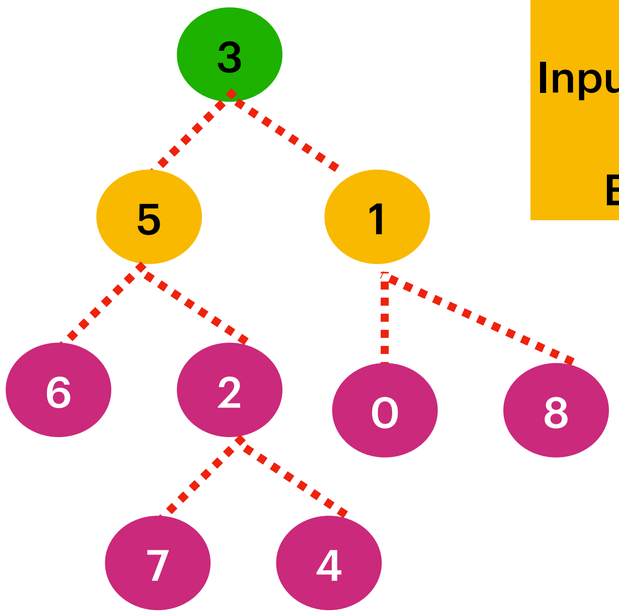
Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.
According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”



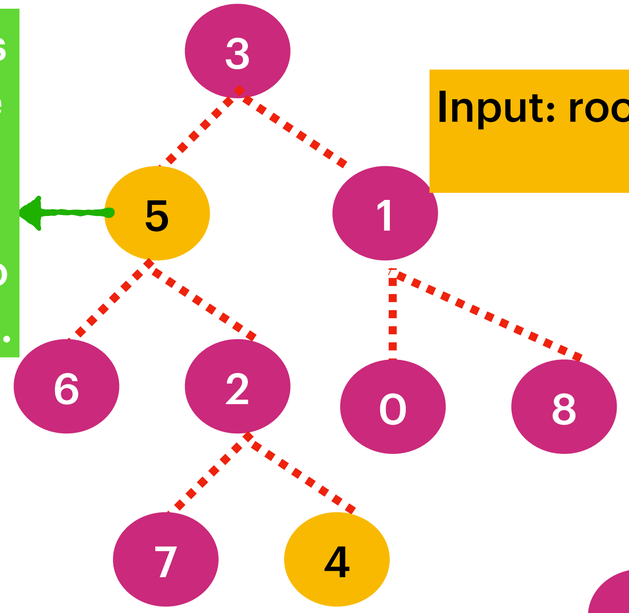
Example 1:
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.

Constraints:
The number of nodes in the tree is in the range [2, 105].
-109 <= Node.val <= 109
All Node.val are unique.
p != q
p and q will exist in the tree.

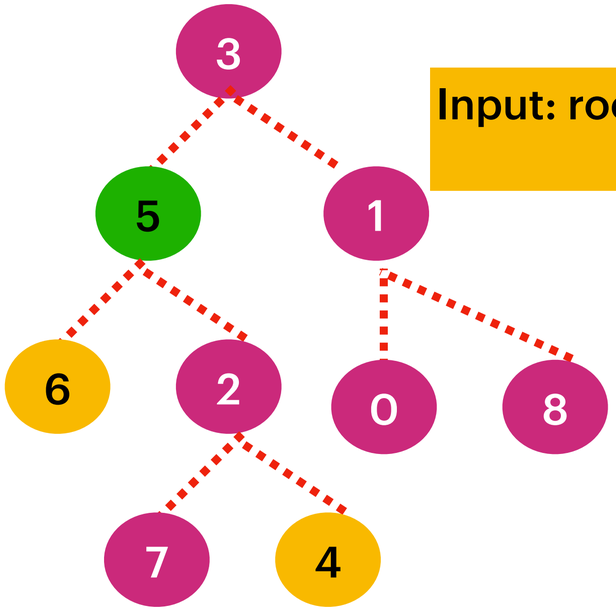
Lowest Common Ancestor [LCA] = Nearest Parent Node

Lowest Common Ancestor of a Binary Tree

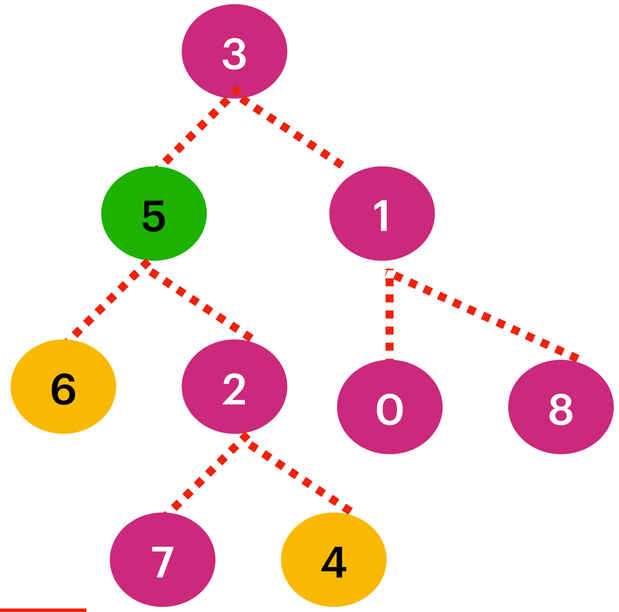
The LCA of nodes 5 and 4 is 5 since a node can be a descendant of itself according to the LCA definition.



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5



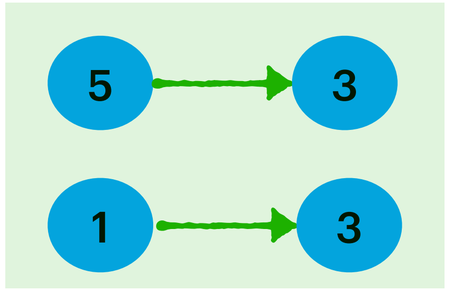
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 6, q = 4
Output: 5



P = 6

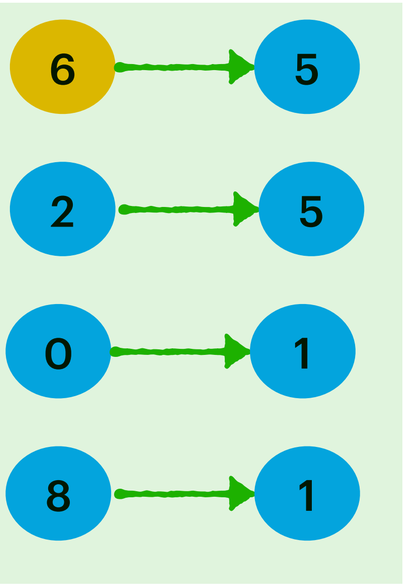
Q = 4

Map<TreeNode(child), TreeNode(parent)>



Step1

Map each node to it's parent util we found node(p) & node(q)

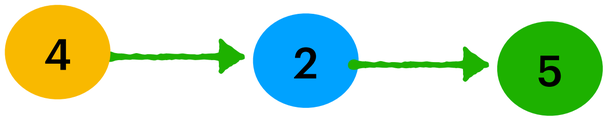


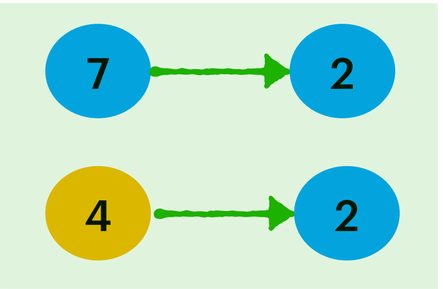
Step2

Add All the Ancestors of node(p) in the Set

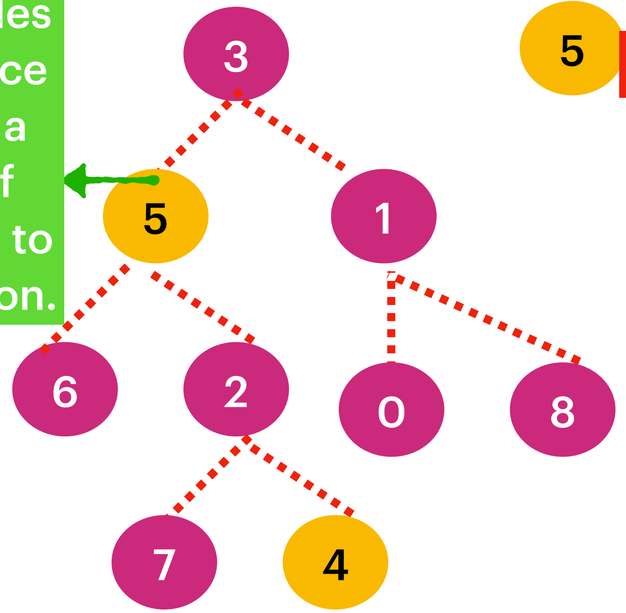
Step3

The first Ancestors of node(q), Present in Set is the LCA





The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.



5 P =

4 Q =

Map<TreeNode(child), TreeNode(parent)>

3 → Null

5 → 3

1 → 3

6 → 5

2 → 5

0 → 1

8 → 1

7 → 2

4 → 2

Step1

Map each node to it's parent util we found node(p) & node(q)

Step2

Add All the Ancestors of node(p) in the Set

5 → 3

Step3

The first Ancestors of node(q), Present in Set is the LCA

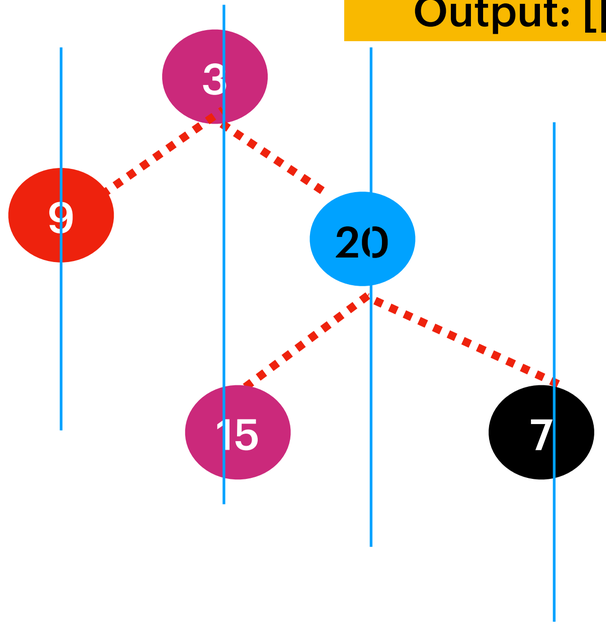
4 → 2 → 5

Binary Tree Vertical Order Traversal

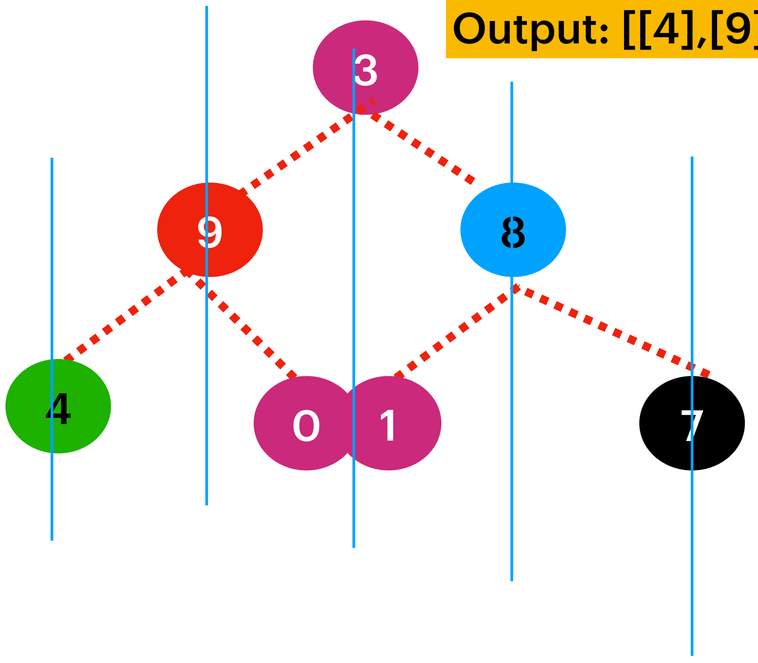
Given the root of a binary tree, return the vertical order traversal of its nodes' values. (i.e., from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from left to right.

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



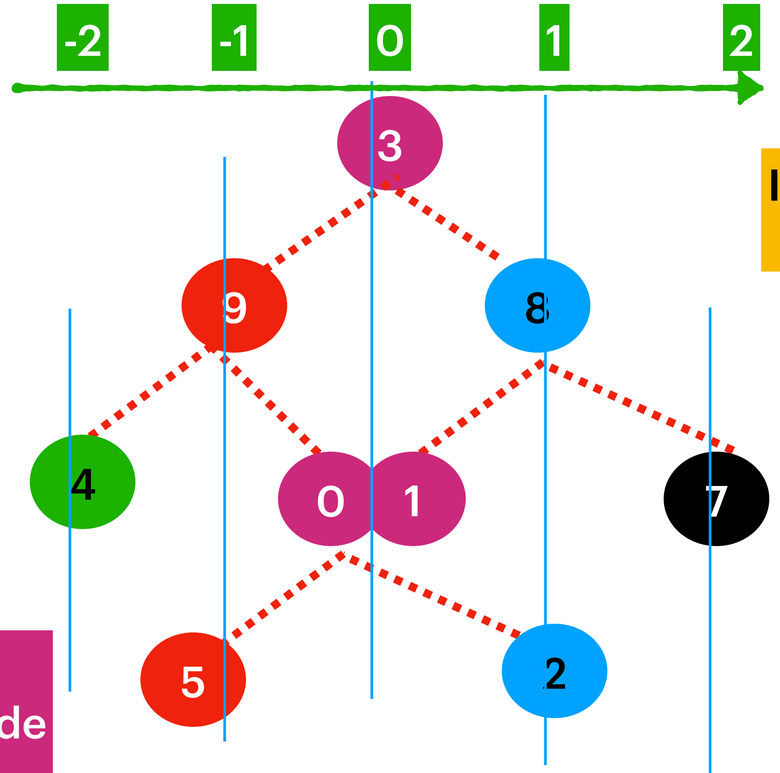
Input: root = [3,9,8,4,0,1,7]
Output: [[4],[9],[3,0,1],[8],[7]]



Constraints:

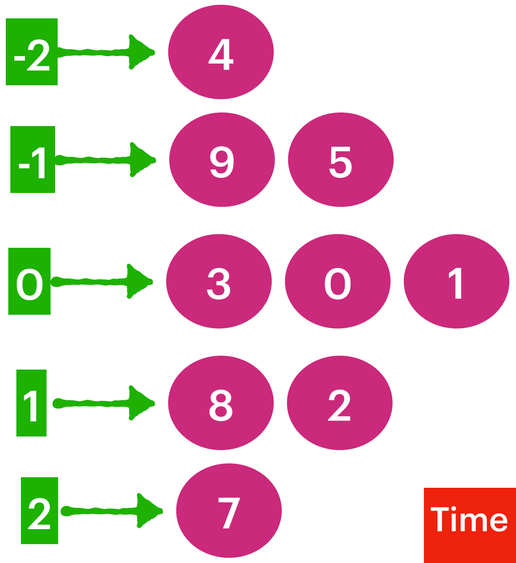
The number of nodes in the tree is in the range [0, 100].
-100 <= Node.val <= 100

Binary Tree Vertical Order Traversal



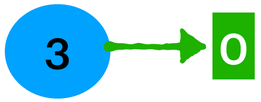
Input: root = [3,9,8,4,0,1,7,null,null,null,2,5]
Output: [[4],[9,5],[3,0,1],[8,2],[7]]

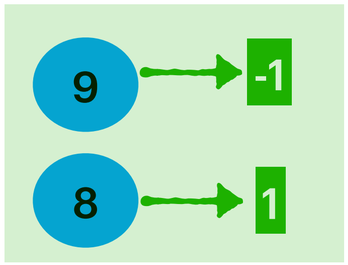
TreeMap<Integer [columNo], List<Integer> value>

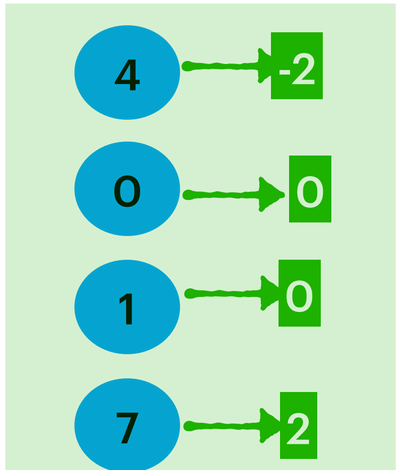


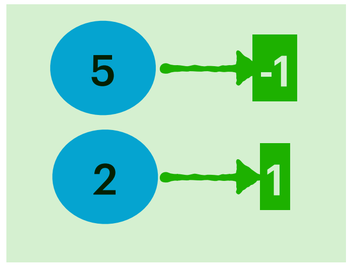
class Pair
TreeNode: node
column: int

Queue<Pair> [BFS]





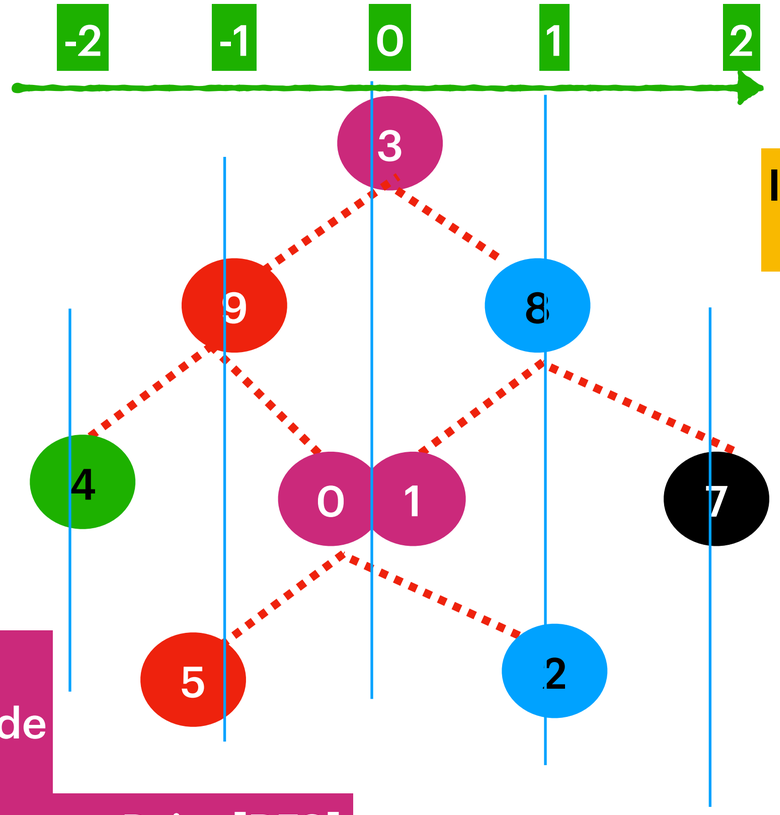




Time Complexity : BFS + TreeMap(sort)
O(n) + O(nlogn) = O(nlogn)

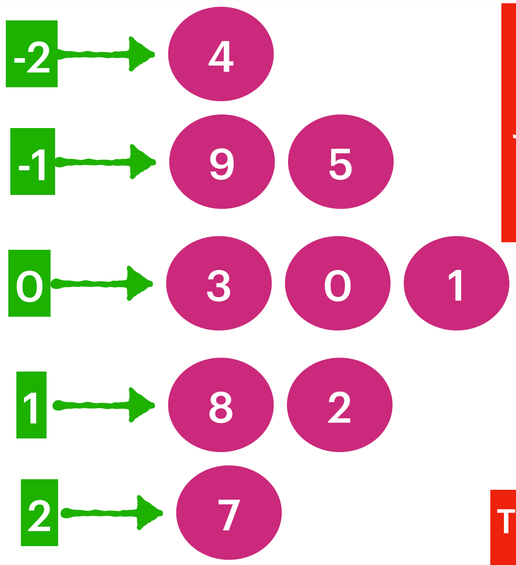
Space Complexity : Queue + Map
O(n) + O(n) => O(2n) = O(n)

Binary Tree Vertical Order Traversal



Input: root = [3,9,8,4,0,1,7,null,null,null,2,5]
Output: [[4],[9,5],[3,0,1],[8,2],[7]]

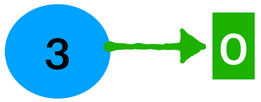
Map<Integer [columNo], List<Integer> value>

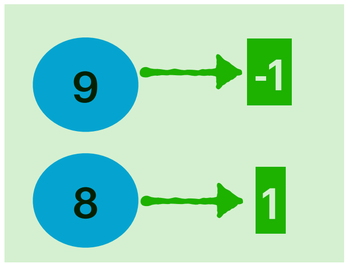


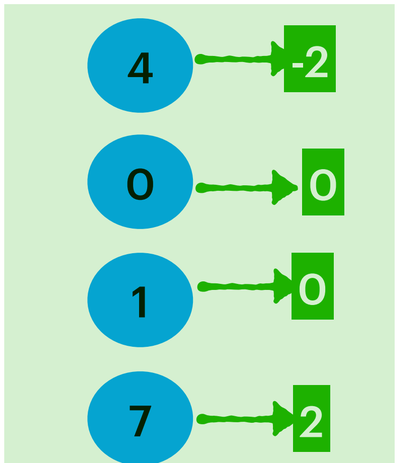
Make use of Map so that
no sorting.
Just maintain min & max index auxiliary variables so that
We don't need to sort

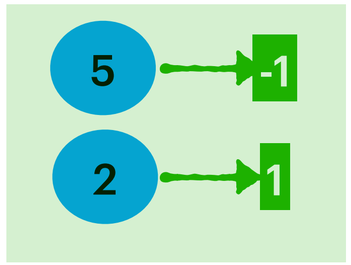
class Pair
TreeNode: node
column: int

Queue<Pair> [BFS]









Time Complexity : BFS + Map()
O(n) + O(n) = O(n)

Space Complexity : Queue + Map
O(n) + O(n) => O(2n) = O(n)