1. What is the Balanced Tree, Complete Tree and Non-Complete Tree?

Ans: **Balanced Tree:** A balanced binary tree is a binary tree in which the left and right subtrees of every node differ in height by no more than 1.That means no leaf is much farther away from the root than any other leaf.

**Complete Tree:** A complete binary tree is the one that is completely filled except possibly the last and all nodes are in the last level are as far left as possible. It can have between 1 and 2^h nodes at the last level h. That means the rightmost leaves have been removed.

**Non-Complete Tree:** It is binary tree, where in there are 2 children nodes at all levels except in last level but is filled from left.

2. Consider following data: {3,7,9,23,45,1,5,14,55,24,13,11,8,19,4,31,35,56}

a) Construct Binary Tree

b) Construct 2-3 Tree

c) Construct 2-3-4 Tree

d) Construct Binary Heap Tree

e) What is Time complexity of each case, Why would you use one versus the other?

f) Insert 17, 22, 32 in (b)

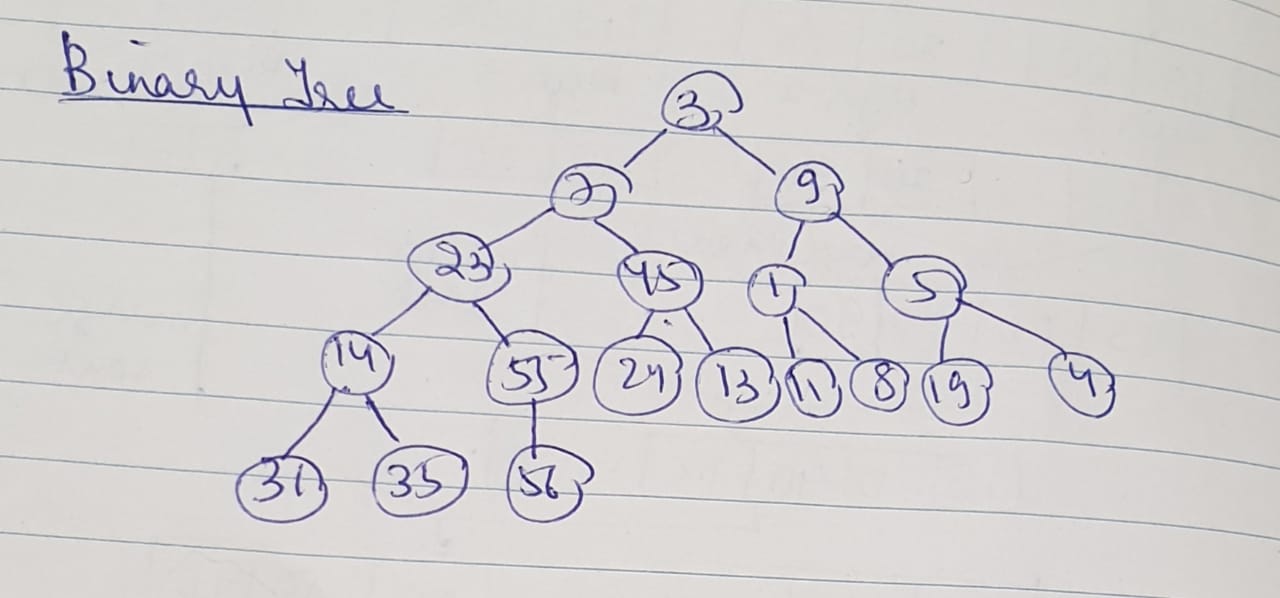
g) Delete 13 in (a) and (b)

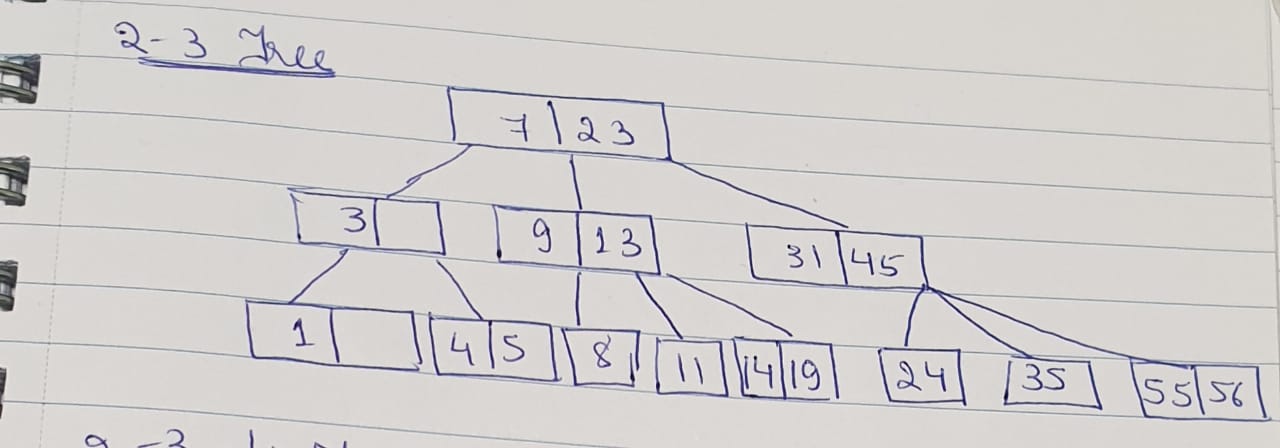
h) What is the Height of (a), (b), (c)?

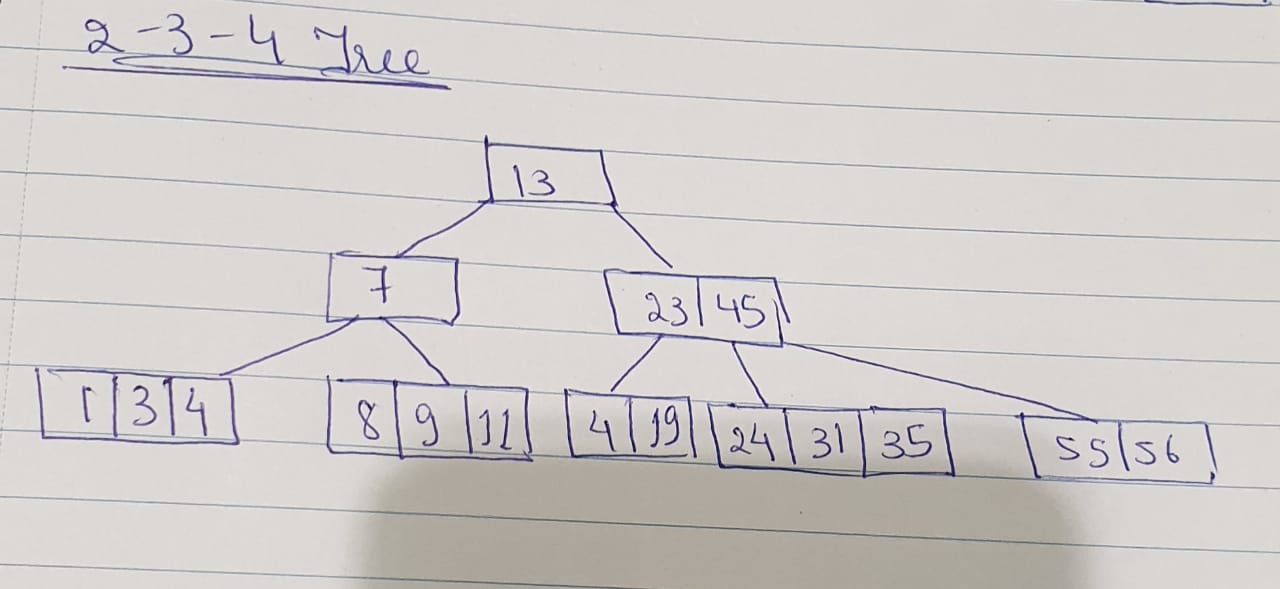
i) Write Java Search and Insert code for (a) and (b)

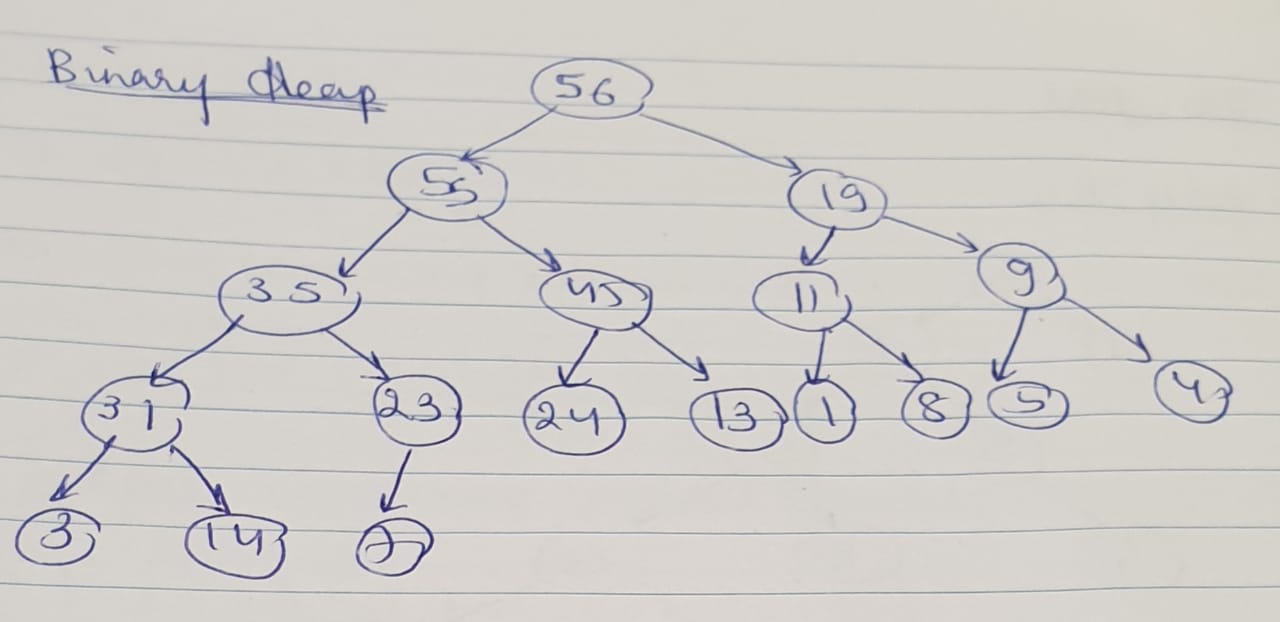
j) Write Java code for DeleteMin() and DeleteMax Algorithms for (a), provide example

Ans:

a) Binary Tree:

b) 2-3 Tree:

c) 2-3-4 Tree:

d) Binary Heap Tree:

e) What is Time complexity of each case, Why would you use one versus the other?

Time Complexity(Insertion)

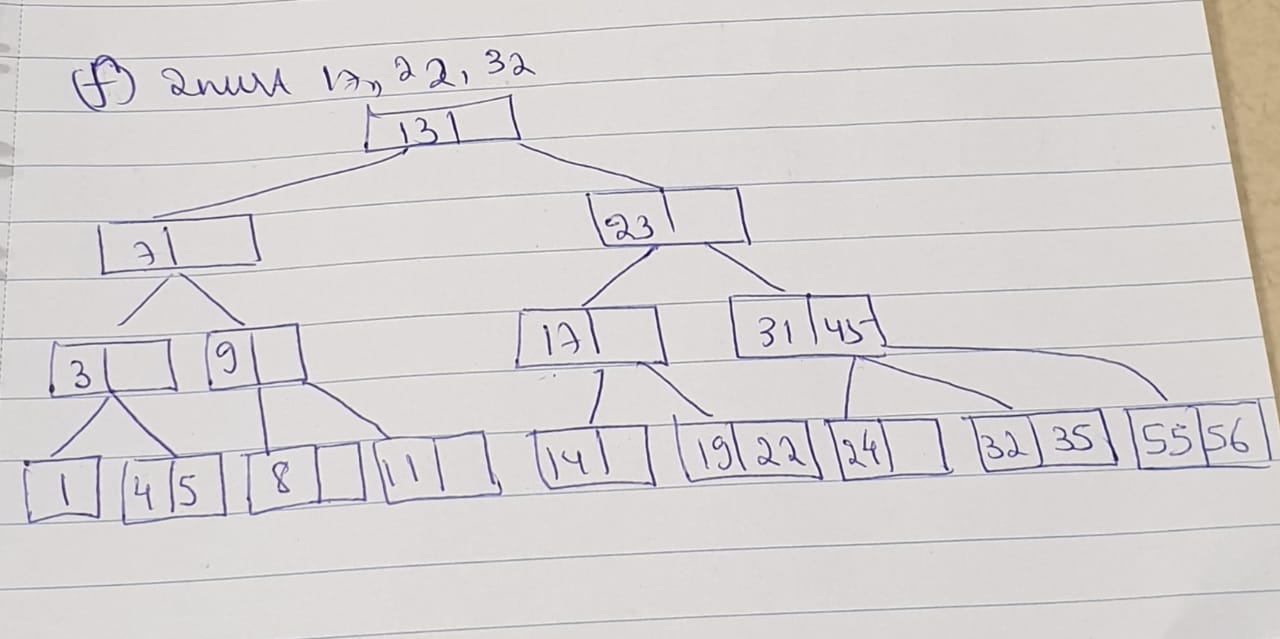
Binary Tree: **O(n)**

2-3 Tree: **O(Log n)**

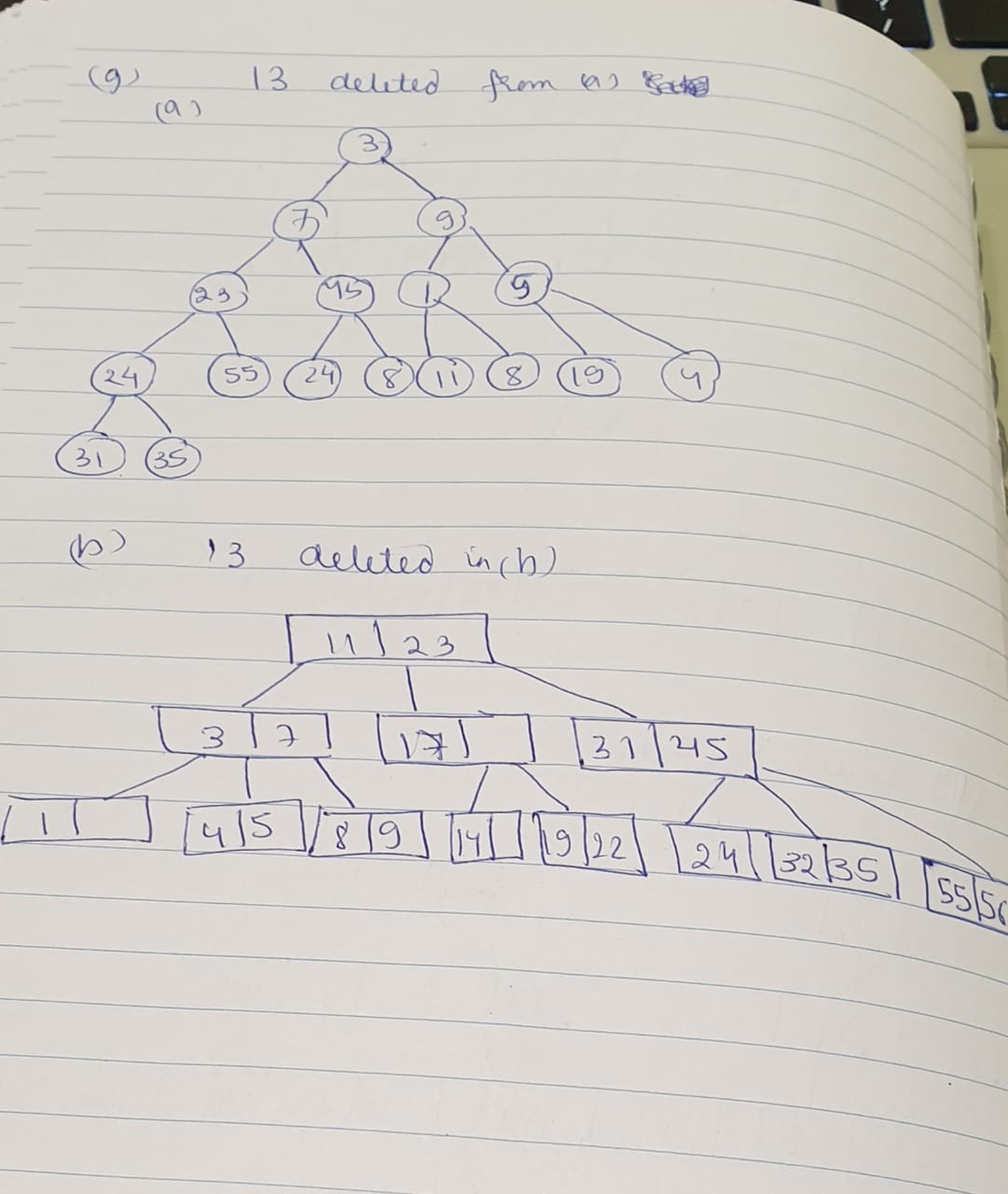
2-3-4 Tree: **O(Log n)**

Binary Heap Tree: **O(Log n)**

**2-3-4 Tree** is best because of reduced height in comparison to its counter-parts.

f) Insert 17, 22, 32 in (b)

g) Delete 13 in (a) and (b)



h) What is the Height of (a), (b), (c)?

Height:

Binary Tree: **4**

2-3 Tree: **2**

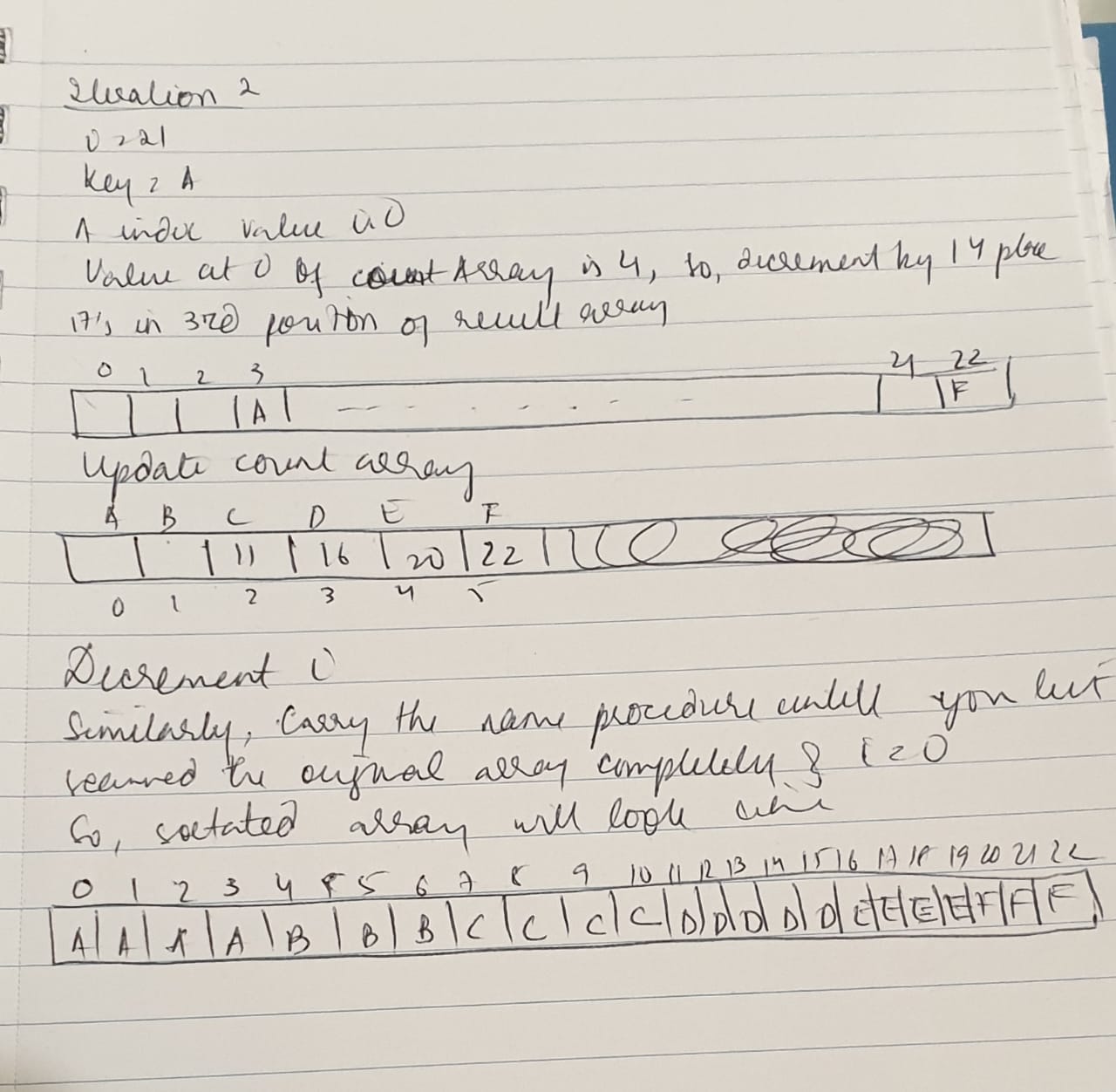
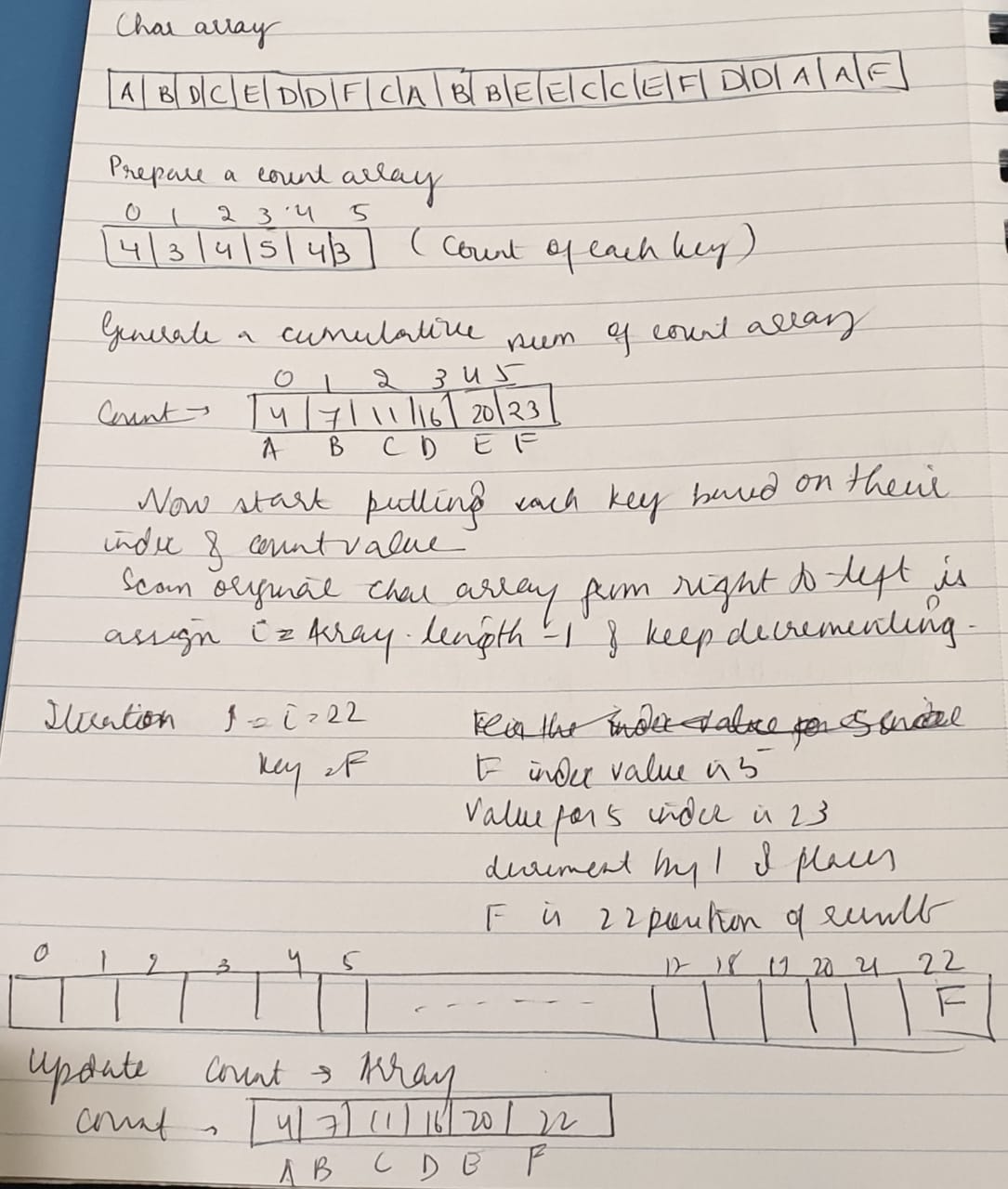
2-3-4 Tree: **2**

3. Consider the following string: “ABDCEDDFCABBEECCEFDDAAF”.

a) Use key-indexed counting sort algorithm to sort the string. Show each step

and the results.

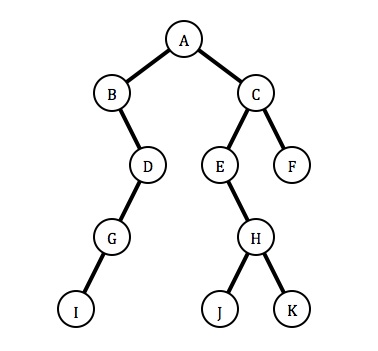
Ans:



b) What is the running time complexity of the algorithm as compared to Selection sort?

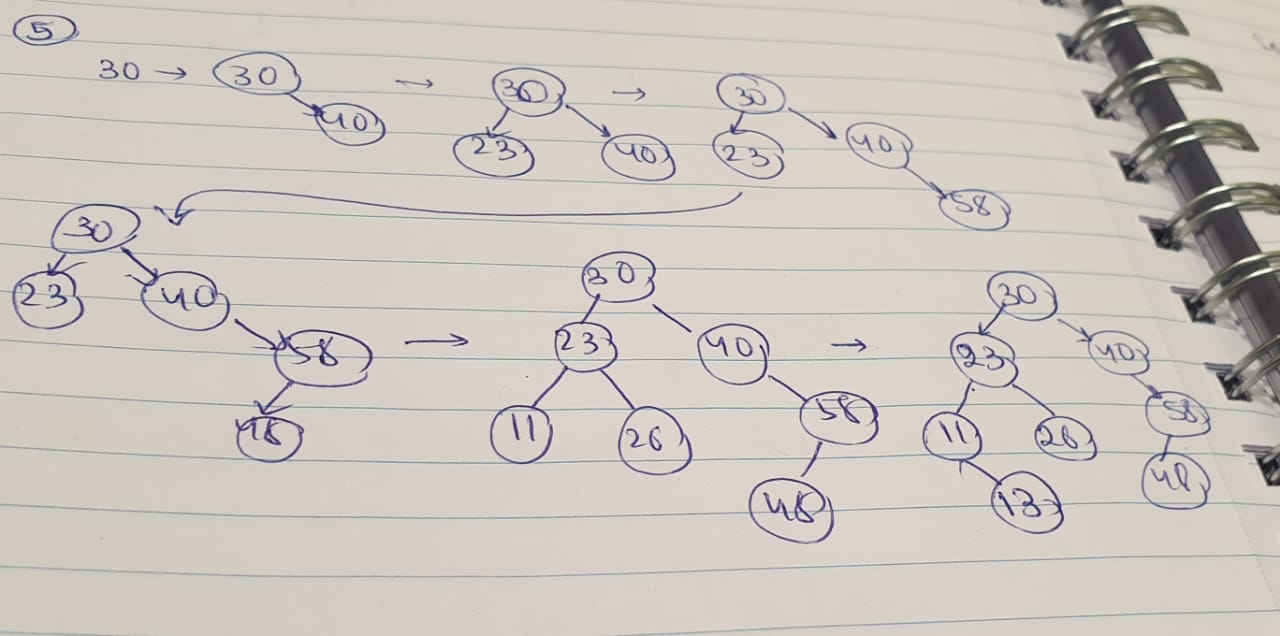
Ans: The time complexity of the Selection sort is: **O(N+R) ,** where N is the number of elements in input array and is the range of input.

4. Consider the following Binary tree, write Java code to find **maximum** element in binary search tree. You may write either a recursive or iterative implementation.



5. Insert the following items into an empty binary search tree in order:

{30, 40, 23, 58, 48, 26, 11, 13}

Ans:

6. What is the maximum height of a binary search tree in problem-5? Why?

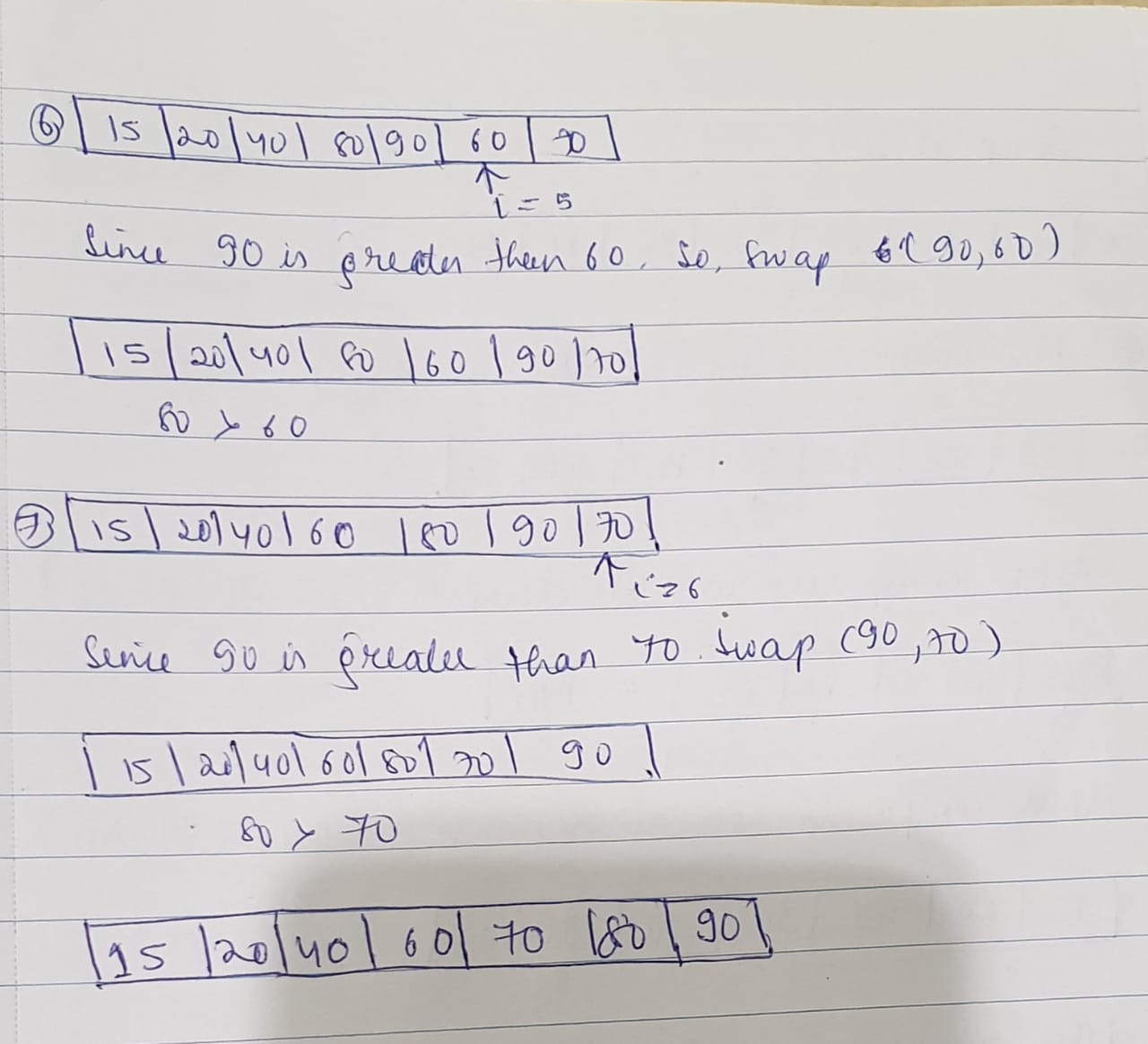
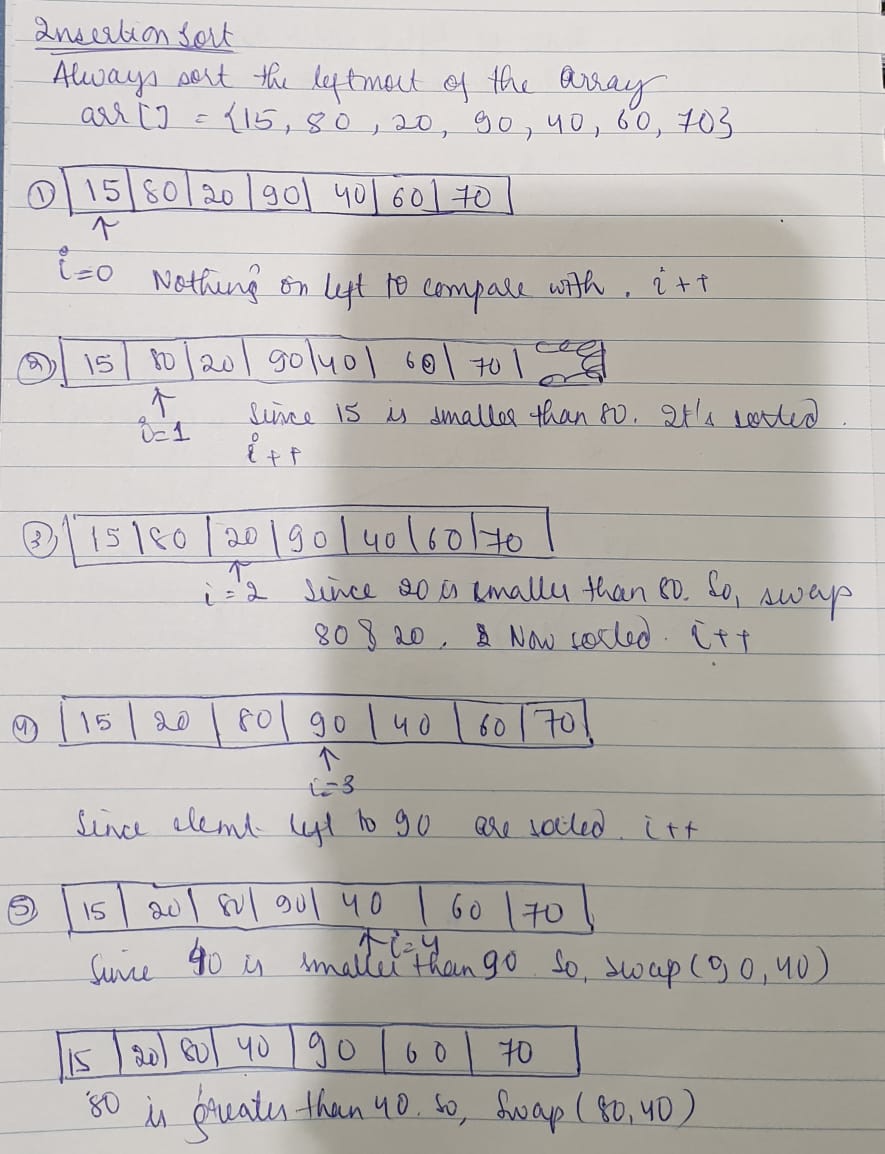
What is the time complexity of the Tree you built in problem-5? Why?

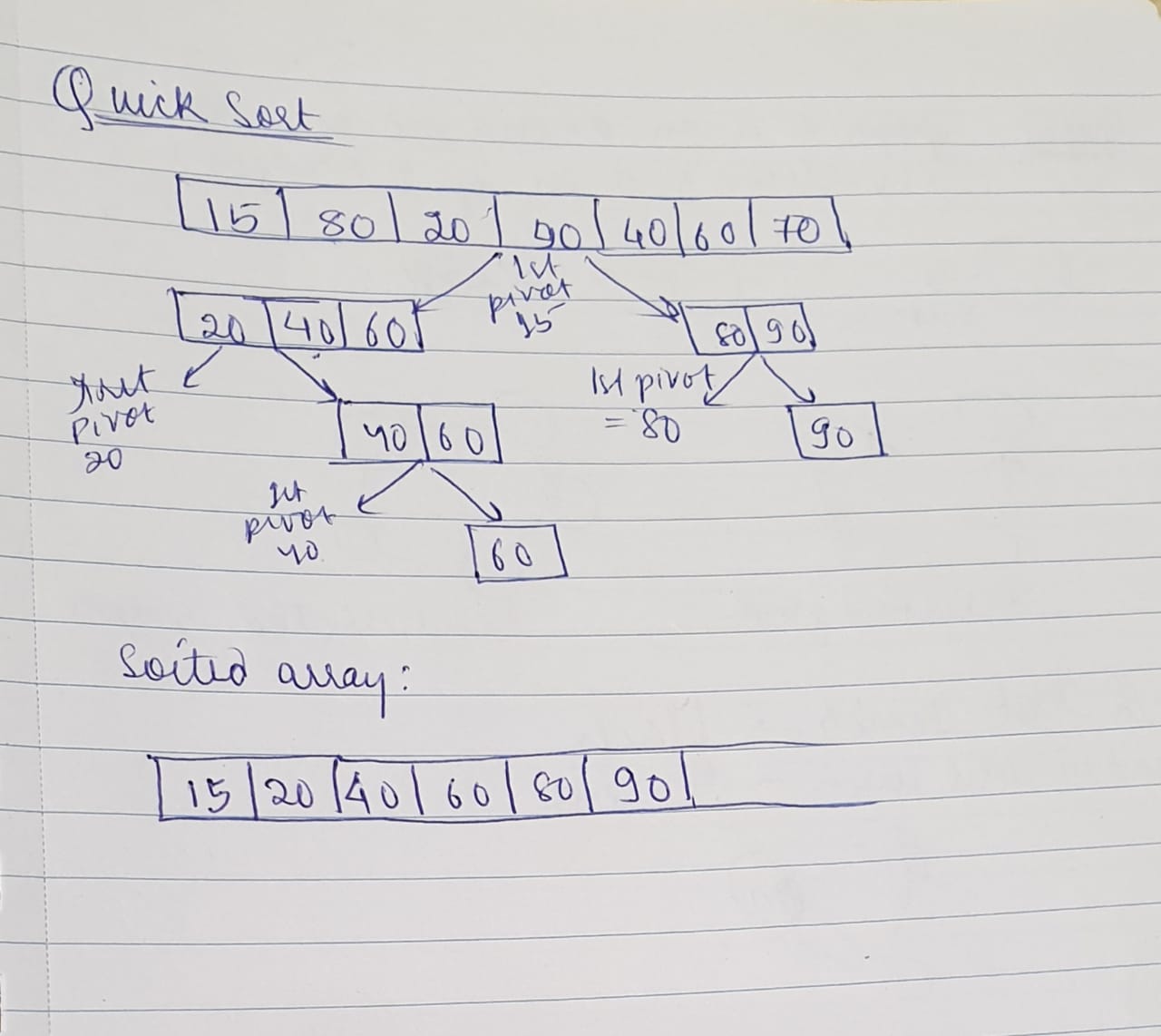
Ans: **Maximum height:** Maximum height of a tree can be determined by the maximum level of any node in the tree. The height of the tree in question 5 is **3** because the height of the tree’s formula is from the top till the leaf node.

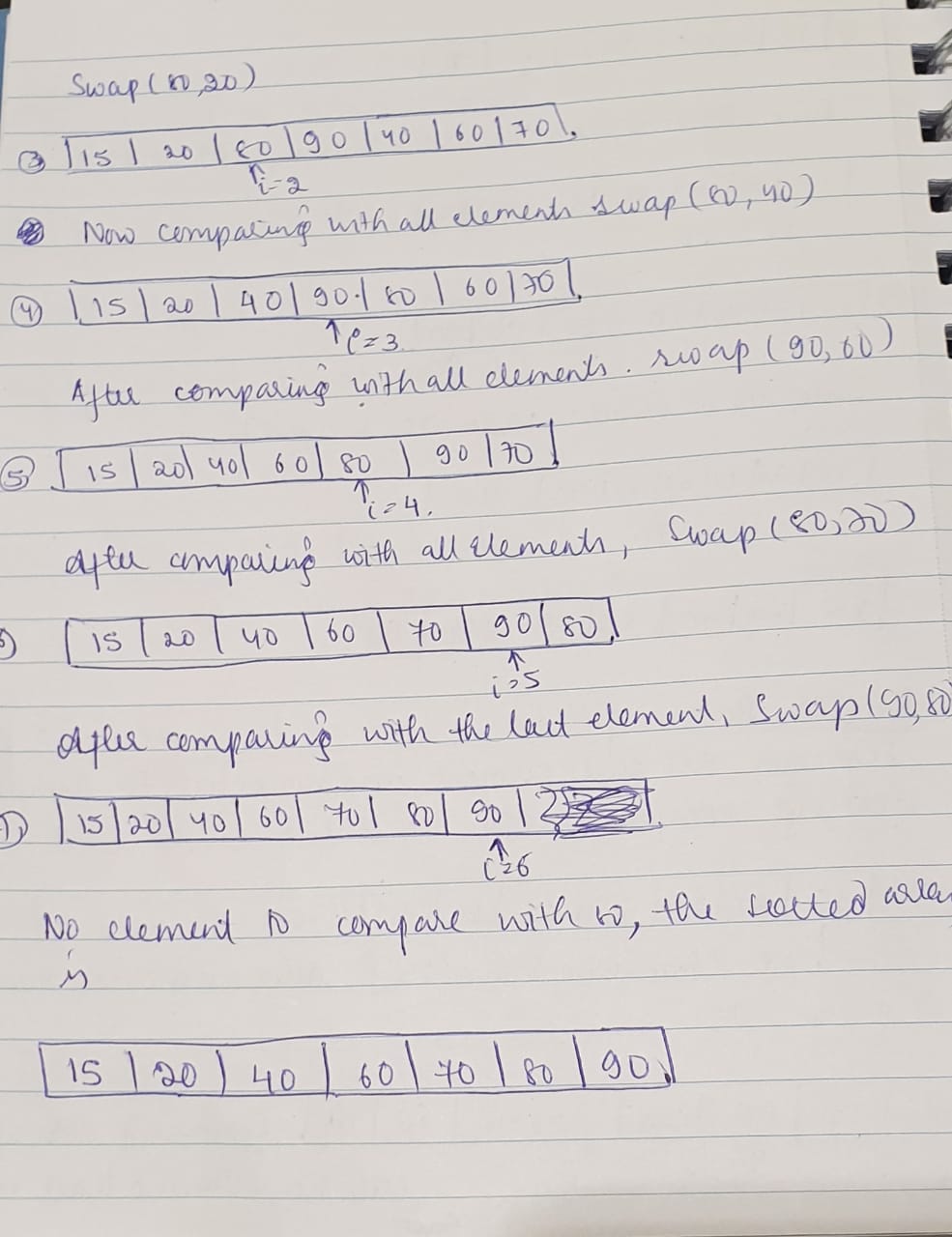
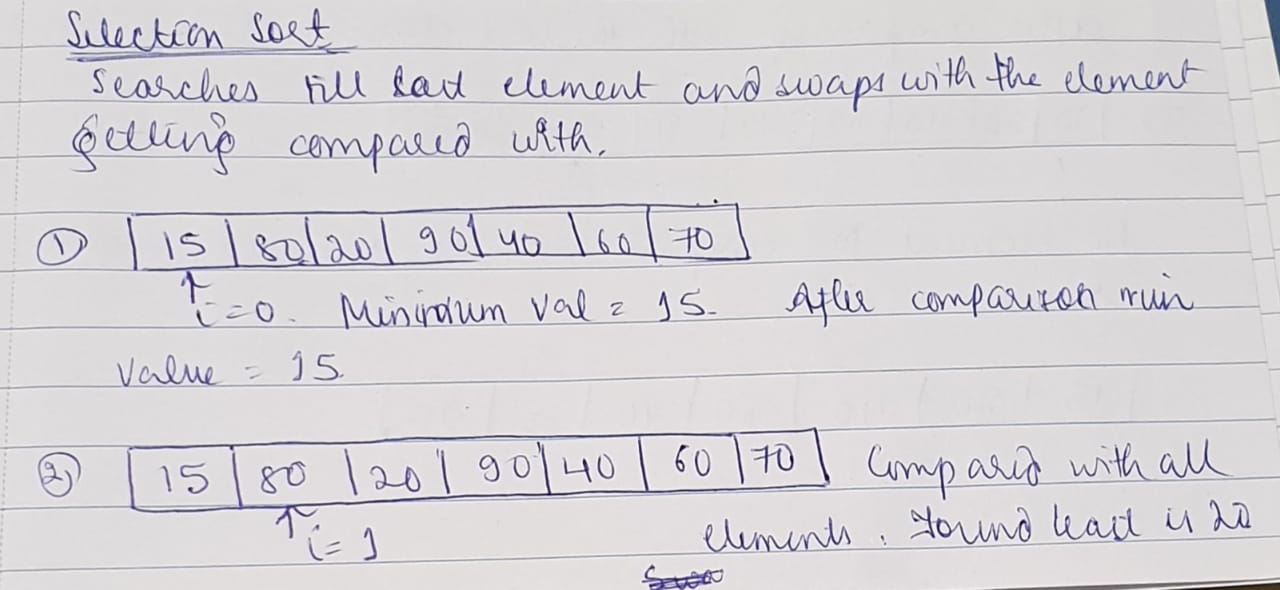
**Time Complexity:** The time complexity of the binary search tree is **O(N)** because we need to insert n nodes.

8. Consider arr[] = {15, 80, 20, 90, 40, 60, 70}. Walk through InsertionSort, SelectionSort, TimSort, and QuickSort algorithms, show step by step how the sort works.

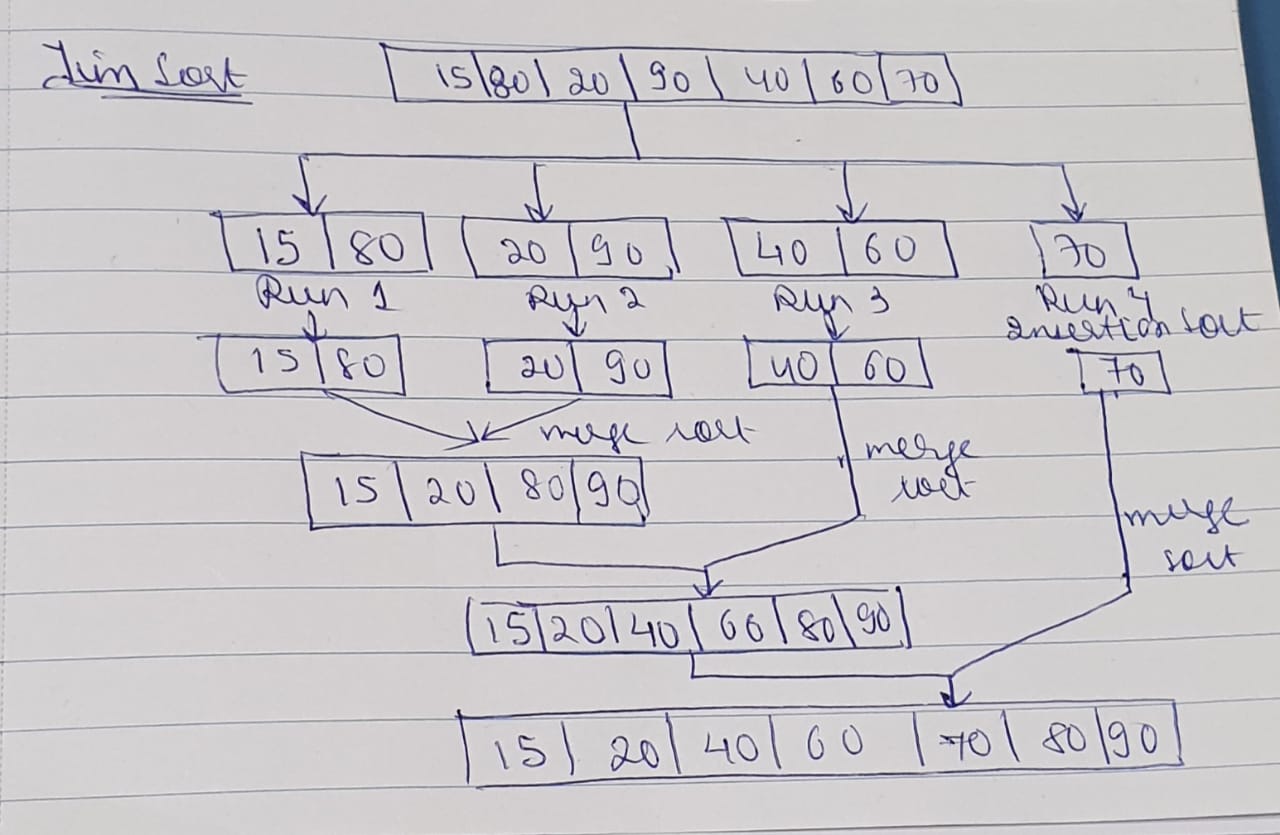
Ans:

Insertion Sort:

Selection Sort:



Quick Sort:

Tim Sort:

9. Consider attached image Boston.jpg. Write a program to sort the image Pixels by “brightness”. Use four sorting algorithms: InsertionSort, SelectionSort, TimSort, and QuickSort. You need to sort the Pixel array size of the image in Descending order and compare and show the runtime time complexity of each Sorting algorithm.

Ans: Time Complexity:

Insertion Sort: **O(N^2)**

Selection Sort: **O(N^2)**

Tim Sort: **O(N Log N)**

Quick Sort: **O(N^2)**