Day 1

1. https://leetcode.com/problems/climbing-stairs/

Whenever we are given options in the problem, that is the hint that recursion can be used for this question.

```
You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you
climb to the top?

Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

```
public class Solution {
    public int ClimbStairs(int n) {
        return Climb(n, new Dictionary<int,int>());
    }

    private int Climb(int n, Dictionary<int, int> dict) {

        if(n == 0)
            return 1;
        else if(n < 0)
            return 0;

        if(dict.ContainsKey(n))
            return dict[n];

        var leftPath = Climb(n - 1, dict);
        var rightPath = Climb(n - 2, dict);

        dict[n] = leftPath + rightPath;

        return dict[n];
    }
}
```

---

2. https://leetcode.com/problems/min-cost-climbing-stairs/

```
You are given an integer array cost where cost[i] is the cost of ith step on a
staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Input: cost = [10,15,20]
Output: 15
Explanation: Cheapest is: start on cost[1], pay that cost, and go to the top.


Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: Cheapest is: start on cost[0], and only step on 1s, skipping
cost[3].
```

```
public class Solution {
    public int MinCostClimbingStairs(int[] cost) {

        var zero = MinCost(0, cost.Length, cost, new Dictionary<int,int>());
        var one = MinCost(1, cost.Length, cost, new Dictionary<int,int>());

        return Math.Min(zero, one);
    }

    private int MinCost(int current, int target, int[] cost, Dictionary<int,
int> results){

        if(current == target){
            return 0;
        }

        if(current > target)
            return 10000000; // int.MaxValue does not work here.. it overflows
somehow

        if(results.ContainsKey(current))
            return results[current];
```

```
        var oneStepCost = cost[current] + MinCost(current + 1, target, cost,
results);
        var twoStepCost = cost[current] + MinCost(current + 2, target, cost,
results);

        results[current] = Math.Min(oneStepCost, twoStepCost);
        return results[current];
    }
}
```

---

3. https://leetcode.com/problems/house-robber/

```
You are a professional robber planning to rob houses along a street. Each house
has a certain amount of money stashed, the only constraint stopping you from
robbing each of them is that adjacent houses have security systems connected and
it will automatically contact the police if two adjacent houses were broken into
on the same night.

Given an integer array nums representing the amount of money of each house,
return the maximum amount of money you can rob tonight without alerting the
police.

Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5
(money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

Recursive approach:

```
public class Solution {
    public int Rob(int[] nums) {
        return RobHouse(0, nums, new Dictionary<int, int> ());
    }

    private int RobHouse(int current, int[] nums, Dictionary<int, int> memo)
    {

        if(current >= nums.Length){
            return 0;
        }
```

```
        if(memo.ContainsKey(current))
            return memo[current];

        var ifRobbedProfit = nums[current] + RobHouse(current + 2, nums, memo);
        var ifNotRobbedProfit = RobHouse(current + 1, nums, memo);

        memo[current] = Math.Max(ifRobbedProfit, ifNotRobbedProfit);

        return memo[current];
    }
}
```

iterative approach:

```
public class Solution {
    public int Rob(int[] nums) {

        if(nums.Length == 1)
            return nums[0];

        if(nums.Length == 2)
            return Math.Max(nums[0], nums[1]);

        var maxRobbery = new int[nums.Length];
        maxRobbery[0] = nums[0];
        maxRobbery[1] = Math.Max(nums[0], nums[1]);

        for(int idx = 2; idx < nums.Length; idx++){
            var current = nums[idx];

            if(current + maxRobbery[idx - 2] > maxRobbery[idx - 1])
                maxRobbery[idx] = current + maxRobbery[idx - 2];
            else
                maxRobbery[idx] = maxRobbery[idx - 1];
        }

        return maxRobbery[nums.Length - 1];

    }
}
```

More optimal approach (space complexity wise)

```
public class Solution {
    public int Rob(int[] nums) {

        if(nums.Length == 1)
```

```
            return nums[0];

        if(nums.Length == 2)
            return Math.Max(nums[0], nums[1]);

        var firstRobbery = nums[0];
        var secondRobbery = Math.Max(nums[0], nums[1]);

        for(int idx = 2; idx < nums.Length; idx++){
            var current = nums[idx];
            var currentRobbery = 0;
            if(current + firstRobbery > secondRobbery)
                currentRobbery = current + firstRobbery;
            else
                currentRobbery = secondRobbery;

            firstRobbery = secondRobbery;
            secondRobbery = currentRobbery;
        }

        return secondRobbery;

    }
}
```

Day 2

4. https://leetcode.com/problems/fibonacci-number/

```
The Fibonacci numbers, commonly denoted F(n) form a sequence, called the
Fibonacci sequence, such that each number is the sum of the two preceding ones,
starting from 0 and 1. That is,

F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
Given n, calculate F(n).

Input: n = 4
Output: 3
```

```
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
```

Normal approach:

```
public class Solution {
    public int Fib(int n) {
        if( n == 0)
            return 0;
        if( n == 1)
            return 1;
        return Fib(n-1) + Fib(n-2);
    }
}
```

With DP:

```
public class Solution {
    public int Fib(int n) {

        return FibHelper(n, new Dictionary<int, int>());
    }

    private int FibHelper(int current, Dictionary<int, int> memo) {

        if(current == 0)
            return 0;

        if(current == 1)
            return 1;

        memo[current] = FibHelper(current - 1, memo) +
                        FibHelper(current - 2, memo);

        return memo[current];
    }
}
```

5. https://leetcode.com/problems/n-th-tribonacci-number/

```
The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.
```

```
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
```

```
public class Solution {
    public int Tribonacci(int n) {
        return FindTribonacci(n, new Dictionary<int, int>());
    }

    private int FindTribonacci(int current, Dictionary<int,int> memo){
        if(current == 0)
            return 0;

        if(current == 1 || current == 2)
            return 1;

        if(memo.ContainsKey(current))
            return memo[current];

        memo[current] = FindTribonacci(current - 1, memo) +
                        FindTribonacci(current - 2, memo) +
                        FindTribonacci(current - 3, memo);
        return memo[current];
    }
}
```

6. https://leetcode.com/problems/counting-bits/

```
Given an integer n, return an array ans of length n + 1 such that for each i (0
<= i <= n), ans[i] is the number of 1's in the binary representation of i.

Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
```

Recursive approach with O(nLogn) complexity:

```
public class Solution {
    public int[] CountBits(int n) {

        var output = new int[n + 1];

        for(int idx = 0; idx <= n; idx++){
            output[idx] = GetNumberOfOnes(idx);
        }

        return output;
    }

    private int GetNumberOfOnes(int n){

        if(n == 0)
            return 0;

        if(n == 1)
            return 1;

        return n % 2 + GetNumberOfOnes(n/2);
    }
}
```

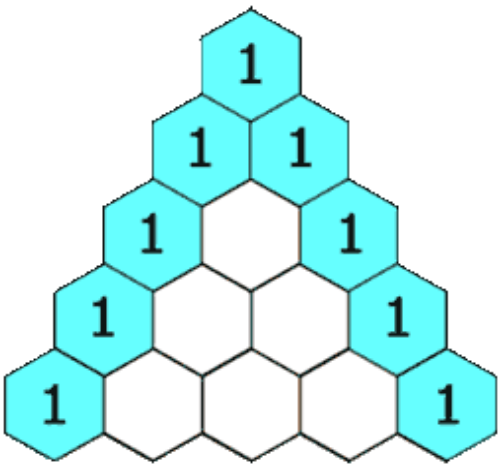
Optimal approach with O(n) complexity:

```
public class Solution {
    public int[] CountBits(int n) {

        var output = new int[n + 1];

        for(int idx = 0; idx <= n; idx++)
            output[idx] = idx%2 + output[idx / 2];

        return output;
    }
}
```

7. https://leetcode.com/problems/pascals-triangle/

```
Given an integer numRows, return the first numRows of Pascal's triangle.
```

```
In Pascal's triangle, each number is the sum of the two numbers directly above
it as shown:

Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

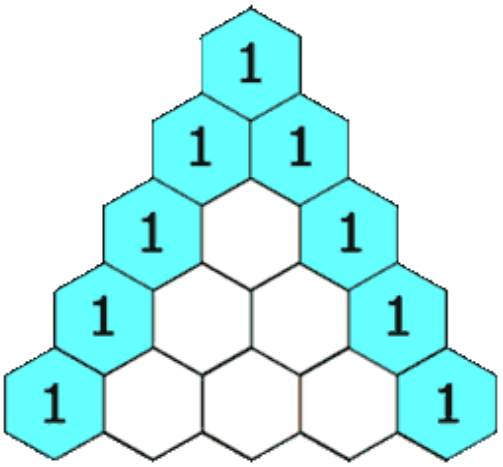



```
public class Solution {
    public IList<IList<int>> Generate(int numRows) {

        IList<IList<int>> output = new List<IList<int>>();

        var initialList = new List<int> {1};
        output.Add(initialList);

        if(numRows == 1)
            return output;

        for(int idx = 2; idx <= numRows; idx++){

            var previousArray = output[output.Count() - 1];
            var currentList = new List<int>();
            currentList.Add(previousArray[0]);
            for(int innerIdx = 0; innerIdx < previousArray.Count() - 1;
innerIdx++){
                currentList.Add(previousArray[innerIdx] + previousArray[innerIdx
+ 1]);
            }

            currentList.Add(previousArray[previousArray.Count() - 1]);

            output.Add(currentList);
        }

        return output;
    }
```

```
}
```

8. https://leetcode.com/problems/pascals-triangle-ii/

```
Given an integer rowIndex, return the rowIndexth (0-indexed) row of the Pascal's
triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above
it as shown:

Input: rowIndex = 3
Output: [1,3,3,1]
```




```
public class Solution {
    public IList<int> GetRow(int rowIndex) {

        var output = new int[rowIndex + 1];
        output[0] = 1;

        if(rowIndex == 0)
            return output;

        for(int idx = 1; idx <= rowIndex; idx++){
            long val = output[idx - 1]; // to prevent overflow // it might
happen because of multiplication
            val = (val * (rowIndex - idx + 1)) / idx;
            output[idx] = (int)val;
        }
```

```
        return output.ToList();
    }
}
```

Day 3

9. https://practice.geeksforgeeks.org/problems/0-1-knapsack-problem0945/1

```
You are given weights and values of N items, put these items in a knapsack of
capacity W to get the maximum total value in the knapsack. Note that we have
only one quantity of each item.
In other words, given two integer arrays val[0..N-1] and wt[0..N-1] which
represent values and weights associated with N items respectively. Also given an
integer W which represents knapsack capacity, find out the maximum value subset
of val[] such that sum of the weights of this subset is smaller than or equal to
W. You cannot break an item, either pick the complete item or don't pick it (0-1
property).

Input:
N = 3
W = 4
values[] = {1,2,3}
weight[] = {4,5,1}
Output: 3
```

```
// { Driver Code Starts
//Initial Template for C#

using System;
using System.Numerics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DriverCode
{

    class GFG
    {
        static void Main(string[] args)
        {
            int testcases;// Taking testcase as input
```

```
            testcases = Convert.ToInt32(Console.ReadLine());
            while (testcases-- > 0)// Looping through all testcases
            {
                int N = Convert.ToInt32(Console.ReadLine());
                int[] val = new int[N];
                int[] wt = new int[N];
                int K = Convert.ToInt32(Console.ReadLine());
                string elements = Console.ReadLine().Trim();
                elements = elements + " " + "0";
                val = Array.ConvertAll(elements.Split(), int.Parse);
                elements = Console.ReadLine().Trim();
                elements = elements + " " + "0";
                wt = Array.ConvertAll(elements.Split(), int.Parse);
                Solution obj = new Solution();
                int res = obj.knapSack(K,wt,val, N);
                Console.Write(res+"\n");
          }

        }
    }
}
// } Driver Code Ends


//User function Template for C#


class Solution
    {
        //Complete this function
        public int knapSack(int W, int[] val, int[] wt,int n)
        {
            //Your code here

            return knapSackHelper(0, W, val, wt);
        }

        private int knapSackHelper(int currentIndex, int currentCapacity, int[]
val, int[] wt){

            if(currentIndex == val.Length)
                return 0;

            var currentWeight = val[currentIndex];

            var consideredProfit = 0;
            if(currentWeight <= currentCapacity){

                consideredProfit = wt[currentIndex] +
                                    knapSackHelper(currentIndex + 1,
currentCapacity - currentWeight, val, wt);
```

```
            }

            var notConsideredProfit = knapSackHelper(currentIndex + 1,
currentCapacity, val, wt);

            return Math.Max(consideredProfit, notConsideredProfit);
        }

    }
```

With DP:

```
// { Driver Code Starts
//Initial Template for C#

using System;
using System.Numerics;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DriverCode
{

    class GFG
    {
        static void Main(string[] args)
        {
            int testcases;// Taking testcase as input
            testcases = Convert.ToInt32(Console.ReadLine());
            while (testcases-- > 0)// Looping through all testcases
            {
                int N = Convert.ToInt32(Console.ReadLine());
                int[] val = new int[N];
                int[] wt = new int[N];
                int K = Convert.ToInt32(Console.ReadLine());
                string elements = Console.ReadLine().Trim();
                elements = elements + " " + "0";
                val = Array.ConvertAll(elements.Split(), int.Parse);
                elements = Console.ReadLine().Trim();
                elements = elements + " " + "0";
                wt = Array.ConvertAll(elements.Split(), int.Parse);
                Solution obj = new Solution();
                int res = obj.knapSack(K,wt,val, N);
                Console.Write(res+"\n");
          }

        }
```

```
    }
}
// } Driver Code Ends


//User function Template for C#


class Solution
    {
        //Complete this function
        public int knapSack(int W, int[] val, int[] wt,int n)
        {
            //Your code here

            return knapSackHelper(0, W, val, wt, new Dictionary<string, int>());
        }

        private int knapSackHelper(int currentIndex, int currentCapacity, int[]
val, int[] wt,
        Dictionary<string, int> memo){

            if(currentIndex == val.Length)
                return 0;

            var currentVal = wt[currentIndex];
            var currentWeight = val[currentIndex];

            var currentKey = $"{currentIndex}:{currentCapacity}";
            if(memo.ContainsKey(currentKey))
                return memo[currentKey];

            var consideredProfit = 0;
            if(currentWeight <= currentCapacity){

                consideredProfit = currentVal +
                                    knapSackHelper(currentIndex + 1,
currentCapacity - currentWeight,
                                    val, wt, memo);
            }

            var notConsideredProfit = knapSackHelper(currentIndex + 1,
currentCapacity,
            val, wt, memo);

            memo[currentKey] = Math.Max(consideredProfit, notConsideredProfit);
            return memo[currentKey];
        }

    }
```

10. https://leetcode.com/problems/target-sum/

```
You are given an integer array nums and an integer target.

You want to build an expression out of nums by adding one of the symbols '+' and
'-' before each integer in nums and then concatenate all the integers.

For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and
concatenate them to build the expression "+2-1".
Return the number of different expressions that you can build, which evaluates
to target.

Input: nums = [1,1,1,1,1], target = 3
Output: 5
Explanation: There are 5 ways to assign symbols to make the sum of nums be
target 3.
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
```

```
public class Solution {
    public int FindTargetSumWays(int[] nums, int target) {

        return FindTargethelper(0, target, nums);
    }

    private int FindTargethelper(int current, int target, int[] nums){

        if(target == 0 && current == nums.Length)
            return 1;

        if(current > nums.Length - 1)
            return 0;

        var currentItem = nums[current];
        var ifPositive = FindTargethelper(current + 1, target - currentItem,
nums);
        var ifNegative = FindTargethelper(current + 1, target + currentItem,
nums);

        return ifPositive + ifNegative;
```

```
    }
}
```

With DP:

```
public class Solution {
    public int FindTargetSumWays(int[] nums, int target) {

        return FindTargethelper(0, target, nums, new Dictionary<string, int>());
    }

    private int FindTargethelper(int current, int target, int[] nums,
                                 Dictionary<string, int> memo){

        if(target == 0 && current == nums.Length)
            return 1;

        if(current > nums.Length - 1)
            return 0;

        var currentKey = $"{current}:{target}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        var currentItem = nums[current];
        var ifPositive = FindTargethelper(current + 1, target - currentItem,
nums, memo);
        var ifNegative = FindTargethelper(current + 1, target + currentItem,
nums, memo);

        memo[currentKey] = ifPositive + ifNegative;

        return memo[currentKey];
    }
}
```

11. https://leetcode.com/problems/partition-equal-subset-sum/

```
Given a non-empty array nums containing only positive integers, find if the
array can be partitioned into two subsets such that the sum of elements in both
subsets is equal.

Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

```
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into equal sum subsets.
```

This is not a complete solution.. Time limit is exceeding in this (89 cases passed out of 116).

```
public class Solution {
    public bool CanPartition(int[] nums) {
        var sum = 0;
        foreach(var num in nums)
            sum += num;

        if(sum % 2 == 1)
            return false;

        var target = sum / 2;

        return SubsetFinder(0, target, nums, new Dictionary<string, bool> ());
    }

    private bool SubsetFinder(int current, int target, int[] nums,
Dictionary<string, bool> memo){

        if(target == 0)
            return true;

        if(current >= nums.Length || target < 0)
            return false;

        var currentKey = $"{current}:{target}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        var considered = SubsetFinder(current + 1, target - nums[current], nums,
memo);
        var notConsidered = SubsetFinder(current + 1, target, nums, memo);

        memo[currentKey] = considered || notConsidered;

        return memo[currentKey];
    }
}
```

Day 4

12. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

```
You are given an array prices where prices[i] is the price of a given stock on
the ith day.

You want to maximize your profit by choosing a single day to buy one stock and
choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot
achieve any profit, return 0.

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-
1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must
buy before you sell.
```

With array: O(n) space complexity

```
public class Solution {
    public int MaxProfit(int[] prices) {

        if(prices.Length == 1)
            return 0;

        var profits = new int[prices.Length];

        var boughtPrice = prices[0];
        for(int idx = 1; idx < prices.Length; idx++){

            profits[idx] = (prices[idx] > boughtPrice)  && (profits[idx - 1] <
prices[idx] - boughtPrice)
                ? prices[idx] - boughtPrice
                : profits[idx - 1];

            if(boughtPrice > prices[idx])
                boughtPrice = prices[idx];
        }

        return profits[profits.Length - 1];
    }
}
```

With constant space:

```
public class Solution {
    public int MaxProfit(int[] prices) {

        if(prices.Length == 1)
            return 0;

        var currentProfit = 0;
        var boughtPrice = prices[0];

        for(int idx = 1; idx < prices.Length; idx++){

            if((prices[idx] > boughtPrice)  && (currentProfit < prices[idx] -
boughtPrice) ){
                currentProfit =  prices[idx] - boughtPrice;
            }

            if(boughtPrice > prices[idx])
                boughtPrice = prices[idx];
        }

        return currentProfit;
    }
}
```

Day 5

13. https://leetcode.com/problems/coin-change-2/

```
You are given an integer array coins representing coins of different
denominations and an integer amount representing a total amount of money.

Return the number of combinations that make up that amount. If that amount of
money cannot be made up by any combination of the coins, return 0.

You may assume that you have an infinite number of each kind of coin.

The answer is guaranteed to fit into a signed 32-bit integer.

Input: amount = 5, coins = [1,2,5]
Output: 4
```

```
Explanation: there are four ways to make up the amount:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
```

```
public class Solution {
    public int Change(int amount, int[] coins) {
        return TotalWays(coins, 0, amount, new Dictionary<string, int>());
    }

    private int TotalWays(int[] coins, int current, int amount,
Dictionary<string, int> memo){

        if(amount == 0)
            return 1;

        if(current >= coins.Length)
            return 0;

        var currentKey = $"{current}:{amount}";

        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        var currentAmount = coins[current];

        var considered = 0;
        if(coins[current] <= amount)
            considered = TotalWays(coins, current, amount - currentAmount, memo);
        var notConsidered = TotalWays(coins, current + 1, amount, memo);

        memo[currentKey] = considered + notConsidered;

        return memo[currentKey];
    }
}
```

14. https://leetcode.com/problems/longest-common-subsequence/

```
Given two strings text1 and text2, return the length of their longest common
subsequence. If there is no common subsequence, return 0.

A subsequence of a string is a new string generated from the original string
with some characters (can be none) deleted without changing the relative order
of the remaining characters.
```

```
For example, "ace" is a subsequence of "abcde".
A common subsequence of two strings is a subsequence that is common to both
strings.

Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.

Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

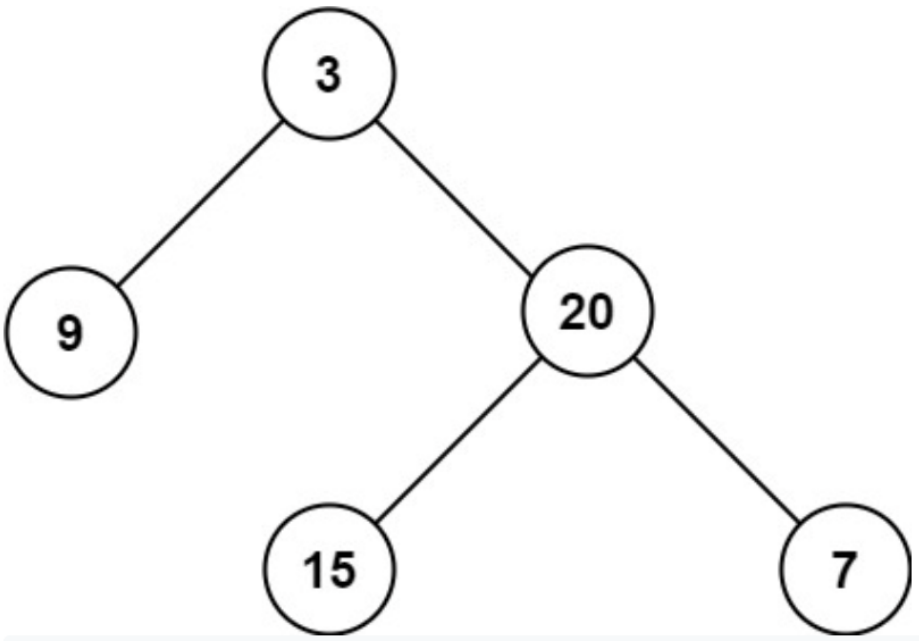
```
public class Solution {
    public int LongestCommonSubsequence(string text1, string text2) {
        return FindLongest(text1, text2, 0, 0, new Dictionary<string, int>());
    }

    private int FindLongest(string text1, string text2, int currentInText1, int
currentInText2,
                            Dictionary<string, int> memo){

        if(currentInText1 >= text1.Length || currentInText2 >= text2.Length)
            return 0;

        var currentKey = $"{currentInText1}:{currentInText2}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(text1[currentInText1] == text2[currentInText2])
            return 1 + FindLongest(text1, text2, currentInText1 + 1,
currentInText2 + 1, memo);

        else
           memo[currentKey] = Math.Max(FindLongest(text1, text2, currentInText1
+ 1, currentInText2, memo),
            FindLongest(text1, text2, currentInText1, currentInText2 + 1, memo));

        return memo[currentKey];
    }
}
```

15. https://leetcode.com/problems/maximum-depth-of-binary-tree/

```
Given the root of a binary tree, return its maximum depth.
```

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:



Input: root = [3,9,20,null,null,15,7]
Output: 3

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public int MaxDepth(TreeNode root) {

        if(root == null)
```

```
            return 0;

        var leftHeight = 1 + MaxDepth(root.left);
        var rightHeight = 1 + MaxDepth(root.right);

        return Math.Max(leftHeight, rightHeight);

    }
}
```

Problems so far

1. Climb stairs
2. Min cost to climb stairs
3. House robber
4. Fibonacci
5. Tribonacci
6. 0-1 knapsack
7. Subset Sum Problem
8. Target Sum
9. Subset Sum equal to given diff
10. Rod cutting
11. Coin change 1
12. Coin change 2
13. LCS

Day 6

16. https://leetcode.com/problems/longest-palindromic-subsequence/

```
Given a string s, find the longest palindromic subsequence's length in s.

A subsequence is a sequence that can be derived from another sequence by
deleting some or no elements without changing the order of the remaining
elements.

Input: s = "bbbab"
```

```
Output: 4
Explanation: One possible longest palindromic subsequence is "bbbb".

Input: s = "cbbd"
Output: 2
Explanation: One possible longest palindromic subsequence is "bb".
```

```
public class Solution {
    public int LongestPalindromeSubseq(string s) {

        var reverseS = new StringBuilder();
        for (int idx = s.Length - 1; idx >= 0; idx--) {
            reverseS.Append(s[idx]);
        }
        //Console.WriteLine($"reverseS: {reverseS}");
        return FindLongest(s, reverseS.ToString(), 0, 0, new Dictionary<string,
int>());

    }

    private int FindLongest(string text1, string text2, int currentInText1, int
currentInText2,
                            Dictionary<string, int> memo){

        if(currentInText1 >= text1.Length || currentInText2 >= text2.Length)
            return 0;

        var currentKey = $"{currentInText1}:{currentInText2}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(text1[currentInText1] == text2[currentInText2])
            memo[currentKey] = 1 + FindLongest(text1, text2, currentInText1 + 1,
currentInText2 + 1, memo);

        else
           memo[currentKey] = Math.Max(FindLongest(text1, text2, currentInText1
+ 1, currentInText2, memo),
            FindLongest(text1, text2, currentInText1, currentInText2 + 1, memo));

        return memo[currentKey];
    }
}
```

Day 7

17. https://leetcode.com/problems/jump-game/

```
You are given an integer array nums. You are initially positioned at the array's
first index, and each element in the array represents your maximum jump length
at that position.

Return true if you can reach the last index, or false otherwise.

Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump
length is 0, which makes it impossible to reach the last index.
```

```
public class Solution {
    public bool CanJump(int[] nums) {
        return FindPath(nums, 0, new Dictionary<int, bool>());
    }

    private bool FindPath(int[] nums, int currentIndex, Dictionary<int, bool>
memo){

        if(currentIndex >= nums.Length - 1)
            return true;

        if(memo.ContainsKey(currentIndex))
            return memo[currentIndex];

        var currentOptions = nums[currentIndex];

        for(int idx = 1; idx <= currentOptions; idx++){
            if(FindPath(nums, currentIndex + idx, memo)){
                memo[currentIndex] = true;
                return true;
            }
        }

        memo[currentIndex] = false;
```

```
        return false;
    }
}
```

18. https://leetcode.com/problems/jump-game-ii/

```
Given an array of non-negative integers nums, you are initially positioned at
the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1
step from index 0 to 1, then 3 steps to the last index.

Input: nums = [2,3,0,1,4]
Output: 2
```

```
public class Solution {

    public int Jump(int[] nums) {

        return FindMinJumps(nums, 0, new Dictionary<int, int> ());

    }

    private int FindMinJumps(int[] nums, int current, Dictionary<int, int> memo){

        //Console.WriteLine($"current : {current}, currentSteps: {currentSteps},
minSteps: {minSteps}");

        if(current >= nums.Length - 1)
            return 0;

        if(memo.ContainsKey(current))
            return memo[current];

        var minSteps = 5753; // just a number greater than limits.
        var currentOptions = nums[current];

        for(int idx = 1; idx <= currentOptions; idx++){
```

```
            var numberOfSteps = 1 + FindMinJumps(nums, current + idx, memo);
            if(numberOfSteps < minSteps)
                minSteps = numberOfSteps;
        }

        memo[current] = minSteps;

        return minSteps;
    }
}
```

19. https://leetcode.com/problems/get-maximum-in-generated-array/

```
You are given an integer n. An array nums of length n + 1 is generated in the
following way:

nums[0] = 0
nums[1] = 1
nums[2 * i] = nums[i] when 2 <= 2 * i <= n
nums[2 * i + 1] = nums[i] + nums[i + 1] when 2 <= 2 * i + 1 <= n
Return the maximum integer in the array nums.

Input: n = 7
Output: 3
Explanation: According to the given rules:
  nums[0] = 0
  nums[1] = 1
  nums[(1 * 2) = 2] = nums[1] = 1
  nums[(1 * 2) + 1 = 3] = nums[1] + nums[2] = 1 + 1 = 2
  nums[(2 * 2) = 4] = nums[2] = 1
  nums[(2 * 2) + 1 = 5] = nums[2] + nums[3] = 1 + 2 = 3
  nums[(3 * 2) = 6] = nums[3] = 2
  nums[(3 * 2) + 1 = 7] = nums[3] + nums[4] = 2 + 1 = 3
Hence, nums = [0,1,1,2,1,3,2,3], and the maximum is 3.
```

```
public class Solution {
    public int GetMaximumGenerated(int n) {

        if(n <= 1)
            return n;

        var output = new int[n + 1];
        output[0] = 0;
        output[1] = 1;
```

```
        for(int idx = 2; idx <= n; idx++)
            output[idx] = output[idx / 2] + ( output[ idx - (idx / 2)] * (idx %
2));

        return output.Max();
    }
}
```

20. https://leetcode.com/problems/counting-bits/

```
Given an integer n, return an array ans of length n + 1 such that for each i (0
<= i <= n), ans[i] is the number of 1's in the binary representation of i.

Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
```

```
public class Solution {
    public int[] CountBits(int n) {

        var output = new int[n + 1];

        for(int idx = 0; idx <= n; idx++)
            output[idx] = idx%2 + output[idx / 2];

        return output;
    }
}
```

Day 8 (Important One)

Following questions is a repetition from day 4.

21. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

```
You are given an array prices where prices[i] is the price of a given stock on
the ith day.

You want to maximize your profit by choosing a single day to buy one stock and
choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot
achieve any profit, return 0.

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-
1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must
buy before you sell.
```

With array: O(n) space complexity

```
public class Solution {
    public int MaxProfit(int[] prices) {

        if(prices.Length == 1)
            return 0;

        var profits = new int[prices.Length];

        var boughtPrice = prices[0];
        for(int idx = 1; idx < prices.Length; idx++){

            profits[idx] = (prices[idx] > boughtPrice)  && (profits[idx - 1] <
prices[idx] - boughtPrice)
                ? prices[idx] - boughtPrice
                : profits[idx - 1];

            if(boughtPrice > prices[idx])
                boughtPrice = prices[idx];
        }

        return profits[profits.Length - 1];
    }
}
```

With constant space:

```
public class Solution {
```

```
    public int MaxProfit(int[] prices) {

        if(prices.Length == 1)
            return 0;

        var currentProfit = 0;
        var boughtPrice = prices[0];

        for(int idx = 1; idx < prices.Length; idx++){

            if((prices[idx] > boughtPrice)  && (currentProfit < prices[idx] -
boughtPrice) ){
                currentProfit =  prices[idx] - boughtPrice;
            }

            if(boughtPrice > prices[idx])
                boughtPrice = prices[idx];
        }

        return currentProfit;
    }
}
```

With recursive approach:

```
public class Solution {
    public int MaxProfit(int[] prices) {
        return CalculateMaxprofit(prices, 0, true, 1, new Dictionary<string,
int> ());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
int transactionCount, Dictionary<string, int> memo){

        if(currentDay >= prices.Length || transactionCount <= 0)
            return 0;

        var currentKey = $"{currentDay}:{canBuy}:{transactionCount}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){

            if(currentDay >= prices.Length - 1)
                return 0;

            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, transactionCount, memo);
```

```
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var selling = prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, true, transactionCount - 1, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

22. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/

```
You are given an array prices where prices[i] is the price of a given stock on
the ith day.

Find the maximum profit you can achieve. You may complete at most two
transactions.

Note: You may not engage in multiple transactions simultaneously (i.e., you must
sell the stock before you buy again).

Input: prices = [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-
0 = 3.
Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.
```

// not passing all the test cases.

```
public class Solution {
    public int MaxProfit(int[] prices) {
        return CalculateMaxprofit(prices, 0, true, 2, new Dictionary<string,
int> ());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
int transactionCount, Dictionary<string, int> memo){

        if(currentDay >= prices.Length || transactionCount <= 0)
            return 0;
```

```
        var currentKey = $"{currentDay}:{canBuy}:{transactionCount}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){

            if(currentDay >= prices.Length - 1)
                return 0;

            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, transactionCount, memo);
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var selling = prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, true, transactionCount - 1, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

23. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

```
You are given an integer array prices where prices[i] is the price of a given
stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at
most one share of the stock at any time. However, you can buy it then
immediately sell it on the same day.

Find and return the maximum profit you can achieve.

Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-
1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.
```

```
public class Solution {
    public int MaxProfit(int[] prices) {
        return CalculateMaxprofit(prices, 0, true, new Dictionary<string, int>
());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
Dictionary<string, int> memo){

        if(currentDay >= prices.Length )
            return 0;

        var currentKey = $"{currentDay}:{canBuy}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, memo);
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, memo);
            var selling = prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, true, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

24. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/

```
You are given an integer array prices where prices[i] is the price of a given
stock on the ith day, and an integer k.

Find the maximum profit you can achieve. You may complete at most k transactions.

Note: You may not engage in multiple transactions simultaneously (i.e., you must
sell the stock before you buy again).

Input: k = 2, prices = [3,2,6,5,0,3]
Output: 7
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-
2 = 4. Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0
```

```
= 3.

Input: k = 2, prices = [2,4,1]
Output: 2
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-
2 = 2.
```

```
public class Solution {
    public int MaxProfit(int k, int[] prices) {
        return CalculateMaxprofit(prices, 0, true, k, new Dictionary<string, int>
());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
int transactionCount, Dictionary<string, int> memo){

        if(currentDay >= prices.Length || transactionCount <= 0)
            return 0;

        var currentKey = $"{currentDay}:{canBuy}:{transactionCount}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){

            if(currentDay >= prices.Length - 1)
                return 0;

            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, transactionCount, memo);
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy,
transactionCount, memo);
            var selling = prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, true, transactionCount - 1, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

25. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/

```
You are given an array prices where prices[i] is the price of a given stock on
the ith day.

Find the maximum profit you can achieve. You may complete as many transactions
as you like (i.e., buy one and sell one share of the stock multiple times) with
the following restrictions:

After you sell your stock, you cannot buy stock on the next day (i.e., cooldown
one day).
Note: You may not engage in multiple transactions simultaneously (i.e., you must
sell the stock before you buy again).


Input: prices = [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell, cooldown, buy, sell]
```

```
public class Solution {
    public int MaxProfit(int[] prices) {
        return CalculateMaxprofit(prices, 0, true, new Dictionary<string, int>
());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
Dictionary<string, int> memo){

        if(currentDay >= prices.Length)
            return 0;

        var currentKey = $"{currentDay}:{canBuy}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){

            if(currentDay >= prices.Length - 1)
                return 0;

            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, memo);
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
```

```
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, memo);
            var selling = prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 2, true, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

26. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/

```
You are given an array prices where prices[i] is the price of a given stock on
the ith day, and an integer fee representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions
as you like, but you need to pay the transaction fee for each transaction.

Note: You may not engage in multiple transactions simultaneously (i.e., you must
sell the stock before you buy again).

Input: prices = [1,3,2,8,4,9], fee = 2
Output: 8
Explanation: The maximum profit can be achieved by:
- Buying at prices[0] = 1
- Selling at prices[3] = 8
- Buying at prices[4] = 4
- Selling at prices[5] = 9
The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.


Input: prices = [1,3,7,5,10,3], fee = 3
Output: 6
```

```
public class Solution {
    public int MaxProfit(int[] prices, int fee) {
        return CalculateMaxprofit(prices, 0, true, fee, new Dictionary<string,
int>());
    }

    private int CalculateMaxprofit(int[] prices, int currentDay, bool canBuy,
int fee, Dictionary<string, int> memo){

        if(currentDay >= prices.Length)
```

```
            return 0;

        var currentKey = $"{currentDay}:{canBuy}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        if(canBuy){

            if(currentDay >= prices.Length - 1)
                return 0;

            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, fee,
memo);
            var buy = -prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, false, fee, memo);
            memo[currentKey] = Math.Max(idle, buy);
        }
        else{
            var idle = CalculateMaxprofit(prices, currentDay + 1, canBuy, fee,
memo);
            var selling = -fee + prices[currentDay] + CalculateMaxprofit(prices,
currentDay + 1, true, fee, memo);
            memo[currentKey] = Math.Max(idle, selling);
        }

        return memo[currentKey];
    }
}
```

Sunday Revision

Trees 1

1. https://leetcode.com/problems/merge-two-binary-trees/

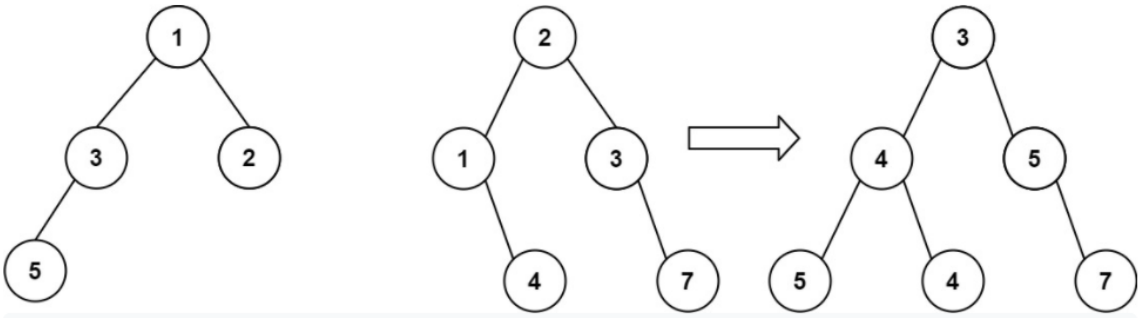
You are given two binary trees root1 and root2.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return the merged tree.

Note: The merging process must start from the root nodes of both trees.

Example 1:



Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public TreeNode MergeTrees(TreeNode root1, TreeNode root2) {

         return Merge(root1, root2);
    }
```

```
    private TreeNode Merge(TreeNode root1, TreeNode root2){

        // this can be commented becuase this can be handled by the other base
cases/
//         if(root1 == null && root2 == null)
//             return null;

        if (root1 == null)
            return root2;
        if (root2 == null){
            return root1;
        }

        var root = new TreeNode(root1.val + root2.val);
        root.left = Merge(root1.left, root2.left);
        root.right = Merge(root1.right, root2.right);

        return root;
    }
}
```

```python
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sumRootToLeaf(self, root: Optional[TreeNode]) -> int:
        return self.sumRTL(root, "" + str(root.val))

    def sumRTL(self, root, path):

        if root.left is None and root.right is None:
            return int(path, 2)

        leftAns = rightAns = 0

        if root.left:
            leftAns = self.sumRTL(root.left, path + str(root.left.val))

        if root.right:
            rightAns = self.sumRTL(root.right, path + str(root.right.val))

        return leftAns + rightAns
```

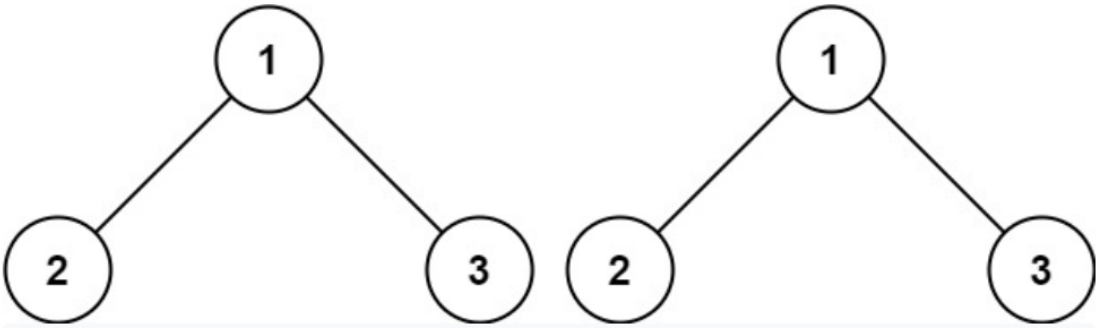
```
#         self.left = left
#         self.right = right
class Solution:
    def sumNumbers(self, root: TreeNode) -> int:
        return self.sumRTL(root, root.val)

    def sumRTL(self, root, path):

        if root.left is None and root.right is None:
            return path

        leftAns = rightAns = 0

        if root.left:
            leftAns = self.sumRTL(root.left, path*10 + root.left.val)

        if root.right:
            rightAns = self.sumRTL(root.right, path*10 + root.right.val)

        return leftAns + rightAns
```

Trees 2

1. https://leetcode.com/problems/same-tree/

```
Given the roots of two binary trees p and q, write a function to check if they
are the same or not.

Two binary trees are considered the same if they are structurally identical, and
the nodes have the same value.
```

Example 1:




```
Input: p = [1,2,3], q = [1,2,3]
Output: true
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool IsSameTree(TreeNode p, TreeNode q) {

        if(p == null && q == null)
            return true;

        if((p == null && q != null) ||(p != null && q == null) )
            return false;

        if(p.val == q.val)
            return IsSameTree(p.left, q.left) && IsSameTree(p.right, q.right);
        else
            return false;
    }
}
```

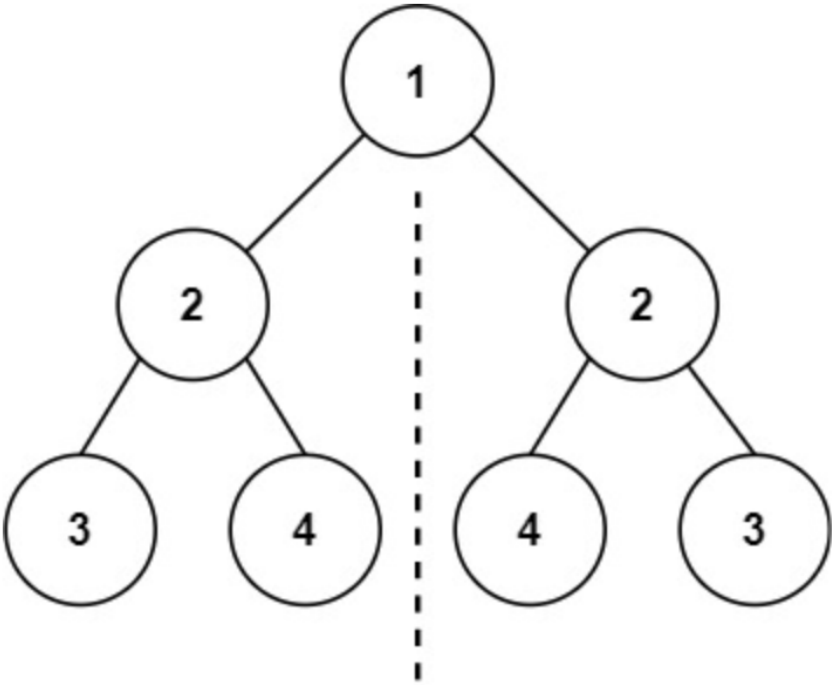
```
/**
```

```
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool IsSameTree(TreeNode p, TreeNode q) {

        if(p == null && q == null)
            return true;

        if(p == null || q == null)
            return false;

        if(p.val != q.val)
            return false;

        return IsSameTree(p.left, q.left) && IsSameTree(p.right, q.right);;
    }
}
```

2. https://leetcode.com/problems/symmetric-tree/

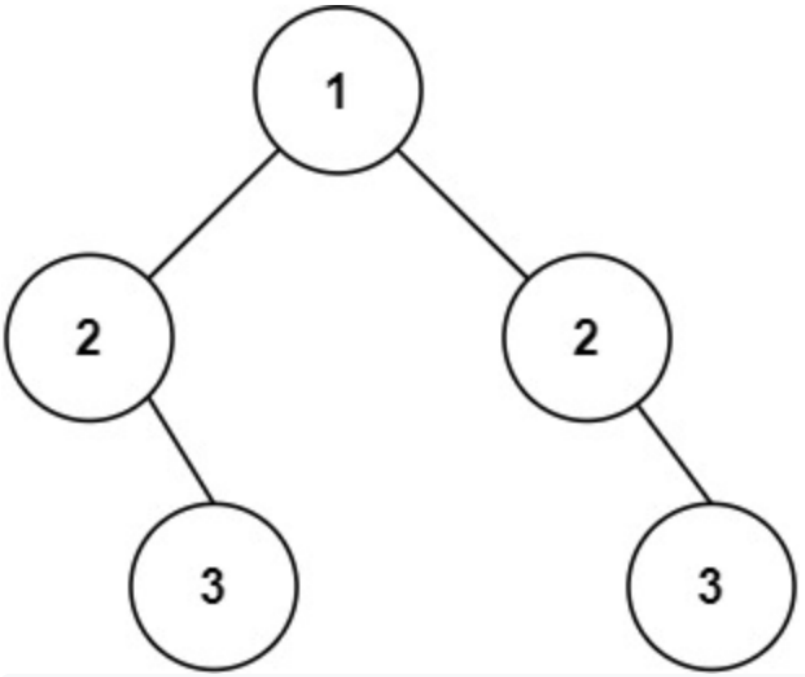
```
Given the root of a binary tree, check whether it is a mirror of itself (i.e.,
symmetric around its center).
```

Example 1:




```
Input: root = [1,2,2,3,4,4,3]
Output: true
```

Example 2:



Input: root = [1,2,2,null,3,null,3]
Output: false

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool IsSymmetric(TreeNode root) {

        return CheckSymmetric(root.left, root.right);
    }

    private bool CheckSymmetric(TreeNode left, TreeNode right){

        if(left == null && right == null)
            return true;
```

```
        if(left == null || right == null)
            return false;

        if(left.val != right.val)
            return false;

        return CheckSymmetric(left.left,right.right) &&
CheckSymmetric(left.right, right.left);
    }
}
```

3. https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1

```
Given a Binary Tree, print Left view of it. Left view of a Binary Tree is set of
nodes visible when tree is visited from Left side. The task is to complete the
function leftView(), which accepts root of the tree as argument.

Left view of following tree is 1 2 4 8.

          1
       /     \
     2        3
   /     \    /    \
  4       5  6      7
   \
     8

Example 1:

Input:
   1
 /  \
3    2
Output: 1 3

Example 2:

Input:

Output: 10 20 40
Your Task:
You just have to complete the function leftView() that prints the left view. The
newline is automatically appended by the driver code.
Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Height of the Tree).
```

```
Constraints:
0 <= Number of nodes <= 100
1 <= Data of a node <= 1000
```

```
// { Driver Code Starts
//Initial Template for C#

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DriverCode
{
    class Node
    {
        public int data;
        public Node left;
        public Node right;

        public Node(int key)
        {
            this.data = key;
            this.left = null;
            this.right = null;
        }
    }

    class GFG
    {
        // Function to Build Tree
        public Node buildTree(string str)
        {
            // Corner Case
            if (str.Length == 0 || str[0] == 'N')
                return null;

            // Creating vector of strings from input
            // string after spliting by space
            var ip = str.Split(' ');



            Node root = new Node(int.Parse(ip[0]));
```

```
            // Push the root to the queue
            Queue<Node> queue = new Queue<Node>();
            queue.Enqueue(root);

            // Starting from the second element
            int i = 1;
            while (queue.Count != 0 && i < ip.Length)
            {

                // Get and remove the front of the queue
                Node currNode = queue.Peek();
                queue.Dequeue();

                // Get the current node's value from the string
                string currVal = ip[i];

                // If the left child is not null
                if (currVal != "N")
                {

                    // Create the left child for the current node
                    currNode.left = new Node(int.Parse(currVal));

                    // Push it to the queue
                    queue.Enqueue(currNode.left);
                }

                // For the right child
                i++;
                if (i >= ip.Length)
                    break;
                currVal = ip[i];

                // If the right child is not null
                if (currVal != "N")
                {

                    // Create the right child for the current node
                    currNode.right = new Node(int.Parse(currVal));

                    // Push it to the queue
                    queue.Enqueue(currNode.right);
                }
                i++;
            }

            return root;
        }
        static void Main(string[] args)
        {
            int testcases;// Taking testcase as input
```

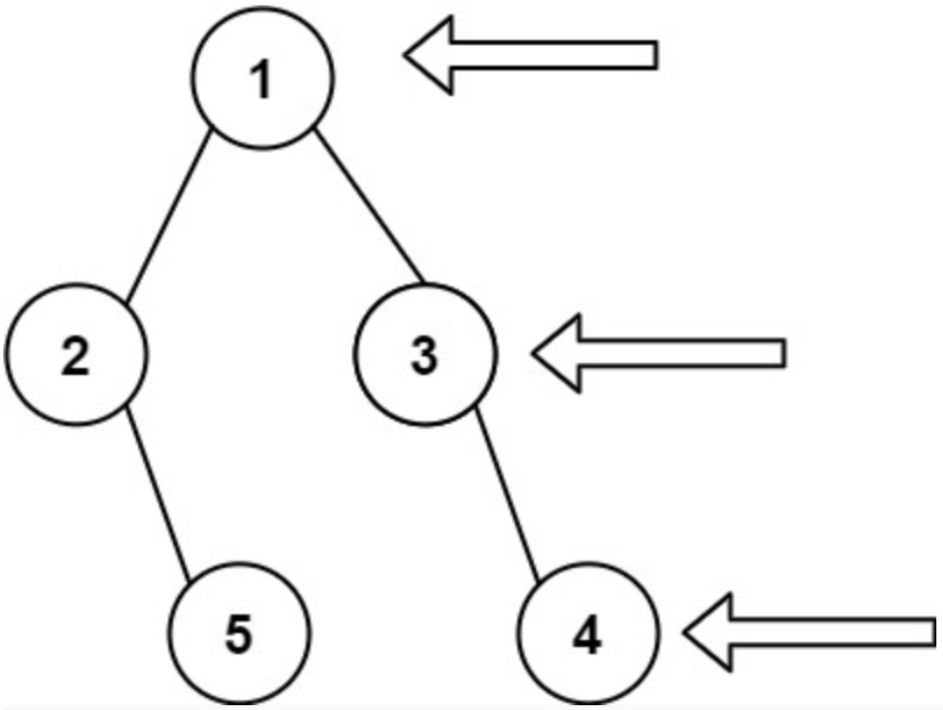
```
            testcases = Convert.ToInt32(Console.ReadLine());
            while (testcases-- > 0)// Looping through all testcases
            {
                var gfg = new GFG();
                var str = Console.ReadLine().Trim();
                var root = gfg.buildTree(str);
                Solution obj = new Solution();
                var res = obj.leftView(root);
                foreach(int i in res){
                    Console.Write(i + " ");
                }
                Console.WriteLine();
            }

        }
    }




 // } Driver Code Ends
//User function Template for C#

    /*  A binary tree Node
        class Node
        {
            public int data;
            public Node left;
            public Node right;

            public Node(int key)
            {
                this.data = key;
                this.left = null;
                this.right = null;
            }
        }
    */
    class Solution
    {

        private static int MaxLevel = 0;
        //Function to return a list containing elements of left view of the
binary tree
        public List<int> leftView(Node root)
        {
            //code here

            var output = new List<int>();

            FindLeftView(root, ref output, 1);
```

```
            return output;
        }

        private void FindLeftView(Node root, ref List<int> output, int
currentlevel){

            if(root == null)
                return;

            //Console.WriteLine($"currentlevel: {currentlevel}, MaxLevel:
{MaxLevel}");

            if(MaxLevel < currentlevel){
                output.Add(root.data);
                MaxLevel = currentlevel;
            }

            FindLeftView(root.left, ref output, currentlevel + 1);
            FindLeftView(root.right, ref output, currentlevel + 1);

            return;
        }
    }

}

// { Driver Code Starts.  // } Driver Code Ends
```

4. https://leetcode.com/problems/binary-tree-right-side-view/

```
Given the root of a binary tree, imagine yourself standing on the right side of
it, return the values of the nodes you can see ordered from top to bottom.
```

Example 1:




```
Input: root = [1,2,3,null,5,null,4]
Output: [1,3,4]
```

Example 2:

```
Input: root = [1,null,3]
Output: [1,3]
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
```

```
    private int MaxLevel = 0;

    public IList<int> RightSideView(TreeNode root) {

        var output = new List<int>();
        FindRightView(root, output, 1);
        return output;
    }

    private void FindRightView(TreeNode root, List<int> output, int currentlevel)
{

        if(root == null)
            return;

        if(MaxLevel < currentlevel){
            output.Add(root.val);
            MaxLevel = currentlevel;
        }

        FindRightView(root.right, output, currentlevel + 1);
        FindRightView(root.left, output, currentlevel + 1);
    }
}
```

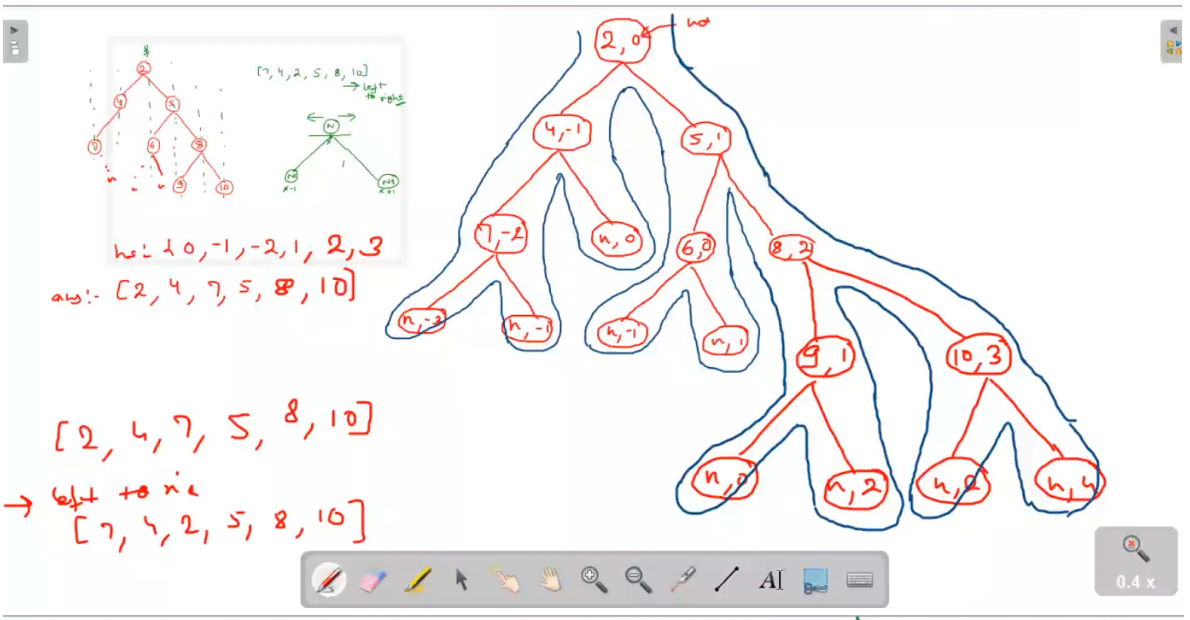
Trees 3

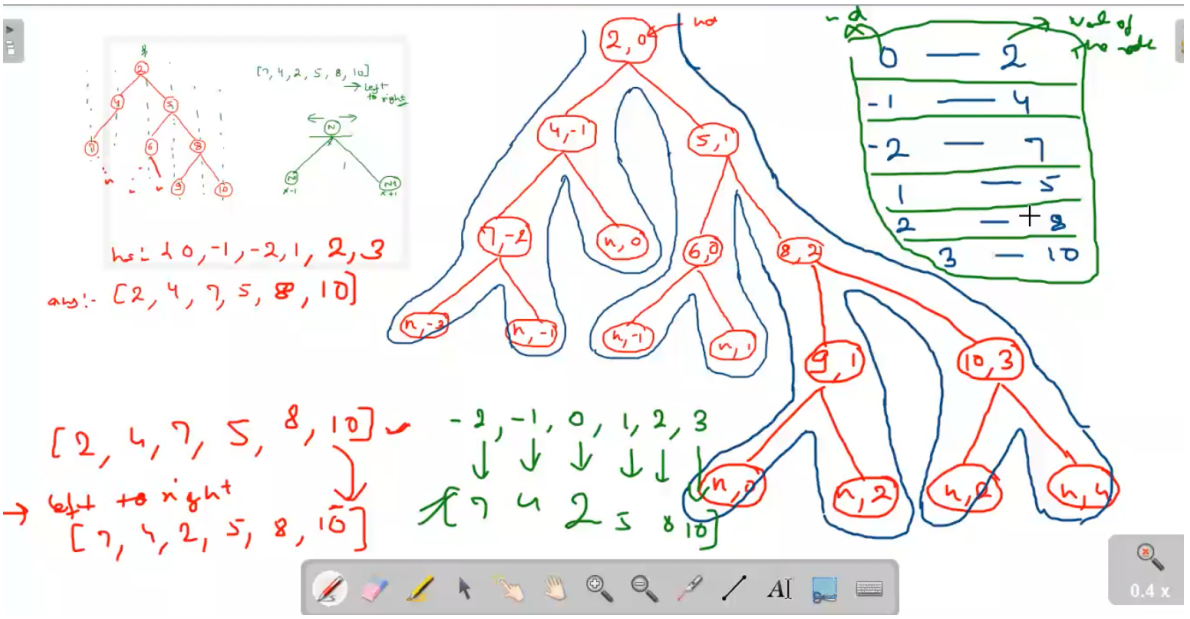
1. https://practice.geeksforgeeks.org/problems/top-view-of-binary-tree/1

Top view of Binary Tree.

```
Given below is a binary tree. The task is to print the top view of binary tree.
Top view of a binary tree is the set of nodes visible when the tree is viewed
from the top. For the given below tree

       1
    /     \
   2       3
  /  \    /   \
4    5  6   7

Top view will be: 4 2 1 3 7
```

```
Note: Return nodes from leftmost node to rightmost node.

Example 1:

Input:
      1
   /    \
  2      3
Output: 2 1 3
```





Assumption: Left subtree will never cross root node on the right side.

Approach 1: Use Hashmap and then store values based on the horizontal distance (HD). Key - HD, value: node value.

Approach 2: Use in order traversal and then keep on adding the values to the output array.

NOTE: THIS QUESTION SHOULD BE SOLVED USING BFS ideally. If we need to solve using DFS, we need to tweak the implementation a bit (e.g. passing a maximum HD till now).

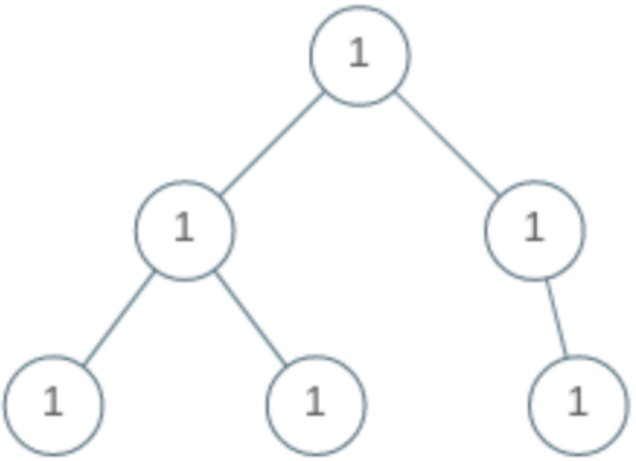
https://www.geeksforgeeks.org/print-nodes-top-view-binary-tree/

```

```

2. https://leetcode.com/problems/univalued-binary-tree/

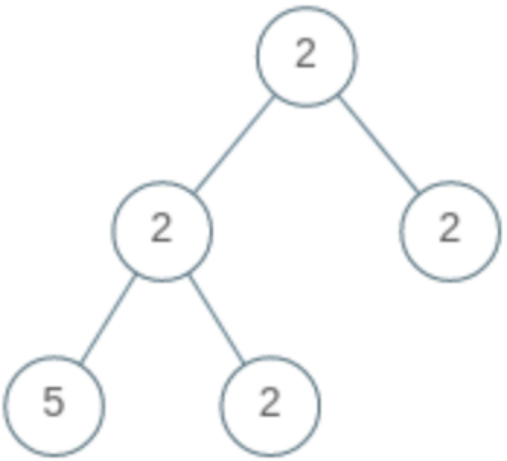
```
A binary tree is uni-valued if every node in the tree has the same value.

Given the root of a binary tree, return true if the given tree is uni-valued, or
false otherwise.
```

Example 1:

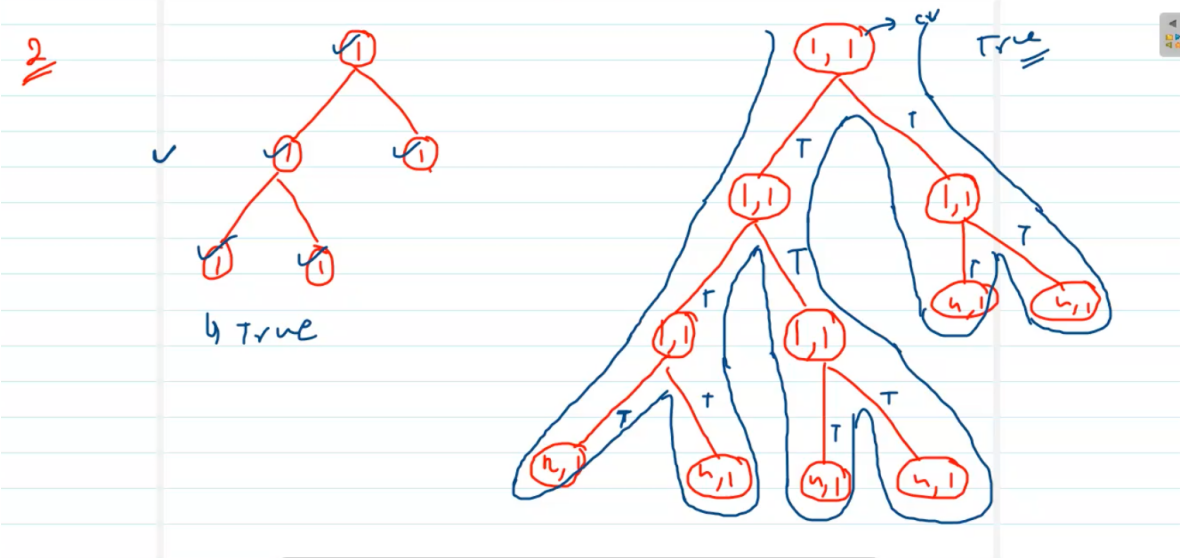


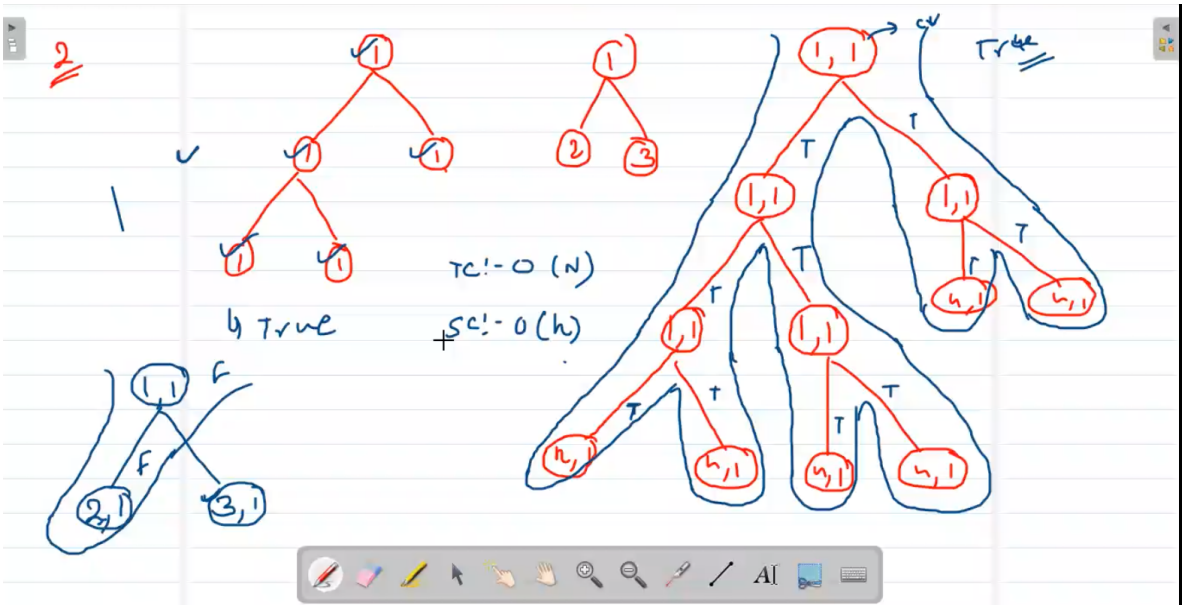

```
Input: root = [1,1,1,1,1,null,1]
Output: true
```

Example 2:




```
Input: root = [2,2,2,5,2]
Output: false
```





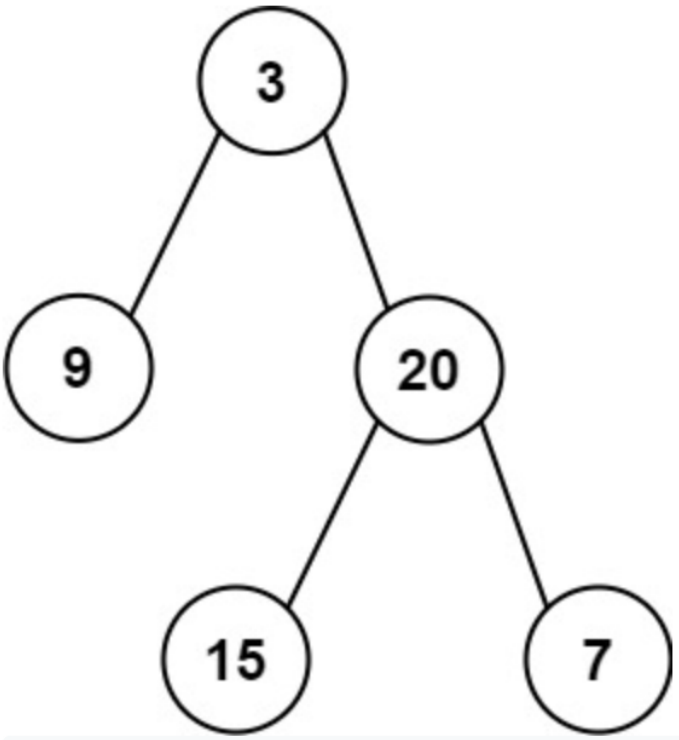

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool IsUnivalTree(TreeNode root) {
```

```
        return CheckIfTreeIsUnival(root, root.val);
    }

    private bool CheckIfTreeIsUnival(TreeNode node, int currentValue){

        if(node == null)
            return true;

        if(node.val != currentValue)
            return false;

        return CheckIfTreeIsUnival(node.left, currentValue) &&
CheckIfTreeIsUnival(node.right, currentValue);

    }
}
```

3. https://leetcode.com/problems/average-of-levels-in-binary-tree/

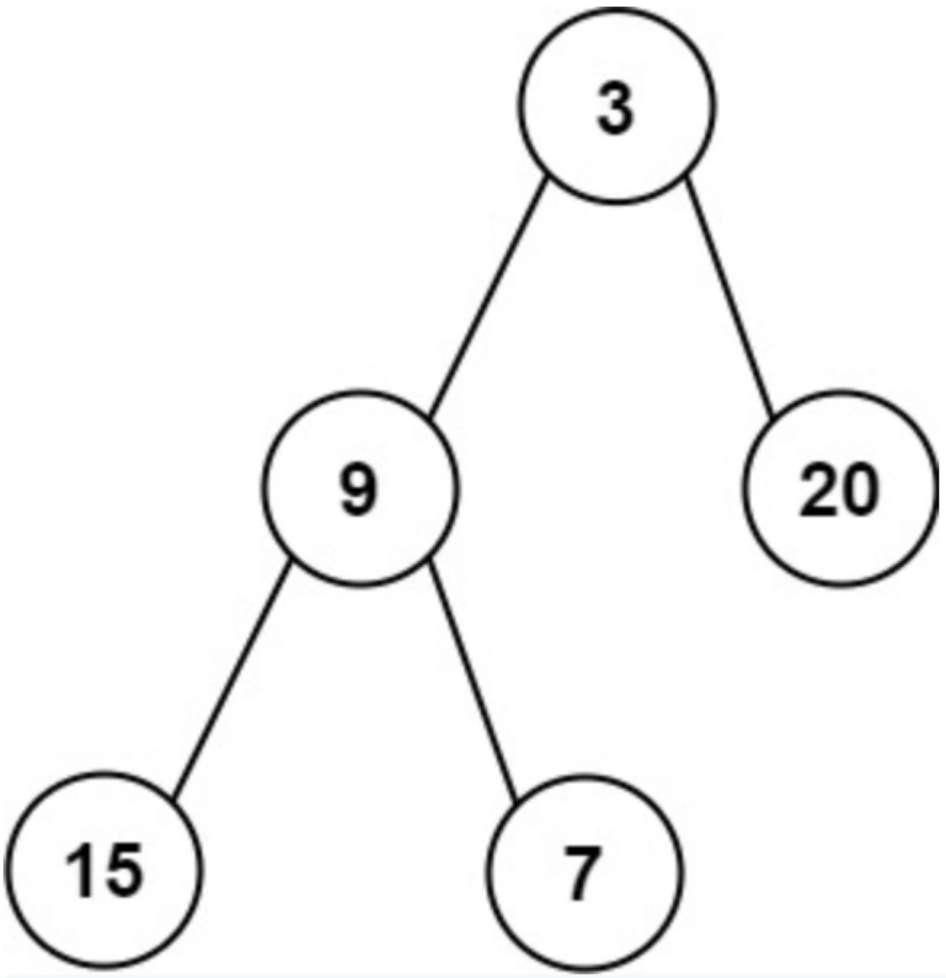
```
Given the root of a binary tree, return the average value of the nodes on each
level in the form of an array. Answers within 10-5 of the actual answer will be
accepted.
```

Example 1:

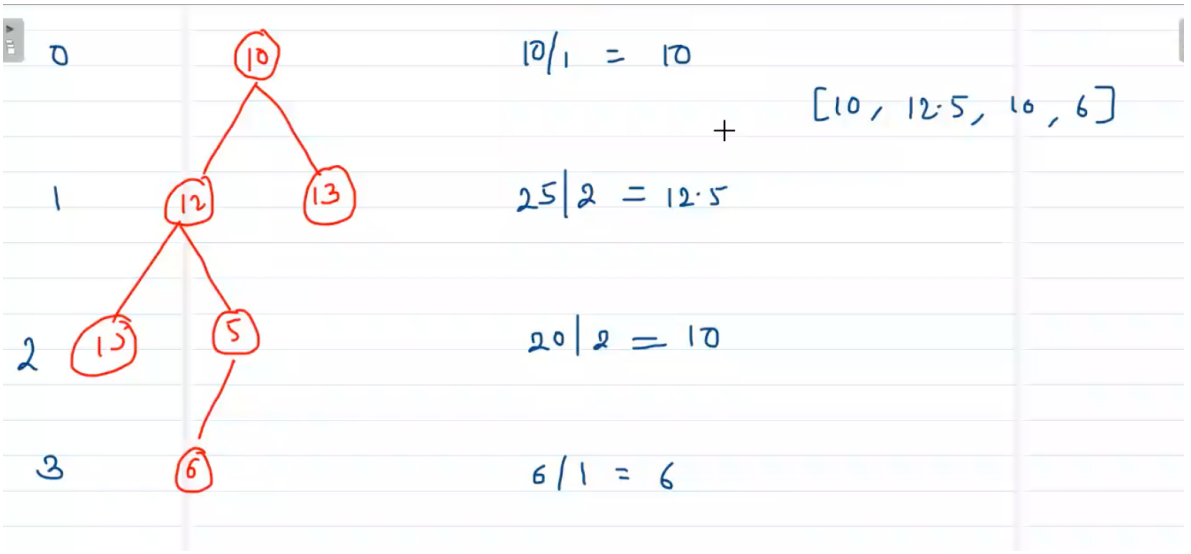


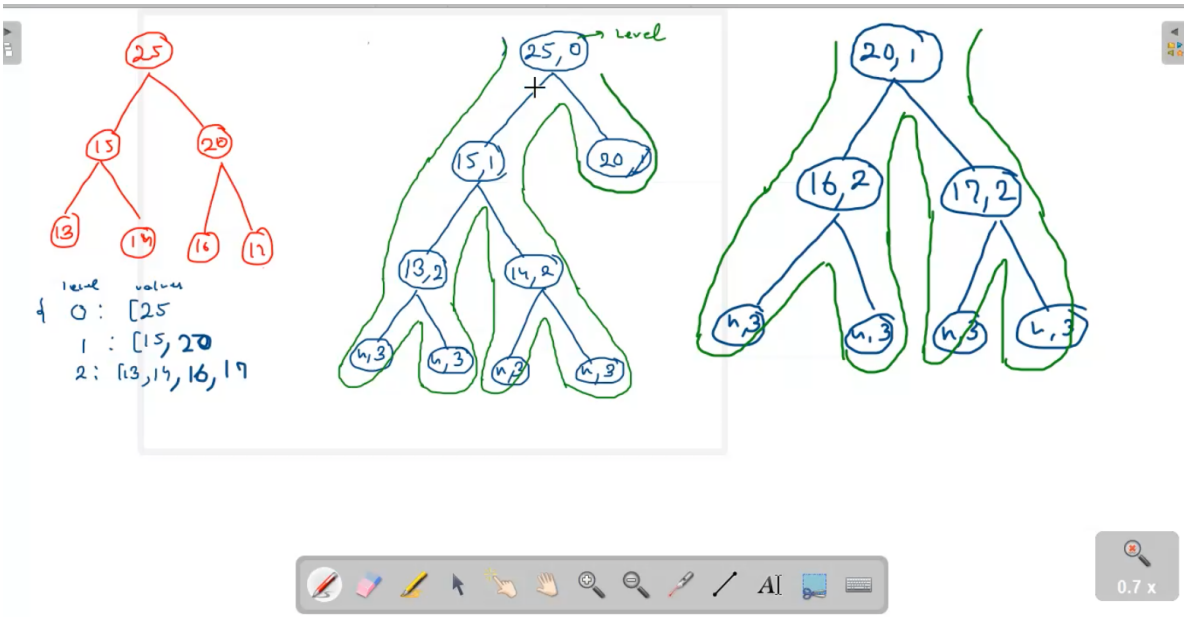

```
Input: root = [3,9,20,null,null,15,7]
Output: [3.00000,14.50000,11.00000]
Explanation: The average value of nodes on level 0 is 3,
on level 1 is 14.5, and on level 2 is 11.
Hence return [3, 14.5, 11].
```

Example 2:




```
Input: root = [3,9,20,15,7]
Output: [3.00000,14.50000,11.00000]
```





Iterate hashmap and calculate average.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
```

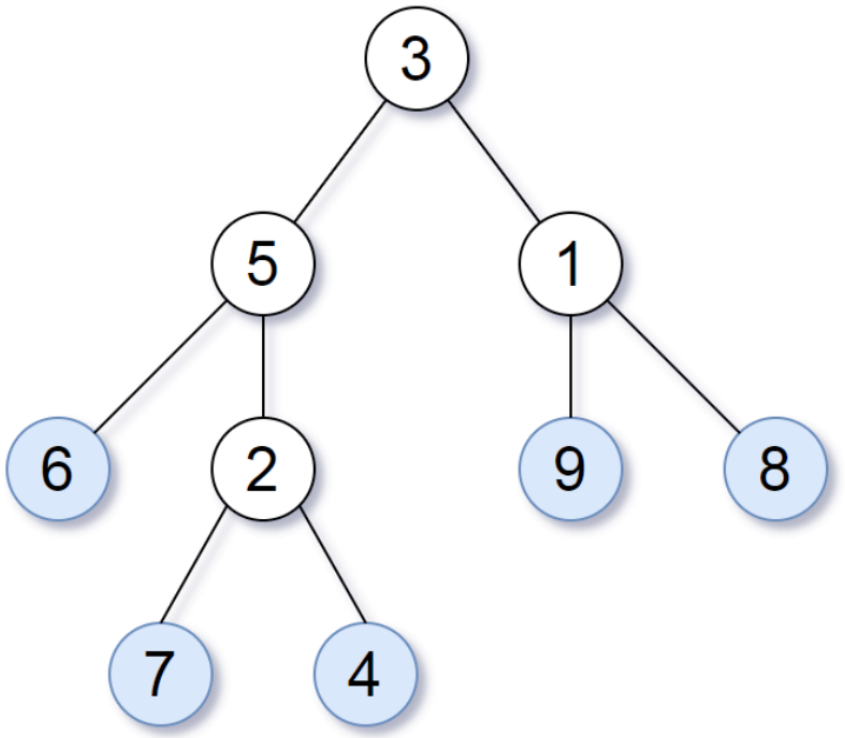
```
public class Solution {

    public IList<double> AverageOfLevels(TreeNode root) {
        var output = new List<double>();

        var levelValues = new Dictionary<int, List<long>>();
        FindLevelValues(root, 0, levelValues);

        foreach(var item in levelValues){
            output.Add(item.Value.Sum() / (double) item.Value.Count());
        }

        return output;
    }

    private void FindLevelValues(TreeNode node, int level, Dictionary<int,
List<long>> levelValues){

        if(node == null)
            return;

        if(levelValues.ContainsKey(level)){
            levelValues[level].Add(node.val);
        }
        else{
            levelValues[level] = new List<long> {node.val};
        }


        FindLevelValues(node.left, level + 1, levelValues);
        FindLevelValues(node.right, level + 1, levelValues);

    }
}
```

NOTE 1: Using long in the hashmap, just to avoid overflow while doing sum.

NOTE 2: This question can be better solved using BFS.

4. https://leetcode.com/problems/leaf-similar-trees/

```
Consider all the leaves of a binary tree, from left to right order, the values
of those leaves form a leaf value sequence.
```



For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

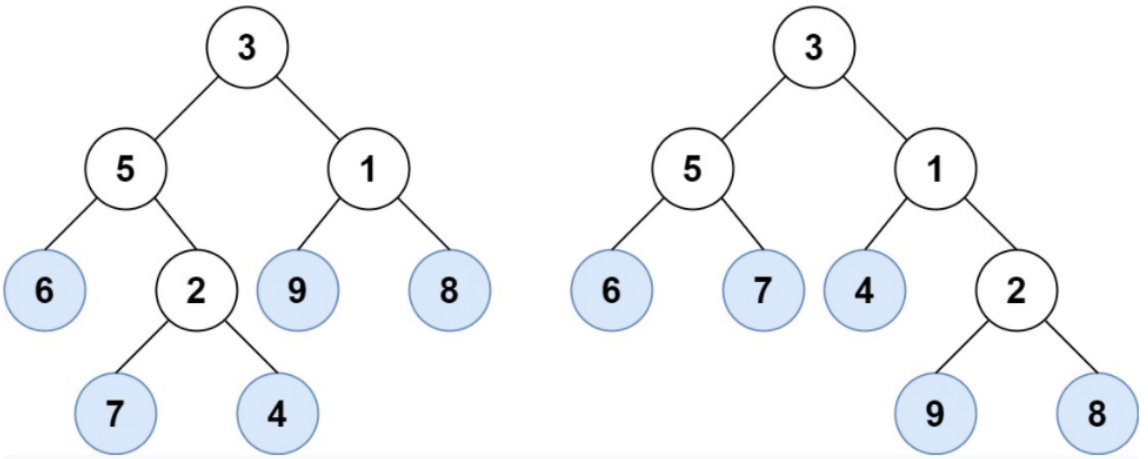
Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

For example, in the given tree above, the leaf value sequence is (6, 7, 4, 9, 8).

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

Return true if and only if the two given trees with head nodes root1 and root2 are leaf-similar.

Example 1:




```
Input: root1 = [3,5,1,6,2,9,8,null,null,7,4], root2 =
[3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
Output: true
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool LeafSimilar(TreeNode root1, TreeNode root2) {

        var leafNodesIn1 = new List<int>();
        var leafNodesIn2 = new List<int>();

        FindLeafNodes(root1, leafNodesIn1);
        FindLeafNodes(root2, leafNodesIn2);

        if(leafNodesIn1.Count() != leafNodesIn2.Count())
            return false;

        int idx = 0;
        for(; idx < leafNodesIn1.Count(); idx++){
            if(leafNodesIn1[idx] != leafNodesIn2[idx])
                return false;
```

```
        }

        return true;
    }

    private void FindLeafNodes(TreeNode node, List<int> leafNodes){

        if(node == null)
            return;

        if(node.left == null && node.right == null)
            leafNodes.Add(node.val);

        FindLeafNodes(node.left, leafNodes);
        FindLeafNodes(node.right, leafNodes);
    }
}
```

Dp Revision

1. https://leetcode.com/problems/unique-paths/

```
A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?
```

Example 1:


```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation:
From the top-left corner, there are a total of 3 ways to
reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Example 3:

```
Input: m = 7, n = 3
Output: 28
```

Example 4:

```
Input: m = 3, n = 3
Output: 6
```

```
public class Solution {
    public int UniquePaths(int m, int n) {

        return FindPaths(m, n, 0, 0, new Dictionary<string, int>());

    }

    private int FindPaths(int m, int n, int currentRow, int currentColumn,
Dictionary<string,int> memo){

        if(currentRow == m - 1 && currentColumn == n - 1)
            return 1;

        if(currentRow >= m || currentColumn >= n)
            return 0;

        var currentKey = $"{currentRow}:{currentColumn}";

        if(memo.ContainsKey(currentKey))
            return memo[currentKey];

        var right = FindPaths(m, n, currentRow, currentColumn + 1, memo);
        var down = FindPaths(m, n, currentRow + 1, currentColumn, memo);

        memo[currentKey] = right + down;

        return memo[currentKey];
    }
}
```

2. https://leetcode.com/problems/unique-paths-ii/

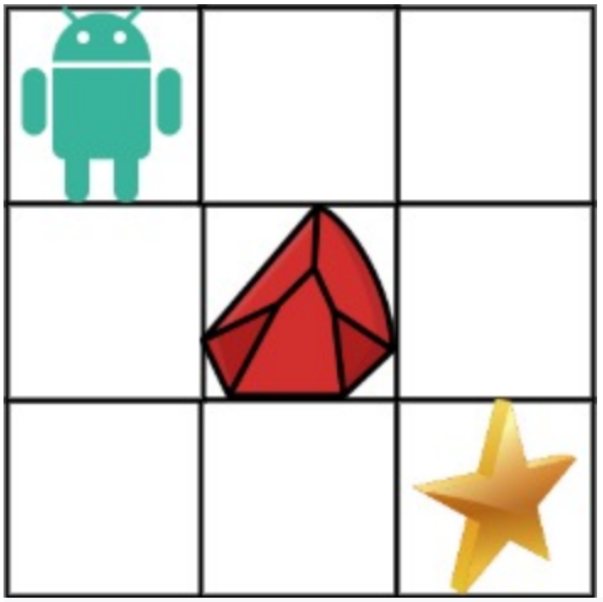
A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

Now consider if some obstacles are added to the grids. How many unique paths would there be?

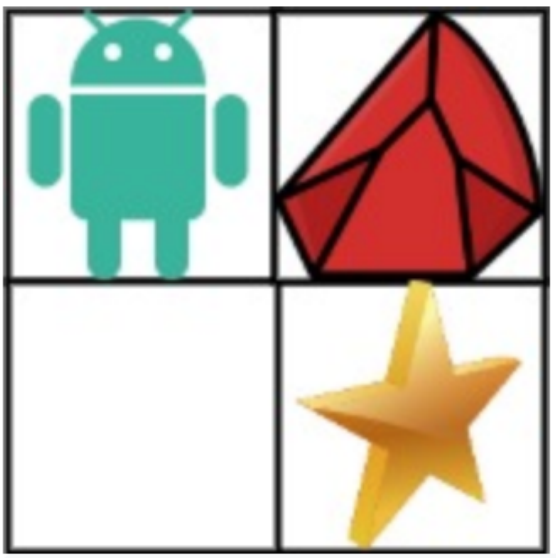
An obstacle and space is marked as 1 and 0 respectively in the grid.

Example 1:



Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
Output: 2
Explanation: There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right

Example 2:



Input: obstacleGrid = [[0,1],[0,0]]
Output: 1

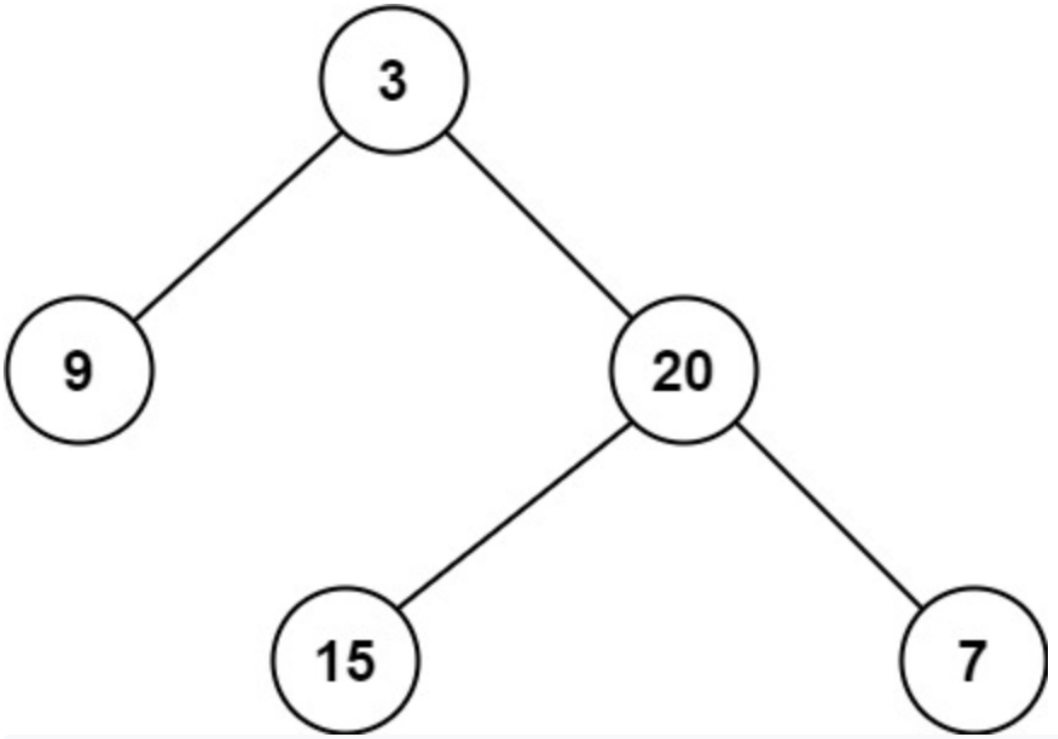
```
public class Solution {
    public int UniquePathsWithObstacles(int[][] obstacleGrid) {
        return FindPaths(obstacleGrid, 0, 0, new Dictionary<string, int>());
    }

    private int FindPaths(int[][] obstacleGrid, int currentRow, int currentCol,
Dictionary<string, int> memo){

        if(currentRow >=  obstacleGrid.Length || currentCol >=
obstacleGrid[0].Length)
            return 0;

        if(obstacleGrid[currentRow][currentCol] == 1)
            return 0;

        if(currentRow ==  obstacleGrid.Length - 1 && currentCol ==
obstacleGrid[0].Length - 1){
            return 1;
        }

        var currentKey = $"{currentRow}:{currentCol}";
        if(memo.ContainsKey(currentKey))
            return memo[currentKey];
```

```
        var right = FindPaths(obstacleGrid, currentRow, currentCol + 1, memo);
        var down = FindPaths(obstacleGrid, currentRow + 1, currentCol, memo);

        memo[currentKey] = right + down;

        return memo[currentKey];
    }

}
```

3. https://leetcode.com/problems/minimum-depth-of-binary-tree/

```
Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root
node down to the nearest leaf node.

Note: A leaf is a node with no children.
```

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 2
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {

    public int MinDepth(TreeNode root) {
        return FindMinDepth(root);
    }

    private int FindMinDepth(TreeNode node){

        if(node == null)
            return 0;

        var leftDepth = FindMinDepth(node.left);
        var rightDepth = FindMinDepth(node.right);

        return leftDepth == 0 || rightDepth == 0 ?
            leftDepth + rightDepth + 1
            : Math.Min(leftDepth, rightDepth) + 1;
    }
}
```