CSci 242: Algorithms and Data Structures  **Spring, 2020**

Instructor: Dr. M. E. Kim Date: February 19th, 2020

Due: by the end of day, February 26th (Wed.), 2020.

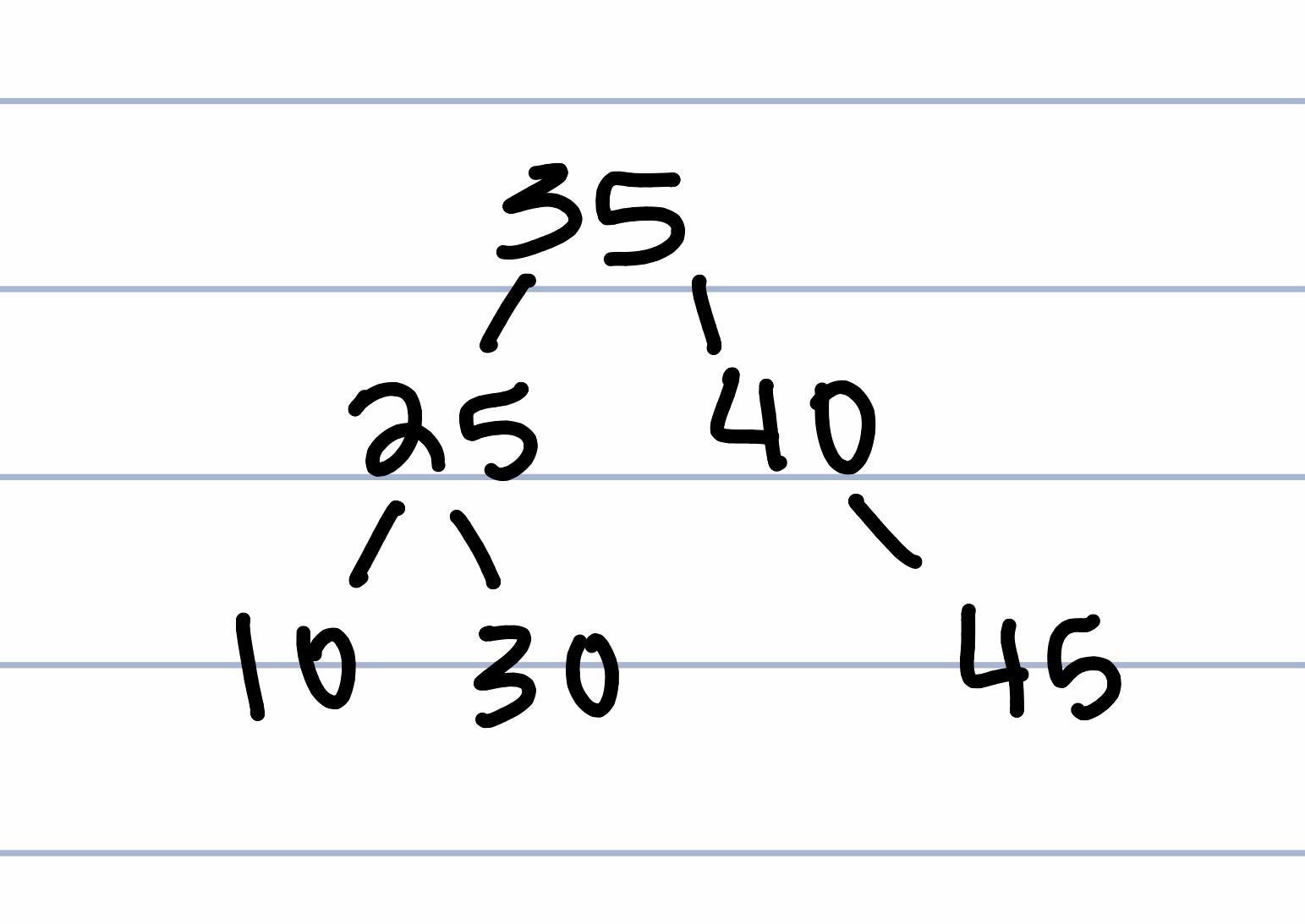
**Home Assignment 3: 50 points + 100(implementation, optional)**

Read the submission instruction in the blackboard and strictly follow it.

Q1. [10] **AVL Tree**

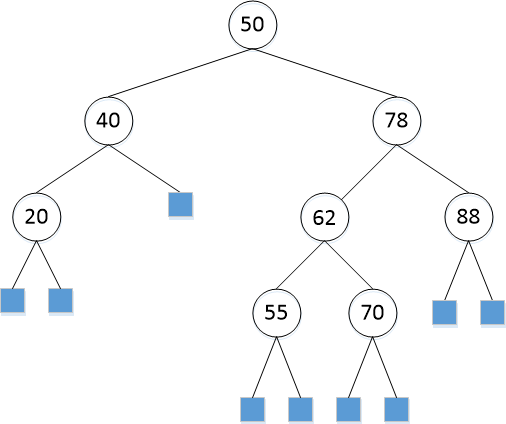
Consider the insertion of the following keys (in the given order) into an initially empty AVL tree:

10, 40, 25, 35, 30, 45, . Draw the final AVL tree.

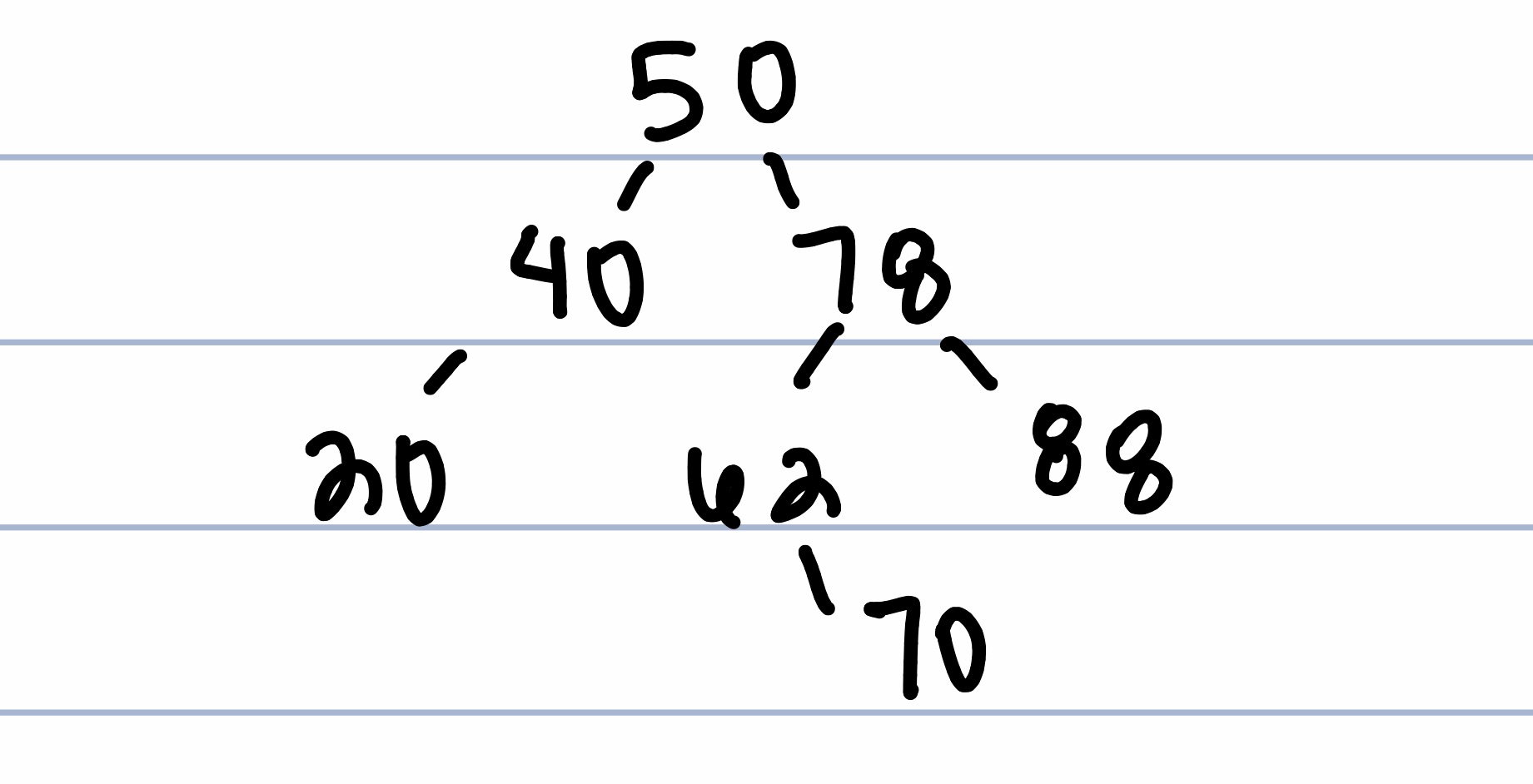


Q2. [10] **AVL Tree**

Draw the AVL tree resulting from the removal of the key 55 from the AVL tree of Figure below.



Removal of 55



Q2B. [50, optional]

For Q1 – Q2, implement the operations (insert, remove, restructure, etc.) of AVL tree in Python or in Java. Print the resulting tree from the root in the following format:

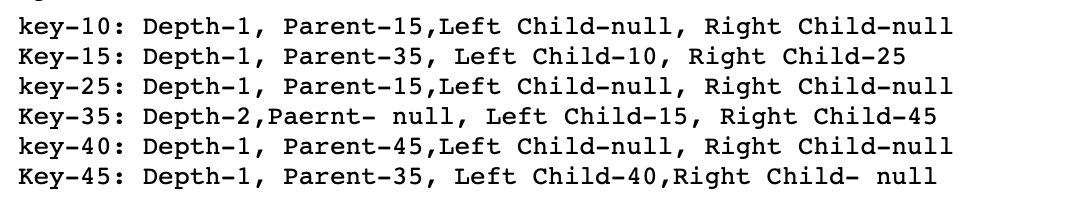
(a key of node: a depth of node, a key of parent, a key of left-child, a key of right-child)

e.g.) (50: 0, null, 40, 78)

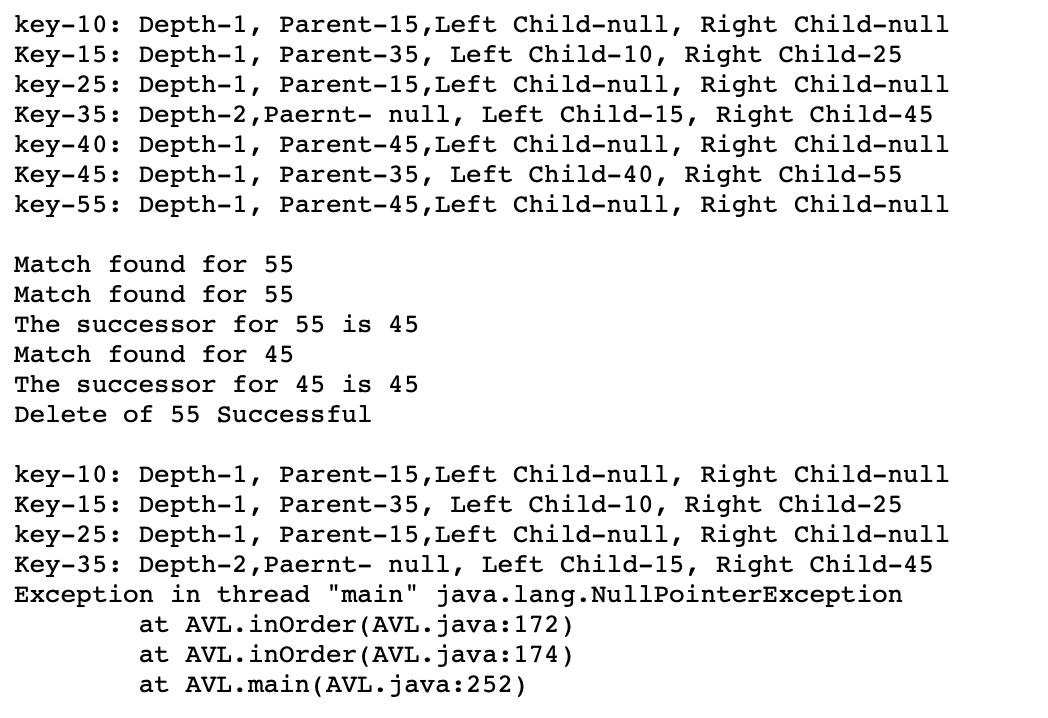
Since AVL tree is a balanced Binary Search Tree, you may be able to reuse your Python/Java codes of certain operations from HW 2.

Run your program and Include the image of your output to the corresponding question.

A = 10, 40, 25, 35, 30, 45



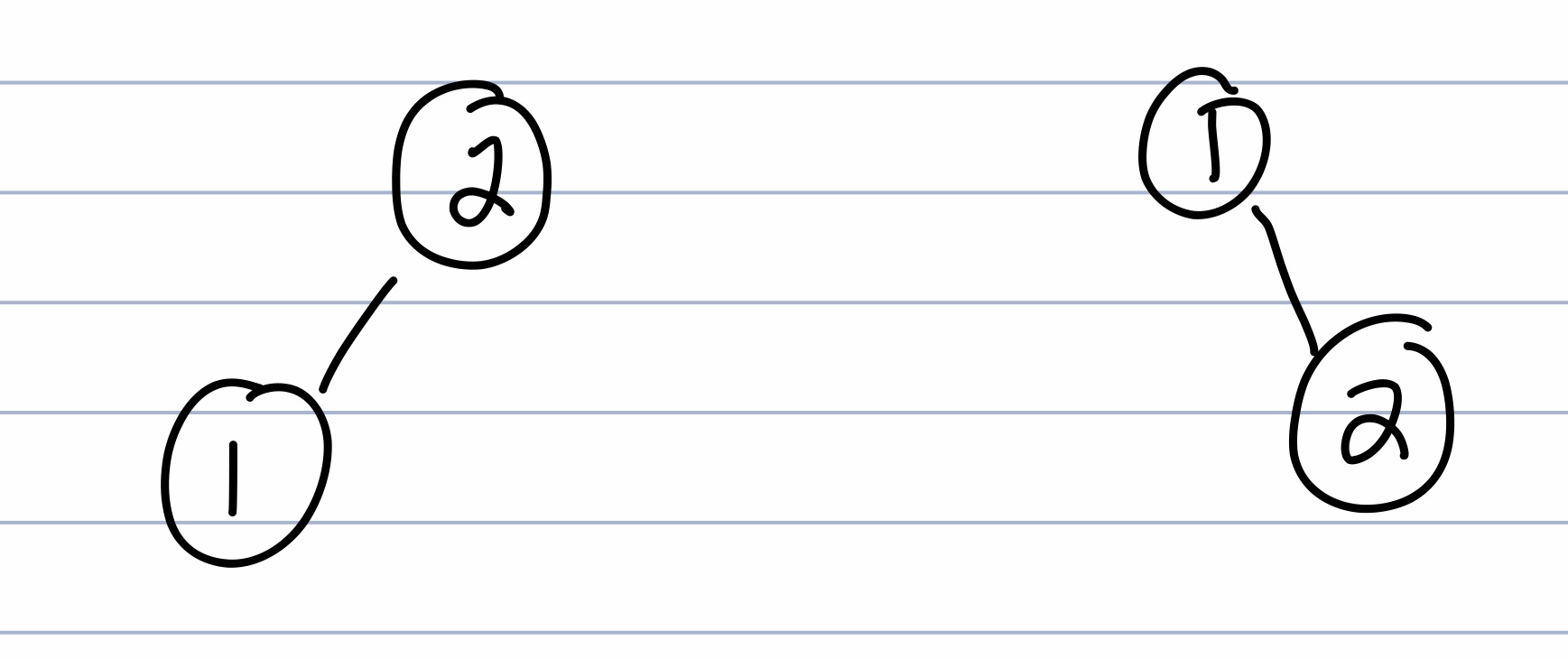
A = 10, 40, 25, 35, 30, 45, 55 (Added to 55 in order to complete the deletion



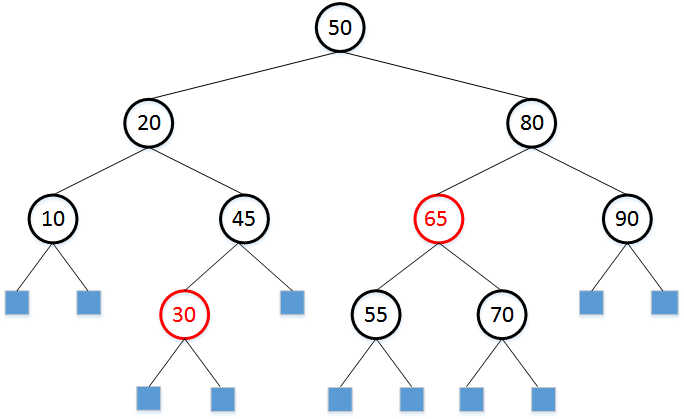
Q3. [10] **AVL Tree**

Professor Amadeus claims that the order of keys to be inserted into an AVL tree does not matter; i.e. the same tree always results every time. Give a counter example that contradicts Prof. Amadeus’s claim.

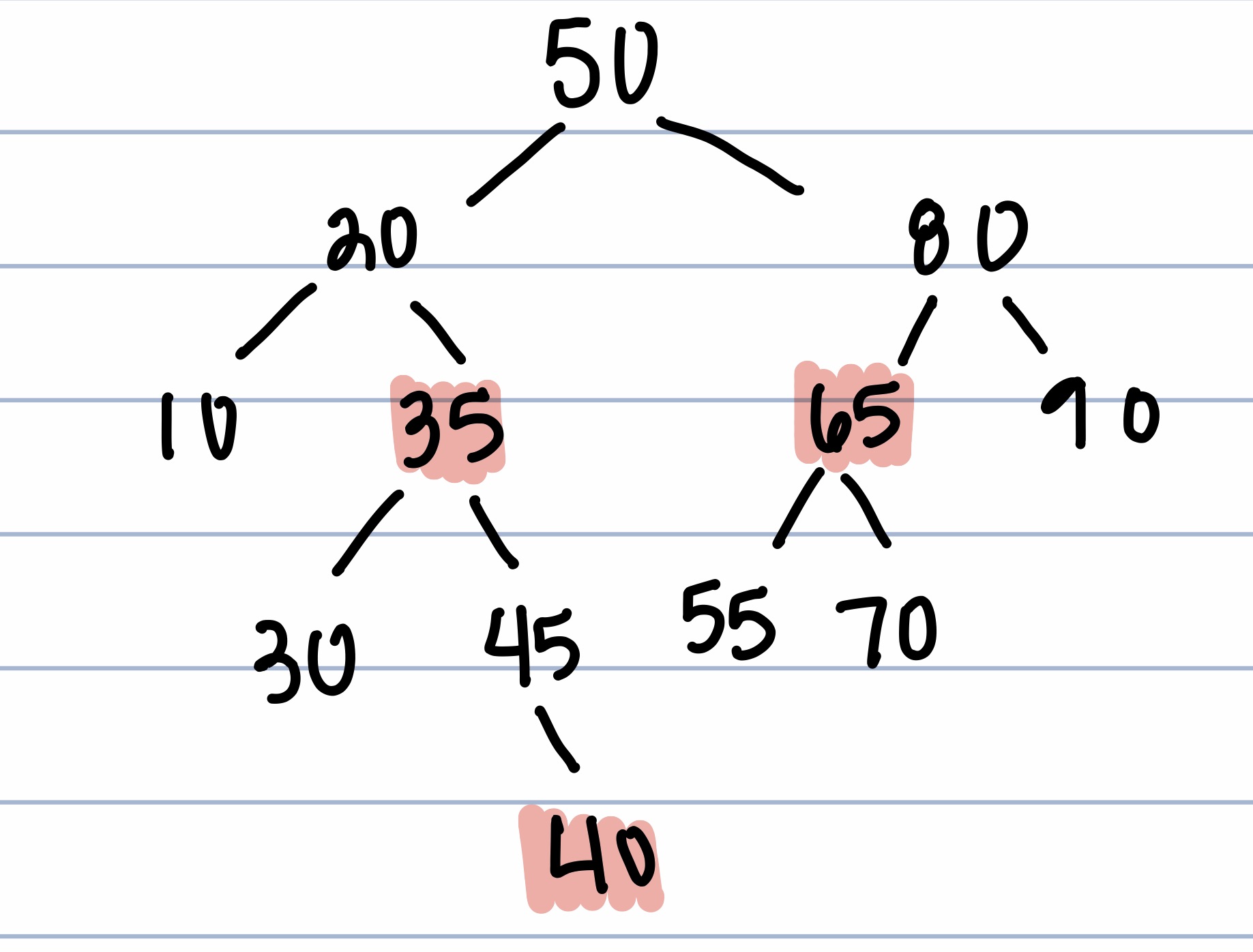
Given the numbers 2,1 they can go either way in a tree, thus proving that order does matter.



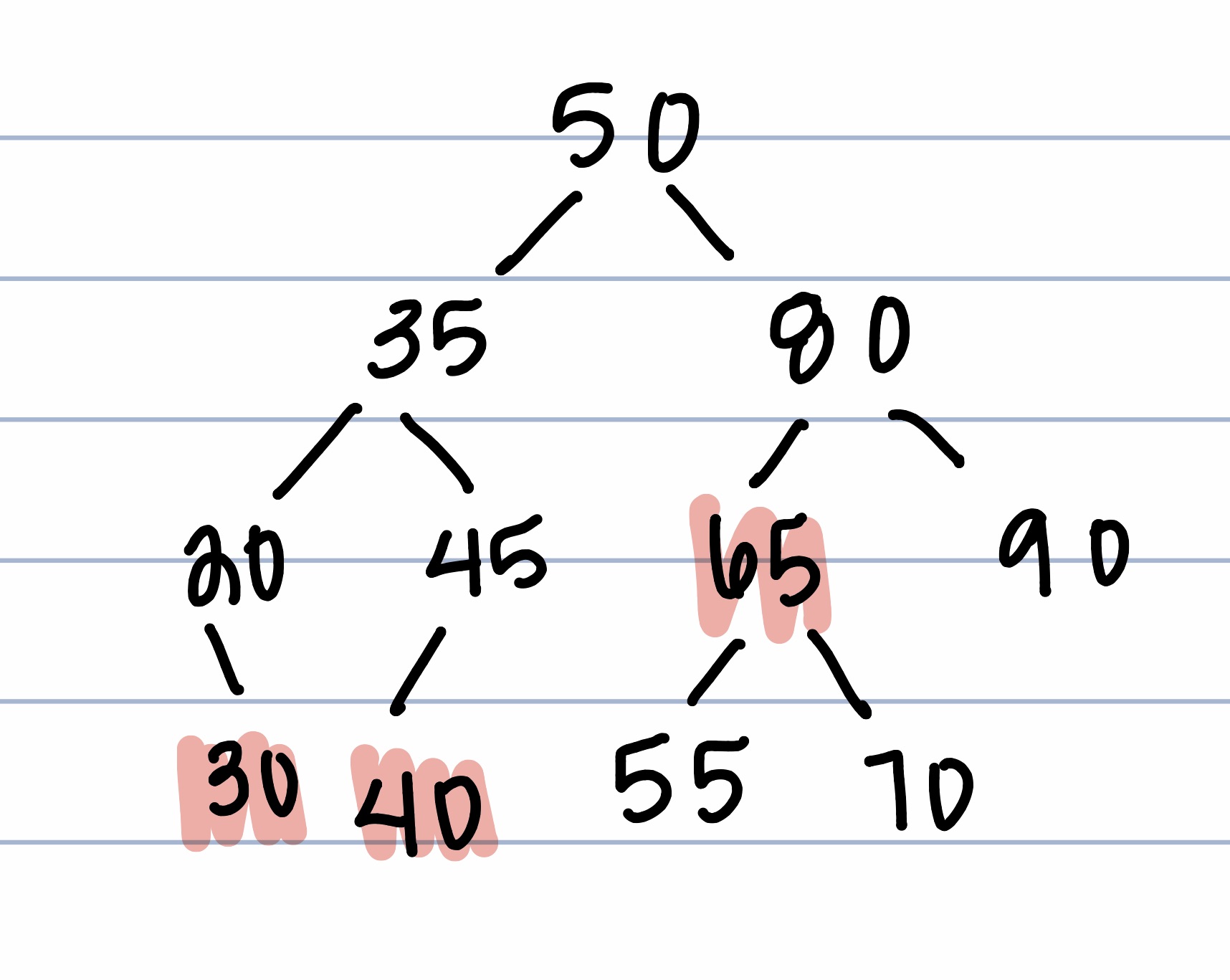
Q4. [20] **Red-Black Tree**



1. [10] Draw the Red-Black tree after each insertion of 40 and 35 into the given RB-tree above.



1. [10] In the given Red-Black Tree above, draw the resulting Red-Black tree after deleting 90.



Q4B. [50, optional] Similarly, implement the operations of Red-Black tree in Q4. Print the resulting tree in the form of:

((a key of node, color), depth, (a key of parent, color), (a key of left-child, color), (a key of right-child, color))

e.g.) ((80, B), 1, (50, B), (65, R), (90, B))

Similarly, you may be able to reuse your Python/Java codes of certain operations from HW 2.

Include the image of your output to the corresponding question.

