# Question

Given preorder and inorder traversal of a tree, construct the binary tree.

**Note:**  
You may assume that duplicates do not exist in the tree.

For example, given

preorder = [3,9,20,15,7]

inorder = [9,3,15,20,7]

Return the following binary tree:

3

/ \

9 20

/ \

15 7

# Solution

#### **How to traverse the tree**

There are two general strategies to traverse a tree:

* Breadth First Search (BFS)

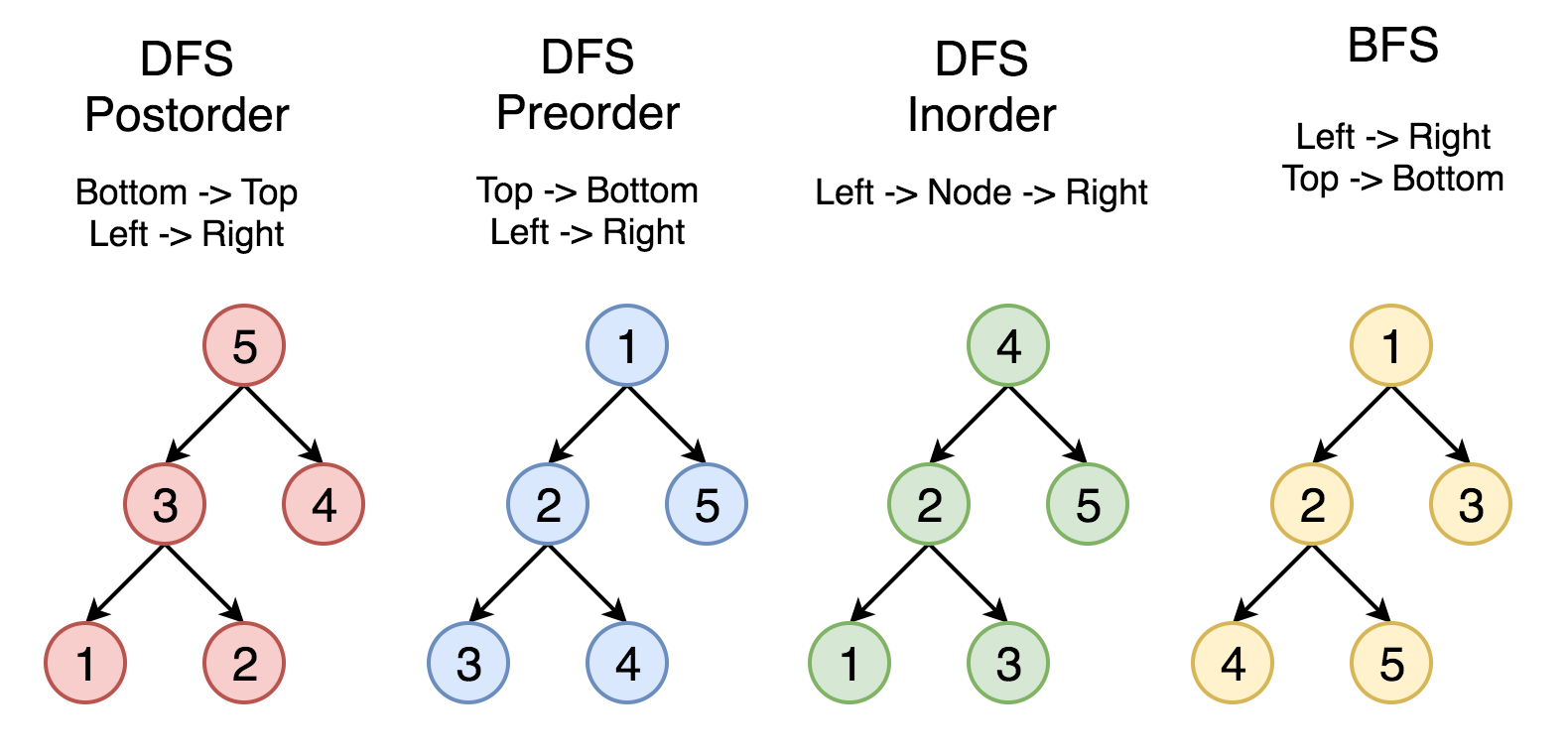
We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

* Depth First Search (DFS)

In this strategy, we adopt the depth as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

The DFS strategy can further be distinguished as preorder, inorder, and postorder depending on the relative order among the root node, left node and right node.

On the following figure the nodes are numerated in the order you visit them, please follow 1-2-3-4-5 to compare different strategies.



Here the problem is to construct a binary tree from its preorder and inorder traversal.

#### **Approach 1: Recursion**

**Tree definition**

First of all, here is the definition of the TreeNode which we would use.

|  |
| --- |
| // Definition for a binary tree node.  public class TreeNode {  int val;  TreeNode left;  TreeNode right;  TreeNode(int x) {  val = x;  }  } |

**Algorithm**

As discussed above the preorder traversal follows Root -> Left -> Right order, that makes it very convenient to construct the tree from its root.

Let's do it. The first element in the preorder list is a root. This root splits inorder list into left and right subtrees. Now one have to pop up the root from preorder list since it's already used as a tree node and then repeat the step above for the left and right subtrees.

1 / 12

|  |
| --- |
| **class Solution {**  **// start from first preorder element**  **int pre\_idx = 0;**  **int[] preorder;**  **int[] inorder;**  **HashMap<Integer, Integer> idx\_map = new HashMap<Integer, Integer>();**  **public TreeNode helper(int in\_left, int in\_right) {**  **// if there is no elements to construct subtrees**  **if (in\_left == in\_right)**  **return null;**  **// pick up pre\_idx element as a root**  **int root\_val = preorder[pre\_idx];**  **TreeNode root = new TreeNode(root\_val);**  **// root splits inorder list**  **// into left and right subtrees**  **int index = idx\_map.get(root\_val);**  **// recursion**  **pre\_idx++;**  **// build left subtree**  **root.left = helper(in\_left, index);**  **// build right subtree**  **root.right = helper(index + 1, in\_right);**  **return root;**  **}**  **public TreeNode buildTree(int[] preorder, int[] inorder) {**  **this.preorder = preorder;**  **this.inorder = inorder;**  **// build a hashmap value -> its index**  **int idx = 0;**  **for (Integer val : inorder)**  **idx\_map.put(val, idx++);**  **return helper(0, inorder.length);**  **}**  **}** |

**Complexity analysis**

* Time complexity : \mathcal{O}(N)O(*N*). Let's compute the solution with the help of [master theorem](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)) T(N) = aT\left(\frac{b}{N}\right) + \Theta(N^d)*T*(*N*)=*aT*(*Nb*​)+Θ(*Nd*). The equation represents dividing the problem up into a*a* subproblems of size \frac{N}{b}*bN*​ in \Theta(N^d)Θ(*Nd*) time. Here one divides the problem in two subproblemes a = 2, the size of each subproblem (to compute left and right subtree) is a half of initial problem b = 2, and all this happens in a constant time d = 0. That means that \log\_b(a) > dlog*b*​(*a*)>*d* and hence we're dealing with [case 1](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)#Case_1_example) that means \mathcal{O}(N^{\log\_b(a)}) = \mathcal{O}(N)O(*N*log*b*​(*a*))=O(*N*) time complexity.
* Space complexity : \mathcal{O}(N)O(*N*), since we store the entire tree.