Assignment 03

Instructions

1. Write your functions according to the signature provided below in a python(.py) file and submit them to the autograder for evaluation.
2. The autograder has a limited number of attempts. Hence, test it thoroughly before submitting it for evaluation.
3. Save your python file as text(.txt) file and submit it to the canvas.
4. If the assignment has theoretical questions, upload those answers as a pdf file on the canvas.
5. Submission on the canvas is mandatory and the canvas submitted text file should be the same as that of the final submission of the autograder. If not, marks will be deducted.
6. No submission on canvas or no submission on autograder then marks will be deducted.
7. Access the auto grader at https://c200.luddy.indiana.edu.

Question 1 - The Order Mystery

You have stumbled upon a cryptic puzzle: a hidden message encoded within the structure of a binary tree. The only clues available are two lists representing the inorder and postorder traversals of this tree. Your task is to write a function construct_tree(inorder: List[int], postorder: List[int]) which takes the inorder and postorder traversals, constructs the original tree.

Constraints

- 1 <= inorder.length <= 3000
- postorder.length == inorder.length
- 3000 <= inorder[i], postorder[i] <= 3000
- inorder and postorder consist of unique values.
- Each value of postorder also appears in inorder.
- inorder is guaranteed to be the inorder traversal of the tree.
- postorder is guaranteed to be the postorder traversal of the tree.

Example:

```
    3
   / \
  9   20
     /  \
    15   7
```

Input:inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]

Output:[3,9,20,null,null,15,7]

Function

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def construct_tree(self, inorder: List[int], postorder: List[int]):
    """
    :rtype: TreeNode
    """
    # Your code goes here
```

Question 2 - UPS Logistics

UPS operates an extensive network of stores and distribution centers connected by transportation routes. This network forms a hierarchical structure, where each node represents a UPS store or facility, and the edges represent the routes connecting them.

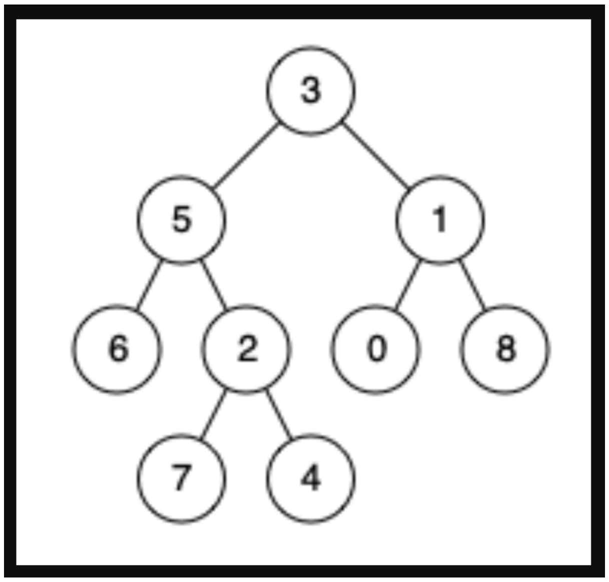
There are situations where two UPS delivery drivers, operating in different regions, need to meet to exchange parcels. This could be due to changes in delivery priorities, vehicle capacity optimization, or last-minute customer requests. To minimize delays and maintain efficiency, they must find the nearest UPS facility where they can meet without significantly deviating from their respective routes.

You have been hired as a software engineer in the UPS logistics department to solve this issue. Write a program that can find the meeting place where they can meet, given two locations of the stores in the network. ### Constraints

- The number of stores in the network is in the range [2, 10^5]
- -109 <= store.id <= 109
- All store ids are unique
- storeId1 != storeId2
- storeId1 and storeId2 will exist in the network

Test Cases:

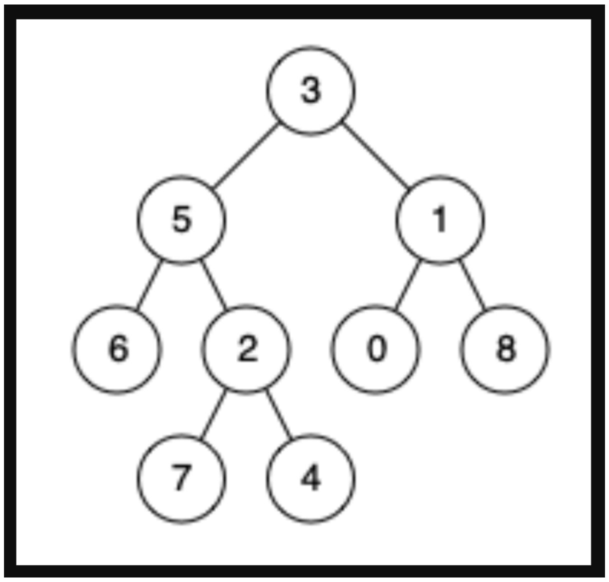
Example 1:



```
Input: storeId1 = 5, storeId2 = 1
Output: 3
```

```
Explanation: The nearest facility they can meet is 3
```

Example 2:



```
Input: storeId1 = 5, storeId2 = 4
Output: 5
Explanation: The nearest facility they can meet is 5
```

Function

```python
class Store:
  def __init__(self, id, left=None, right=None):
    self.id = id
    self.left = left
    self.right = right

def find_nearest_common_facility(root, storeId1, storeId2):
    # Function implementation here
```

Question 3 - Beautiful Ninja Formation

In Konoha village, there's a **beautiful ninja formation** of length `n`. The ninjas must follow a strict formation rule passed down from the Hokage:

1. **Ninja Formation Rule #1** : The team is a permutation of the ninjas numbered from 1 to `n`.
2. **Ninja Formation Rule #2** : For any ninja positions `i < j`, there is no ninja in between them (at position `k` where `i < k < j`) such that the strength of ninja `k` is equal to half the sum of the strengths of the ninjas at positions `i` and `j`. This prevents a stronger ninja from getting sandwiched between weaker ones.
3. **Special Hokage Orders** : The leftmost ninja (first position) must always be an **odd-numbered ninja** (as they represent the unpredictable nature of Naruto) and the rightmost ninja (last position) must always be an **even-numbered ninja** (representing Sasuke's disciplined and balanced nature).

The challenge is to create such a ninja formation of size `n`. The question should be solved using divide and conquer approach.

Constraints

```
* 1 <= n <= 1000
```

Example 1:

Input: n = 4 Output:[1, 3, 2, 4] Explanation: A beautiful array for n = 4 is [1, 3, 2, 4]. This satisfies the beautiful array conditions: 1. [1, 3, 2, 4] is a permutation of [1, 2, 3, 4]. 2. No index k such that 2 * nums[k] = nums[i] + nums[j] for i < k < j. There are other permutation of answer for this n = 4 such as [2,1,4,3], but it will not satisfy the third condition where the left most number should be odd number

Example 2:

Input: n = 5 Output:[1, 5, 3, 2, 4] Explanation: A beautiful array for n = 5 is [1, 5, 3, 2, 4]. This satisfies the beautiful array conditions: 1. [1, 5, 3, 2, 4] is a permutation of [1, 2, 3, 4, 5]. 2. No index k such that 2 * nums[k] = nums[i] + nums[j] for i < k < j. Similarly as previous example, there are other combination such as [4,2,3,5,1], but it will fails as we the third condition.

Function

```python
def beautifulNinjaFormation(n: int) -> list[int]:

    return ans
```

Question 4 - Tree Buddy Finder

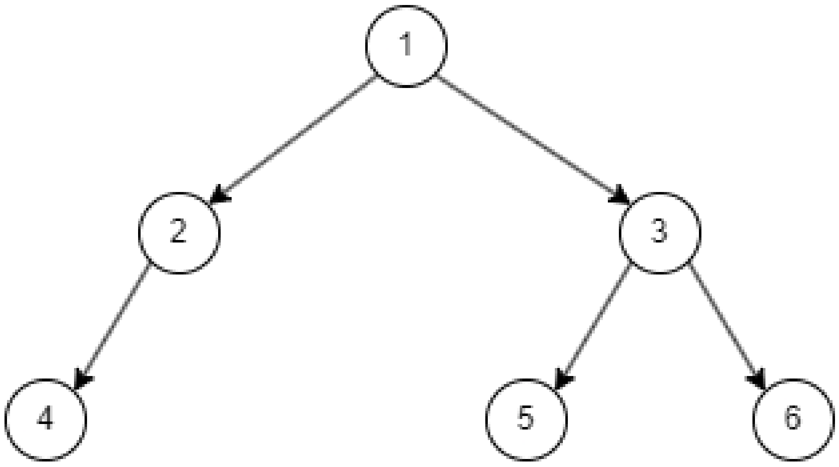
Make good friends in life. Since making good friends takes time, in the meantime, help binary trees find their `next` and `prev` buddies, as shown in the example!

Initially, both `next` and `prev` pointers are set to `None`. You don't need to return anything; modify the binary tree in place.

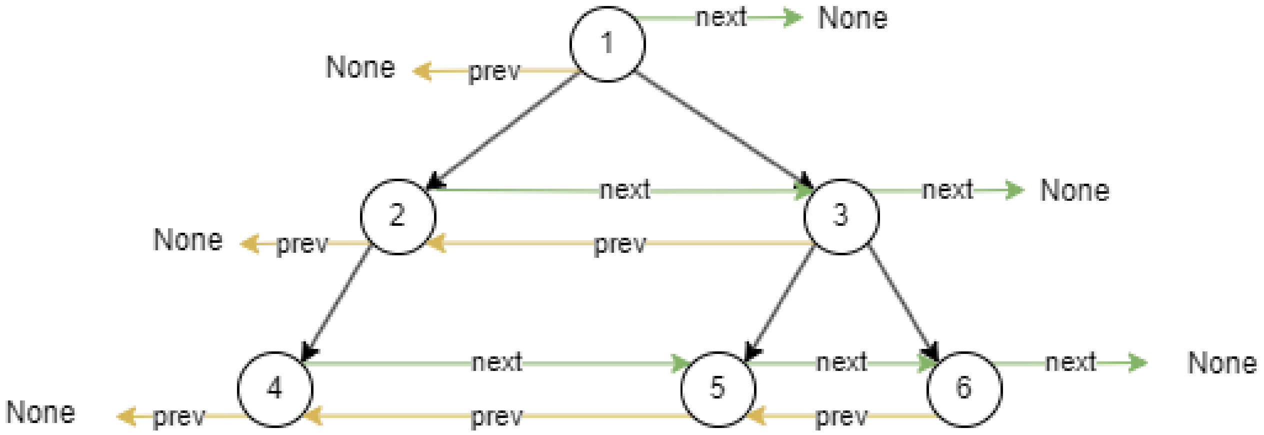
Example:

Example 1:

Input:



Output:



Explanation:

```
{
    "inorder": [4, 2, 1, 5, 3, 6],
    "next": [5, 3, None, 6, None, None],
    "prev": [None, None, None, 4, 2, 5]
}
```

"inorder" represents the inorder traversal of the tree. "next" and "prev" indicate the nodes to which the `next` and `prev` pointers should point for each node in the inorder traversal.

Function

```python
class NodeWithBuddies:
    def __init__(self, val, left=None, right=None, next=None, prev=None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
        self.prev = prev

def findBuddies(root: NodeWithBuddies) -> None:
    pass
```

Question 5 - GOAT Debate

In a legendary matchup, Ronaldo and Messi are facing off in a unique competition called "Endgame Scores." The challenge lies in a series of games where each game can yield a specific number of points. The points for each game are laid out in a line, and each player, starting with Ronaldo, takes turns selecting a game to play from either end of the line-up.

The aim is to determine whether Ronaldo, who always starts first and plays with perfect strategy, can ensure that he ends up with a total score that is at least equal to Messi's score by the end of the competition, given that Messi also plays optimally.

Game Rules:

- Each turn allows a player to choose one game from either end of the remaining line of games.
- Each game can only be played once and is then removed from the line-up.
- The competition continues until all games have been chosen.
- Ronaldo is deemed the winner or at least ties if his total score is not less than Messi's score after all games have been played.

Return True if Ronaldo can win the game; else return False.

Test Cases

```
Input: points = [3, 99, 155, 8]
Output: True
Explanation: To start with, Ronaldo can choose either 3 or 8. If Ronaldo picks 8,
then Messi can pick 155, and he will win the game because it does not matter what
Ronaldo chooses subsequently; he will not gather enough points to match Messi.
Hence, Ronaldo chooses 3 to begin with; then Messi can choose from 99 and 8; no
matter what he picks; after that, Ronaldo will pick 155 and win the game.
```

```
Input: points = [5, 1, 100, 1, 5]
Output: False
Explanation: To begin with, Ronaldo can choose from the two 5's at either end.
Messi will pick the five at the other end, no matter which one Ronaldo chooses.
Now, Ronaldo can choose from the two 1's at either end. No matter which 5 he
chooses, Messi will pick 100 and win the game.
```

Constraints

- 1 <= points.length <= 20
- 0 <= points[i] <= 10^7

Function Signature

```python
def endgameScores(points: list[int]) -> bool:
    pass
```

Question 6 - Taste of India

John, after helping his friends with pizzas at QuarryPie, is craving some Indian food. Since it's 1 PM, right after his shift, he suggests going to Taste of India for lunch with his friends. The restaurant offers an affordable lunch buffet from 12 PM to 3 PM, which is a much better deal than any burger joint around. Including John and Jack, there are a total of X people in the group.

After enjoying a satisfactory meal, they decide to have a refreshing lassi. The restaurant provides X straws, each of a different length, corresponding to the amount of lassi each person can drink. The length of a straw is proportional to the amount of lassi that person drinks. However, in the order of drinking, person i with straw length Si will only be satisfied if all the people who drank before them had straws shorter than Si. In other words, person i is happy only if they drank more lassi than all the people before them.

The owner wants to make exactly k people in the group happy. Can you find the number of possible drinking orders that satisfy this condition? Since the number of possibilities may be large, return the answer modulo 10^9 + 7.

Constraints

1 <= X <= 1000
1 <= k <= X

Example:

Input: X = 4, k = 3
Output: 6
Explanation:
Lets assume the straws of 4 different lengths are 2, 5, 7, 8 are given to the 4 people. We need to find the orders such that 3 people are satisfied. Possible orders are:
[2, 5, 8, 7] - First person is satisfied since there is no one before him. Second person is satisfied, since 1st person only drank a proportion of 2 of lassi. Third person is satisfied since there is no one who drank more than proportion of 8. But 4th person is not satisfied, because third person drank a proportion of 8 of lassi which is more than 7
Similary other valid orders are:
[2, 7, 8, 5], [5, 7, 8, 2], [5, 2, 7, 8], [5, 7, 2, 8], [2, 7, 5, 8]

Function

```
def satisfyingOrders(X: int, k: int) -> int:
    pass
```

Question 7 - Bat-Signal Power Patterns

In a world where Gotham's lights flicker, Batman faces a unique challenge: keeping the Bat-Signal powered. Alfred has set up a special system that only functions if there's always a spark of energy in every adjacent pair of connections. Each connection is represented by a binary string where "x" stands for an active power line and "y" means no power. To keep the Bat-Signal steady, every two consecutive connections must have at least one "x". If the string of connections breaks this rule, Gotham may fall into darkness. Batman needs to generate all possible connection sequences of length `n` that will keep the signal strong, and he asks for your help to ensure all valid sequences are ready for the night.

Now, it's up to you to find all the possible connection patterns that can power up the Bat-Signal and keep Gotham safe. Can you ensure Batman always has a stable signal?

Constraints

1. `1 <= n <= 18`

Example:

Input: `n=3`
Output: `["yxy","yxx","xyx","xxy","xxx"]`
Explanation:
The valid strings of length 3 are: "yxy", "yxx", "xyx", "xxy" and "xxx".

Input: `n = 1`
Output: `["x","y"]`

Function

```python
def batmanSignal(n):
    return ans
```

Question 8 - Sacred Sequence Challenge

In a distant kingdom, there lived a wise king who had a set of magical keys, each engraved with a distinct number from 1 to n. These keys held great power, but their true strength could only be unlocked when arranged in a specific order. The kingdom's ancient scrolls revealed that the keys could be arranged in exactly n! unique ways, forming what the kingdom referred to as "The Sacred Sequences."

The king, fascinated by the infinite possibilities of his magical keys, wanted to know what the k-th sequence of keys would be if they were all arranged in order from the first to the last. The Royal Archivist, tasked with this, was overwhelmed by the sheer number of permutations.

The king posed the following challenge to the clever scholars of the land:

"Given the number of keys, n, and the position of the sequence, k, can you uncover the exact arrangement of the keys in the k-th Sacred Sequence without listing all the possibilities?"

The scholars knew they had to find a way to determine this specific sequence directly, without going through all the arrangements. Can you, like the scholars of old, solve the king's challenge and determine the k-th permutation of the magical keys?x

Constraints

- 1 <= n <= 9
- 1 <= k <= n!

Example 1:

```
Input: n = 3, k = 3

Output: "213"
Explanation:
When n = 3, there are 3! = 6 unique permuations.
["123", "132", "213", "231", "312", "321"]
"213" is the kth arrangement in the ordered sequence.
```

Example 2:

```
Input: n = 4, k = 9
Output: "2314"
Explanation:
"2314" is the kth (9th) arrangement in the ordered list of sequences.
```

Function

```python
def sacred_seqeunce(keys:int, position:int) -> str:
    # Write your code here
    pass
```

Question 9 - Gojo's Cursed Spirit Clean-Up

Gojo Satoru is up against a line of cursed spirits, each represented as a node in a linked list. Every cursed spirit has a power level represented by a number. Gojo has decided to use his Limitless technique to eliminate all cursed spirits with a specific power level.

Given the head of a linked list of cursed spirits and a target power level, help Gojo eliminate all cursed spirits whose power level matches the given value. After all matching cursed spirits are eliminated, return the new head of the linked list.

Constraints

-The number of cursed spirits (nodes) in the list is in the range [0, 10^4]. -Each cursed spirit's power level is between 1 and 50. -Gojo can eliminate cursed spirits with a power level in the range 0 <= val <= 50. -Question has to be solved using Recursion

Example:

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

```
Input: head = [], val = 1
Output: []
```

```
Input: head = [7,7,7,7], val = 7
Output: []
```

Function

```
def removeElements(head: Optional[ListNode], val: int) -> Optional[ListNode]:
    return result

class ListNode:
     def __init__(self, val=0, next=None):
         self.val = val
         self.next = next
```

Question 10- Mr. Bean's Toy Sorting Puzzle

Mr. Bean has made his toy collection even more puzzling! This time, he insists that the toys must follow a specific zig-zag pattern: "< > < >". This means that:

The first toy should be smaller than the second. The second toy should be larger than the third. The third toy should be smaller than the fourth. And so on...

Mr. Bean is confident that there's only one correct way to sort his toys this way for each test case.

Given an integer array toys representing Mr. Bean's toy collection, reorder it so that the toys follow the pattern:

toys[0] < toys[1] > toys[2] < toys[3] > ... There is exactly one valid solution for each test case, so find and return it. Mr. Bean prefers to use the Divide and Conquer approach, so make sure your solution works that way!

Constraints

```
* The input array toys will have a length between 1 and 100,000.
* There will always be exactly one solution.
```

Example 1:

Input: toys = [1, 5, 1, 1, 6, 4] Output: [1, 6, 1, 5, 1, 4] Explanation: AIn the only valid solution, the toys follow the pattern < > < >.

Example 2:

Input: toys = [1, 3, 2, 2, 3, 1] Output: [2, 3, 1, 3, 1, 2] Explanation: In the only valid solution, the toys follow the pattern < > < >.

Function

```
def rearrangeAllToys(toys):
    pass
```

Question - 11 The Student and the Puzzle of the Trees

In the heart of Bloomington, there was a curious student named Ava who loved solving complex problems between her studies at Indiana University. One fall afternoon, while taking a break at Dunn Woods, she found herself surrounded by towering trees. As she gazed at the branches spreading out in different directions, an idea started to form in her mind—a problem-solving challenge that mirrored the way the trees grew.

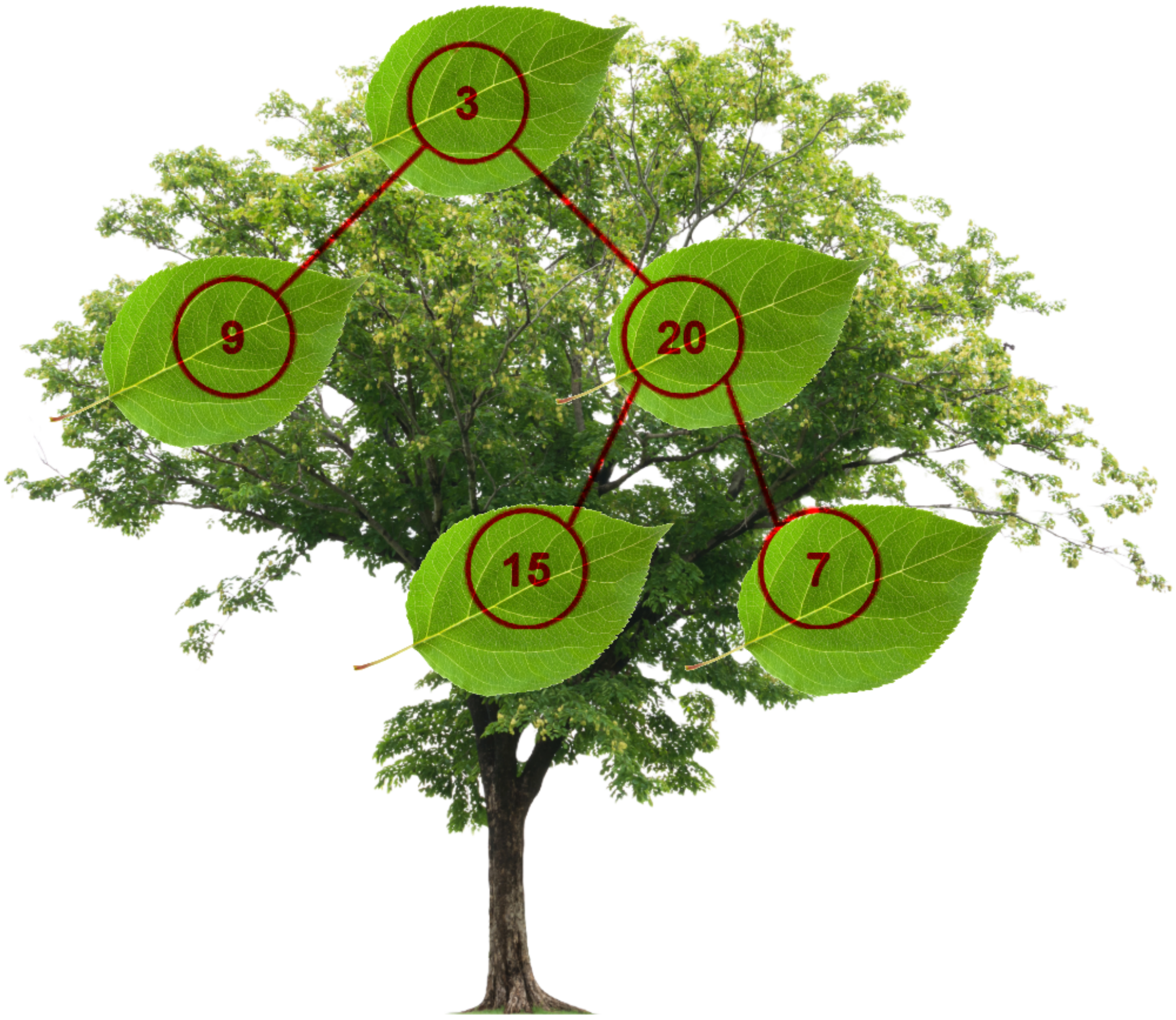
Ava imagined each tree as a complex puzzle where every branch and leaf had a position. The higher branches represented higher levels, while the left and right spreads of the tree resembled an intricate grid. She knew that to solve this puzzle, she'd need to figure out how to "read" the tree from top to bottom, left to right, organizing everything based on where the branches stood in this imaginary grid.

It reminded her of how students sat in a lecture hall at IU: each row of students represented a level, and whether a student sat closer to the left or right was like positioning in the tree. The trick to solving the problem was simple but challenging: start from the leftmost position, gather the students sitting in each column, and move your way to the rightmost. If two students were in the same seat position, you'd sort them by name or ID number.

With this in mind, Ava began sketching out her solution. She had to:

1. **Track the position** of each branch as if it were a student in a lecture hall.
2. **Group branches by their position** and list them from the highest to the lowest in each column.
3. **Sort branches that shared the same level and position** to ensure they were ordered correctly.

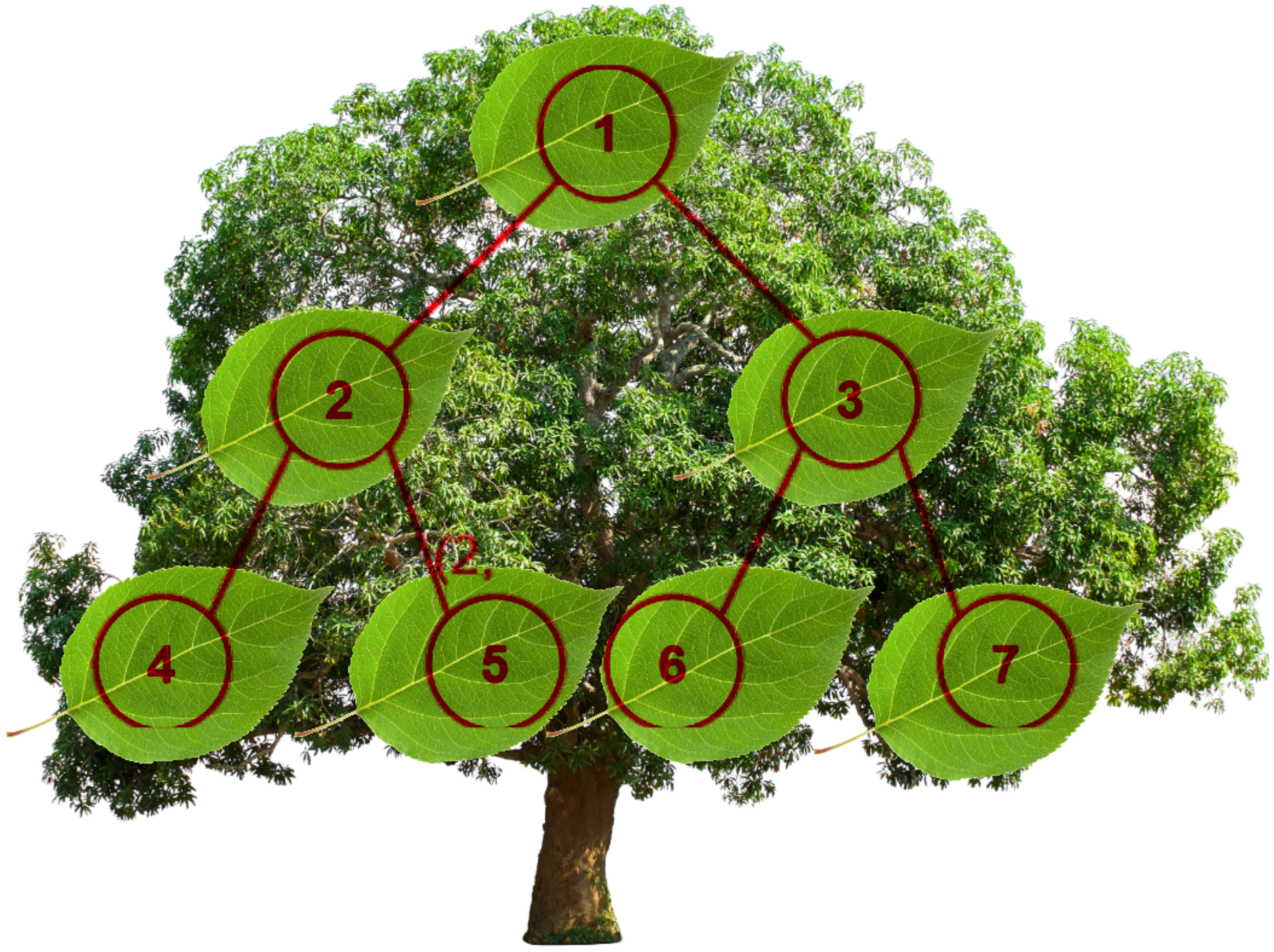
Example 1:



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

Explanation:
- Column -1: Only node 9 is in this column.
- Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.
- Column 1: Only node 20 is in this column.
- Column 2: Only node 7 is in this column.

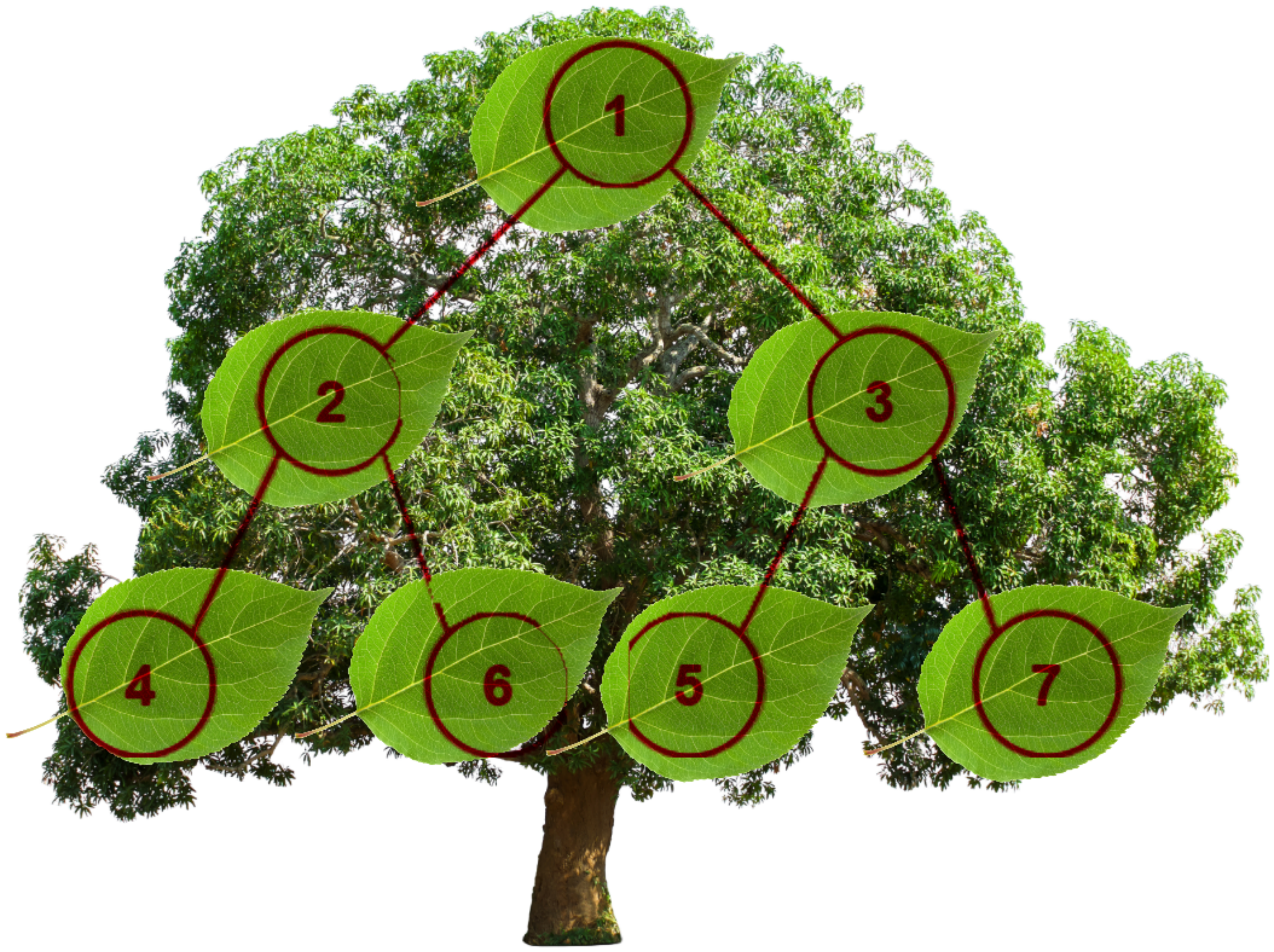
Example 2:



Input: `root = [1,2,3,4,5,6,7]`
Output: `[[4],[2],[1,5,6],[3],[7]]`

Explanation:
- Column -2: Only node 4 is in this column.
- Column -1: Only node 2 is in this column.
- Column 0: Nodes 1, 5, and 6 are in this column.
1 is at the top, so it comes first.
5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.
- Column 1: Only node 3 is in this column.
- Column 2: Only node 7 is in this column.

Example 3:



Input: `root = [1,2,3,4,6,5,7]`
Output: `[[4],[2],[1,5,6],[3],[7]]`

Explanation:
This case is the exact same as Example 2, but with nodes 5 and 6 swapped.
Note that the solution remains the same since 5 and 6 are in the same location and should be ordered by their values.

Function

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def puzzleOfTheTrees(root: TreeNode) -> List[List[int]]:
  # Your Code Here ...
```

Constraints:

- The number of nodes in the tree is in the range `[1, 1000]`.
- `0 <= Node.val <= 1000`