Summary

Time Complexity: O(N*N), exponential since there could be these many substring combinations
Space Complexity: O(N), since the recursion call stack could be up to N deep if every letter was a substring.

Problem Pattern

1. You need to divide your given string into substrings, where each substring should be a palindrome. Note that a single character is also a palindrome, that is, the string 'e' can also be called a palindrome.

```
Input  : s = 'aeebs'

Output : [['a', 'e', 'e', 'b', 's'],

                ['a', 'ee', 'b', 's']]
```

2. We have to list all possible partitions so we will think in the direction of recursion. When we are on index i, we incrementally check all substrings starting from i for being palindromic. If found, we recursively solve the problem for the remaining string and add this in our solution.
3. You will also need to write a function to check if a string is a palindrome. To check whether it's a palindrome or not, iterate on string by taking two pointers. Initialize the first to start and other to end of string.

Problem Approach

1. We can maintain a 2-dimensional vector for storing all possible partitions and a temporary vector for storing the current partition, a new starting index of string to check partitions as we have already checked partitions before this index.
2. Note that we will be using a vector because we do not know how many substrings we will obtain. And we cannot create an array of unknown size.
3. Now keep on iterating further on string and check if it is palindrome or not.
4. If it is a palindrome then add this string in the current partitions vector. Recurse on this new string if it is not the end of the string. After coming back, change the

current partition vector to the old one as it might have changed in the recursive step.

5. If we reach the end of the string while iterating then we have our partitions in our temporary vector so we will add it in results.

Problem Pseudocode

```
List<List<String>> partition(String s) {

    List<List<String>> partitions

    partition(s, partitions)

    return partitions

}


// Generates all palindromic partitions of 's' and stores the
result in 'v'

void partition(String s, List<List<String>> v) {

    List<String> temp

    // calling addString method it adds all the palindromic
partitions to v

    v = addStrings(v, s, temp, 0)

}


List<List<String>> addStrings(List<List<String>> v, String s,

                                                    List<String>
temp, int index) {
```

```
    int len = s.length()
    String str = ''
    List<String> current   // Initialize to current temp


    if (index == 0)
        temp.clear()


    // Iterate over the string
    for (i = index to len) {
        str = str + s.charAt(i)
        // check whether the substring is palindrome or not
        if (isPalindrome(str)) {
            // if palindrome add it to temp list
            temp.add(str)
            if (i + 1 < len)
                v = addStrings(v,s,temp,i+1)
            else
                // if end of the string is reached add temp to v
                v.add(temp)
            // temp is reinitialized with the current i.
```

```
            temp = new ArrayList<String>(current)

        }

    }

    return v;

}
```