**Algorithmic Thinking in Problem Solving**

**Recursion**

1. **Why is recursion important in CS?**

* This is a problem-solving strategy. It is important when trying to solve problems that look repetitive. This is an option to iterate over different things without using for loops.

1. **What are the elements of recursion? What properties should an ideal recursive method have?**
   1. General case: here the problem is made smaller and smaller until reach the base case.
   2. Base case: this is the case for which the solution finishes and simplest case.

* These are the properties a recursive method should have to solve a problem, then solve a subproblem smaller inside of the same problem, and finally use that solution to solve the whole problem.

1. **Why do we use recursion instead of loops for certain problems?**

* Sometimes problems can be broken down into small problems, and those problems can be similar. Therefore, using a similar solution for all these cases should be better instead of implementing for loops.

1. **How confident do you feel solving problems using recursion (1-10)? 1 – not confident at all, 10 – extremely confident. Share your concerns/comments.**

* 7 because I understand recursion and am able to break down problems into small subproblems, but I need more practice to get better solving problems with recursion.

**Problem 1:** Range Sum of BST

Given the root node of a binary search tree, return the sum of values of all nodes with value between L and R (inclusive).

The binary search tree is guaranteed to have unique values.

|  |  |
| --- | --- |
| **Java** | **Python** |
| /\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/  class Solution {  public int rangeSumBST(TreeNode root, int L, int R) {    }  } | # Definition for a binary tree node.  # class TreeNode:  # def \_\_init\_\_(self, x):  # self.val = x  # self.left = None  # self.right = None  class Solution:  def rangeSumBST(self, root: TreeNode, L: int, R: int) -> int: |

**Problem 2:** Invert Binary Tree

**Example:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  / \  2 7  / \ / \  1 3 6 9 | 4  / \  7 2  / \ / \  9 6 3 1 |

Trivia:  
This problem was inspired by [this original tweet](https://twitter.com/mxcl/status/608682016205344768) by [Max Howell](https://twitter.com/mxcl): Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so f\*\*\* off.

|  |  |
| --- | --- |
| **Java** | **Python** |
| /\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/  class Solution {  public TreeNode invertTree(TreeNode root) {    }  } | # Definition for a binary tree node.  # class TreeNode:  # def \_\_init\_\_(self, x):  # self.val = x  # self.left = None  # self.right = None  class Solution:  def invertTree(self, root: TreeNode) -> TreeNode: |

**Problem 3:** A binary tree is *univalued* if every node in the tree has the same value.

Return true if and only if the given tree is univalued.

|  |  |
| --- | --- |
| **Example 1:** | **Example 2:** |
| A picture containing object, clock  Description automatically generated  **Input:** [1,1,1,1,1,null,1]  **Output:** true | **Input:** [2,2,2,5,2]  **Output:** false |

**Problem 4:** Same Tree: Given the roots of two binary trees, 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.

**Problem 5:** Average of Levels in Binary Tree

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array.

**Example 1:**

**Input:**

3

/ \

9 20

/ \

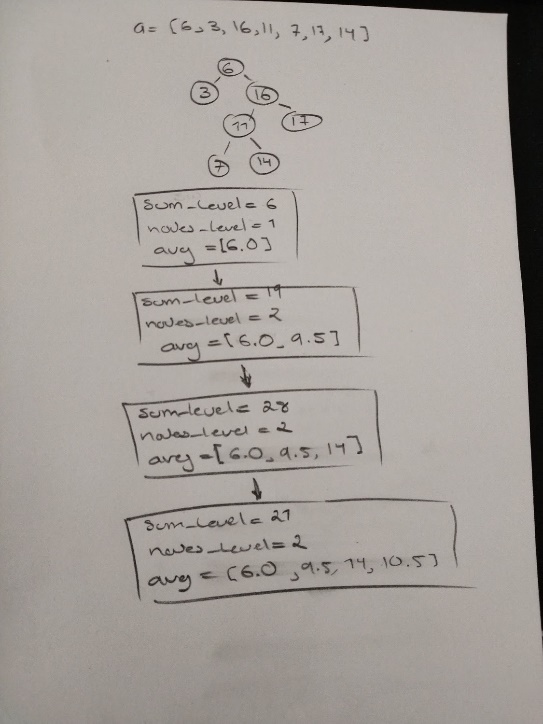
15 7

**Output:** [3, 14.5, 11]

**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].

1. Trace the solution using a concrete instance of the problem



1. What prevented you from solving the problem?

- I did not know how BFS works and had to implement in this problem.

1. What did you learn? Did you have to Google a little more to understand the solution?

- I learned how BFS works in BST.

1. What would you do differently in the future if you were presented with a similar problem?

- I would see if BFS fits for the similar problem. If so, I would implement it.

**Problem 6:** A *full binary tree* is a binary tree where each node has exactly 0 or 2 children. Write a function/method that, given the root of a binary tree, determines if it is full or not.

**Problem 7:** Write a method called splitArray that receives an array of ints as input and determines whether it is possible to divide the ints into two groups, so that the sums of the two groups are the same. Every int must be in one group or the other. Write a recursive helper method that takes whatever arguments you like, and make the initial call to your recursive helper from splitArray. (No loops needed.)

Spend at most 1 hour trying to solve each of the problems. If you are unable so solve the problem after 1 hour, Google the problem and find a solution, then do the following:

1. Trace the solution using a concrete instance of the problem
2. What prevented you from solving the problem?
3. What did you learn? Did you have to Google a little more to understand the solution?
4. What would you do differently in the future if you were presented with a similar problem?

Sometimes… recursion comes in a different *flavor.* You can still use recursion without having to write recursive methods. Sometimes solutions are defined *recursively*, but no recursive calls are needed. Give the following problem a try ( you don’t have to have the right answer to get full credit, nor Google the solution, as long as you show that you tried to solve the problem, you’ll get all the points):

**Bonus Problem:** 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 system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Example 1:

Input: [1,2,3,1]

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: [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.