Permutation Partioning

We choose position to partition. The partition can only be made if the string before partitioning point is palindrome. we choose an index and make sure that we find all the partitions that can be done with that index.

So use a loop to find out all positions at which partition can be made for each combination.

```
#include <bits/stdc++.h>
bool checkPalin(int start, int end,string& s)
{
    if(start==end)
        return true;
    if(s[start]!=s[end])
        return false;
    if(start<end)
        return checkPalin(start+1,end-1,s);
}
void helper(int ind,int n,string& s,vector<string> temp,vector<vector<string>>& ans)
{
    if(ind==n)
    {
        ans.push_back(temp);
        return;
    }
    for(int i=ind;i<n;i++)
    {
        if(checkPalin(ind,i,s))
        {
            temp.push_back(s.substr(ind,i-ind+1));
            helper(i+1,n,s,temp,ans);
            temp.pop_back();
        }
    }
}
vector<vector<string>> partition(string &s)
{
    // Write your code here.
    vector<string> temp;
    vector<vector<string>> ans;
    helper(0,s.length(),s,temp,ans);
    return ans;
}
```

- Time Complexity : **O((2^n) *k*(n/2))**

Reason: O(2^n) to generate every substring and **O(n/2)** to check if the substring generated is a palindrome. O(k) is for inserting the palindromes in another data structure, where k is the average length of the palindrome list.

- **Space Complexity: O(k * x)**

Reason: The space complexity can vary depending upon the length of the answer. k is the average length of the list of palindromes and if we have x such list of palindromes in our final answer. The depth of the recursion tree is n, so the auxiliary space required is equal to the O(n).