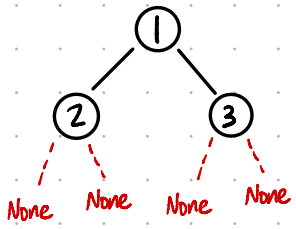
Recursion

Solving a problem where a function calls itself to break the problem into smaller and smaller pieces until it reaches the simplest version of the problem, which is called the base case. Once the base case is solved, the function "works its way back up" to solve the bigger problem.

1. Base Case: simplest version of the problem, where no more recursion is needed
2. Base Case acts like a stop signal for the function. W/out it, the recursion would go on forever, causing the program to crash w/ stack overflow.
3. Recursive Case: part where function calls itself
4. Recursion keeps "breaking down" the problem into smaller chunks until it reaches the base case.

Invert Binary Tree — using pre-order DFS

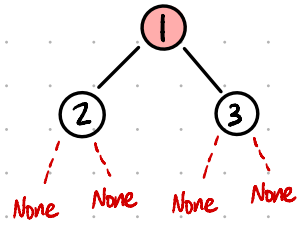



```
def invertTree(root):
    # Base Case
    if not root:
        return
    # Recursive Case
    # Swaps
    root.left, root.right = root.right, root.left
    # Recursive Calls
    invertTree(root.left)
    invertTree(root.right)
    return root
```

Base Case: Checks if current node is None (i.e. the tree is empty or you've reached the end of a branch). The recursion stops for each specific node (doesn't stop entire program) when the "if not root" condition is triggered (i.e. the node is None, doesn't exist). This prevents further calls for nonexisting nodes.

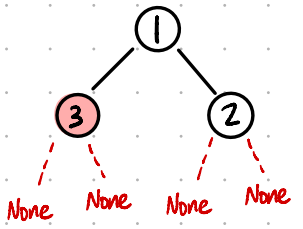
Recursive Calls: This calls the function on the left and right subtrees. The function keeps going deeper into the tree until it hits the base case. The entire left subtree is processed before moving to the right subtree.

Process



Start at Node 1:

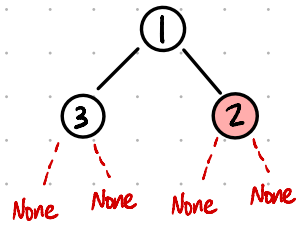
- func. processes ①, swaps it's children, and makes recursive calls for ③ and ②.



Process Node 3:

- ③ has no children (left = None, right = None)
- "if not root" triggers for both children, stopping recursion for ③

* When a recursive func. finishes processing a node and its children, it returns to the previous call (its "parent").



Return to Node 1, then process Node 2:

- ② also has no children
- "if not root" triggers for both children, stopping recursion for ②

Return to the first call for Node 1:

- Both left and right subtrees are done
- The func. returns the final inverted tree

* When the recursion has returned all the way back to the first call (the root of the tree), the entire function stops.

Here's the call stack: (order which func. is called and returned)

1. invertTree(1) → Starts at the root
2. invertTree(3) → Processes left child.
   - Stops for 3's children (None)
   - Returns to invertTree(1)
3. invertTree(2) → Processes right child
   - Stops for 2's children (None)
   - Returns to invertTree(1)
4. invertTree(1) → Returns the final tree