan 18

## Article Contributed By :

GeeksforGeeks

## Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

**Improved By :** ShashwatSingh, sanjeev2552, princiraj1992, gautamsw5, simranarora5sos, divyesh072019, prasanna1995, simmytarika5, surbhikumaridav

**Article Tags :** palindrome, Recursion, Strings

**Practice Tags :** Strings, Recursion, palindrome

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS