**📝 Infinite Champions Programme – Day 6 (Assignment Sheet)**

**📌 Instructions  
• Deadline: Submit your solutions by 3rd October, 2025, EOD.  
• Platform: Test your solutions on LeetCode  
• Collaboration: Discussing concepts is encouraged, but all code must be your own.**

1. [**Word Search (79)**](https://leetcode.com/problems/word-search/)  
   • **Problem:** You are given a 2D board and a word. Find if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cells (horizontally or vertically). Each letter cell may not be used more than once.  
   • **Objective:** Use recursion and backtracking to explore all possible paths while pruning invalid ones.  
   • **YouTube Solution (Java):** [Word Search – Java Solution](https://www.youtube.com/watch?v=vYYNp0Jrdv0)

class Solution {

public boolean exist(char[][] board, String word) {

int n=board.length;

int m=board[0].length;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(board[i][j]==word.charAt(0)){

if(dfs(board,word,i,j,0)){

return true;

}

}

}

}

return false;

}

public boolean dfs(char[][] board,String word,int i,int j,int in){

if(in==word.length()){

return true;

}

if (i < 0 || i >= board.length || j < 0 || j >= board[0].length

|| board[i][j] != word.charAt(in)) {

return false;

}

char temp=board[i][j];

board[i][j]='\*';

boolean found = dfs(board, word, i + 1, j, in + 1) ||

dfs(board, word, i - 1, j, in + 1) ||

dfs(board, word, i, j + 1, in + 1) ||

dfs(board, word, i, j - 1, in + 1);

board[i][j]=temp;

return found;

}

}

1. [**Combinations (77)**](https://leetcode.com/problems/combinations/)  
   • **Problem:** Given two integers n and k, return all possible combinations of k numbers out of the range [1, n].  
   • **Objective:** Apply recursive backtracking to generate combinations, ensuring efficient pruning.  
   • **YouTube Solution (Java):** [Combinations – Java Solution](https://www.youtube.com/watch?v=FC0fa8Y0HbA)**’**

class Solution {

    public List<List<Integer>> combine(int n, int k) {

        List<List<Integer>> arr=new ArrayList<>();

        fun(1,n,k,new ArrayList<>(),arr);

        return arr;

    }

    public void fun(int s,int n,int k,List<Integer> p,List<List<Integer>> arr){

        if(p.size()==k){

            arr.add(new ArrayList<>(p));

            return;

        }

        for(int i=s;i<=n-(k-p.size())+1;i++){

            p.add(i);

            fun(i+1,n,k,p,arr);

            p.remove(p.size()-1);

        }

    }

}

1. [**Combination Sum II (40)**](https://leetcode.com/problems/combination-sum-ii/)  
   • **Problem:** Given a collection of candidate numbers (with possible duplicates) and a target, find all unique combinations where the numbers sum to target. Each number can be used at most once.  
   • **Objective:** Apply backtracking with pruning and duplicate handling using sorting.  
   • **YouTube Solution (Java):** [Combination Sum II – Java Solution](https://www.youtube.com/watch?v=rSA3t6BDDwg)

class Solution {

    public List<List<Integer>> combinationSum2(int[] candidates, int target) {

        List<List<Integer>> arr = new ArrayList<>();

        Arrays.sort(candidates);

        fun(candidates, target, 0, new ArrayList<>(),arr);

        return arr;

    }

    public void fun(int[] candidates,int t,int s,List<Integer> p,List<List<Integer>> arr){

        if(t==0){

            arr.add(new ArrayList<Integer>(p));

            return;

        }

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

            if(i>s && candidates[i]==candidates[i-1]){

                continue;

            }

            if(candidates[i]>t){

                break;

            }

            p.add(candidates[i]);

            fun(candidates,t-candidates[i],i+1,p,arr);

            p.remove(p.size()-1);

        }

    }

}

**📚 Submission Checklist  
• Time and space complexity analysis for each solution.  
• Test cases demonstrating the correctness of your solutions.**