**Backtracking**

<https://leetcode.com/problems/permutations/discuss/18239/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partioning)>

**Subset**

Main{

addSubset(new ArrayList<Integer>(), input, 0, new…);

}

public void addSubset(List<Integer> current, List<Integer> input, int start, List<List<Integer>> result){

result.add(new ArrayList<Integer>(current));

for(int i = start; i < input.size(); i++){

current.add(input.get(i));

addSubset(current, input, start+1; result);

current.remove(current.size()-1);

}

}

**Subset – without redundant**

For(){

If(input.get(i) == input.get(i-1)) continue;

}

**Permutation**

Only difference:

If(start == input.size()) result.add;

Else{

}

**Permutation – without redundant**

//similar

**Palindrome Partition**

private boolean isPalindrome(String s){

int i = 0;

int j = s.length-1;

while(i<j){

if(s.charAt(i) != s.charAt(j)) return false;

}

return true;

}

private void addPartition(List<String> current; List<List<String>> list, int start, String s){

if(start == s.length) list.add(new ArrayList<String>(current));

else{

for(int i = start; i < s.length; i++){

if( isPalindrome(s.subString(start, i) )){

current.add(s.subString(start, i));

addPartition(current, list, i+1, s);

current.remove(current.size()-1);

}

}

}

}

**Combination Sum**

private void addComb(List<Integer> current, List<List<String>> list; int index; int remain; List<Integer> input){

if(remain < 0) return;

else if(remain == 0) list.add(new ArrayList<Integer>(current));

else{

for(int i = index; i < input.length; i++){

current.add(input.get(i));

addComb(current, list, i+1, input);

current.remove(current.size()-1);

}

}

}

**Combination Sum – without redundant**