```
                tempList.add(nums[i]);
                backtrack(list, tempList, nums, remain - nums[i], i + 1);
                tempList.remove(tempList.size() - 1);
            }
        }
    }
```

Palindrome Partitioning : https://leetcode.com/problems/palindrome-partitioning/

```
public List<List<String>> partition(String s) {
    List<List<String>> list = new ArrayList<>();
    backtrack(list, new ArrayList<>(), s, 0);
    return list;
}

public void backtrack(List<List<String>> list, List<String> tempList, String s, int
    if(start == s.length())
        list.add(new ArrayList<>(tempList));
    else{
        for(int i = start; i < s.length(); i++){
            if(isPalindrome(s, start, i)){
                tempList.add(s.substring(start, i + 1));
                backtrack(list, tempList, s, i + 1);
                tempList.remove(tempList.size() - 1);
            }
        }
    }
}

public boolean isPalindrome(String s, int low, int high){
    while(low < high)
        if(s.charAt(low++) != s.charAt(high--)) return false;
    return true;
}
```

backtracking java