

Algorithmic Thinking in Problem Solving

Recursion

1. Why is recursion important in CS?

Recursion is essential when using trees, graphs, etc. It will expand as it goes deeper, not like arrays or LinkedList. It gives the flexibility to compute with every node to be the primary data in a method. Once it returns, everything will be returned, and with that value can be used as well.

2. What are the elements of recursion? What properties should an ideal recursive method have?

A recursion always has a base case and is based on those cases, and it depends if it will make a recursive call or return the value.

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

Such as trees, as it goes deeper, it contains more nodes. If everything is computed with loops, it will require more ifs and loops since the part where it needs to be computed is more, which leads to being a more complicated and not efficient program time and space-wise.

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

I would say (7) because I have understood recursion and programmed using recursion. The main reason why I would not give myself a higher point is that I need to practice now since I have not been using recursion, which I softly forgot some of the recursion techniques.

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
<pre>/** * 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) { } }</pre>	<pre># 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:</pre>

Problem 2: Invert Binary Tree

Example:

Input	Output
<pre> 4 / \ 2 7 /\ /\ 1 3 6 9</pre>	<pre> 4 / \ 7 2 /\ /\ 9 6 3 1</pre>

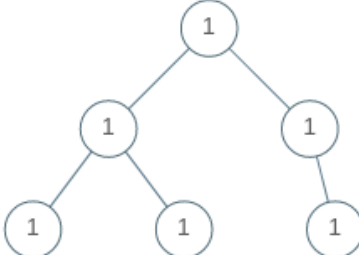
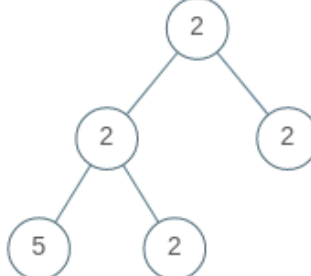
Trivia:

This problem was inspired by [this original tweet](#) by [Max Howell](#): 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
<pre> /** * 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) { } } </pre>	<pre> # 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: </pre>

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:
 <p>Input: [1,1,1,1,1,null,1] Output: true</p>	 <p>Input: [2,2,2,5,2] Output: false</p>

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].

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.)

Problem5:

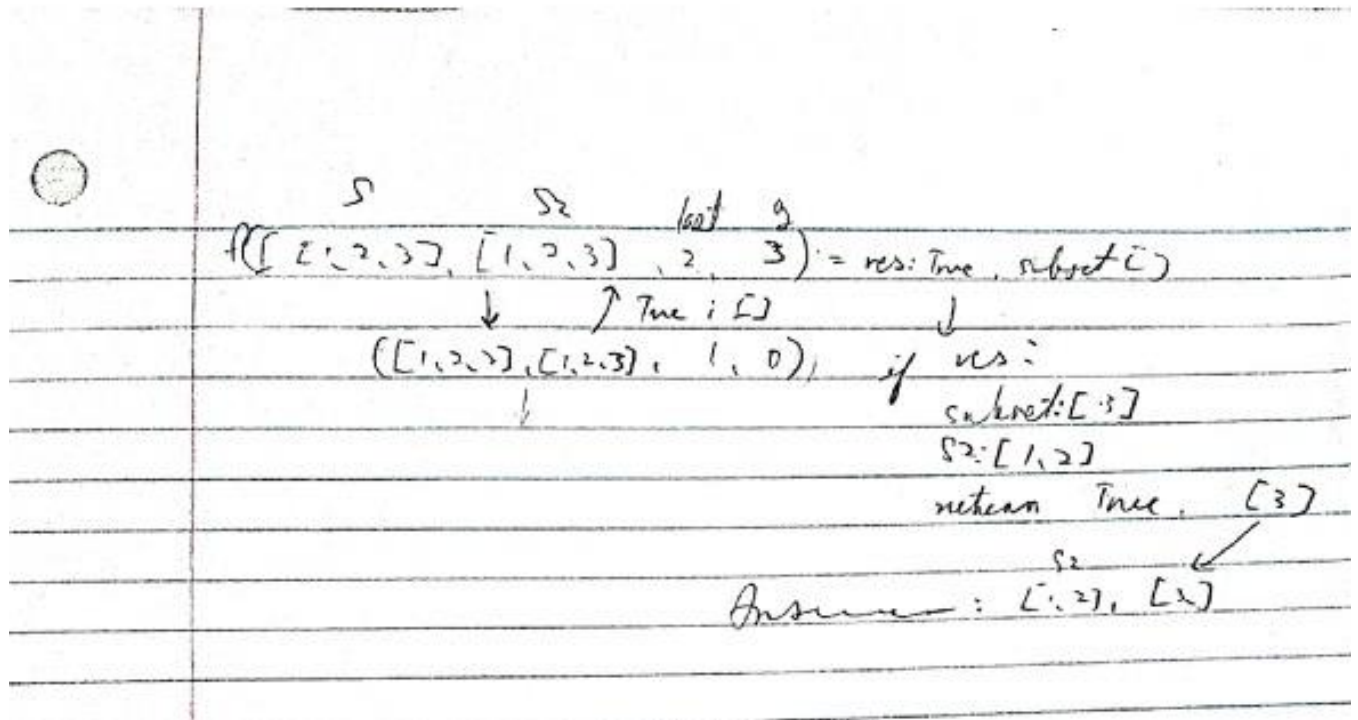
Spend at most 1 hour trying to solve each of the problems. If you are unable so solve the problem after 1 hour, Google the problem and find a solution, then do the following:

1. Trace the solution using a concrete instance of the problem
2. What prevented you from solving the problem?
It prevented me from using other data structures and a deep understanding of a tree and a different data structure. Need practice on recursion as well as trees, queues.
3. What did you learn? Did you have to Google a little more to understand the solution?
I learned that other data structures are available throughout the recursion problem to solve, and my brain did not go to think in that way.
4. What would you do differently in the future if you were presented with a similar problem?
I would write all the possible data structures and algorithms that I can think of in order to solve the problem unless I forget what I can use and will be stuck—having written down all the data structures and algorithms will help me where to start and what I can use.

Problem7:

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:

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



6. What prevented you from solving the problem?

I knew that backtracking is used; however, the lack of recent coding of dynamic algorithms and recursion is mainly not getting to a result. Need more practice.

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

I didn't google. Instead, I went to my GitHub repo to see dynamic algorithms and recursion problems in order to remember and understand the concept. I watched a video to understand more about backtracking.

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

I would need to understand the algorithms and practice recursion very well to convert the algorithm in graph or scratch paper to a programming language.

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 = $2 + 9 + 1 = 12$.