Muhammed Huseyin AYDIN

22203683

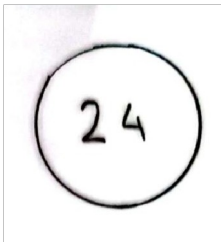
Section : 1

Assignment: 2

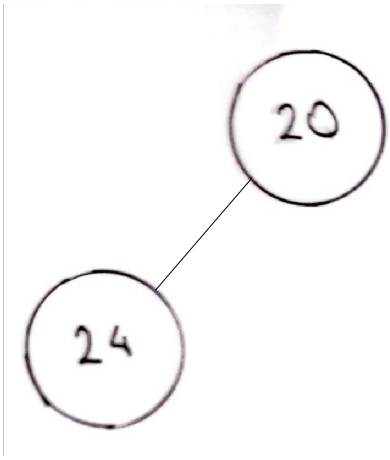
Question 1 Answers

Part A

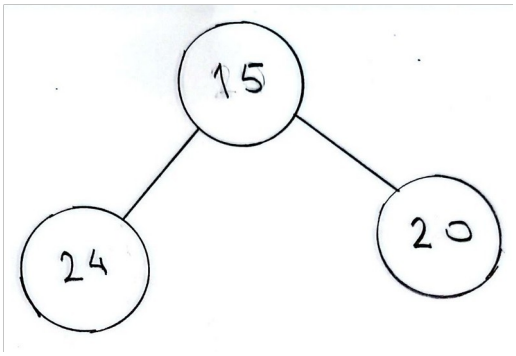
Insert 24:



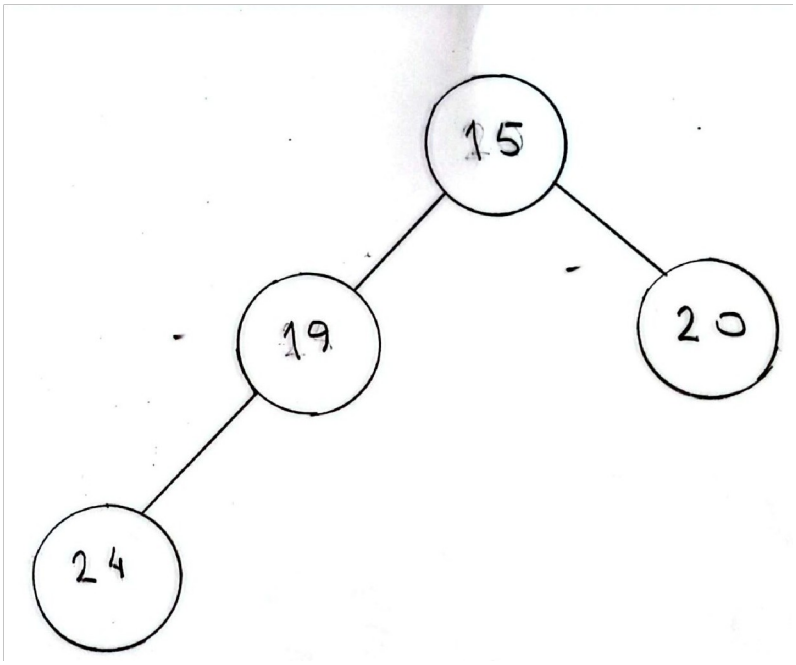
Insert 20:



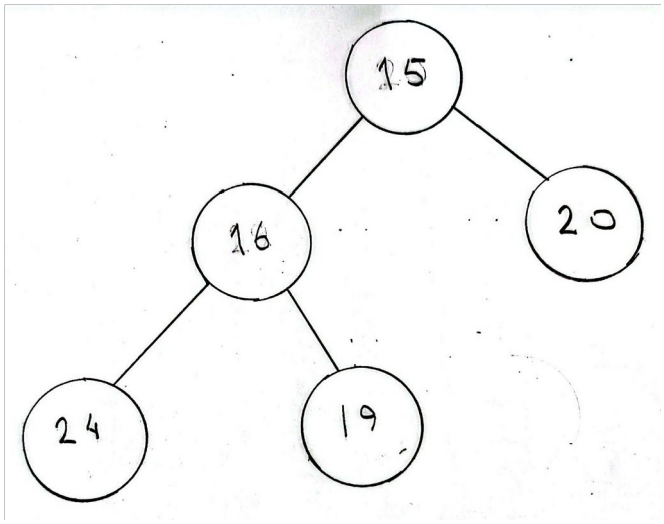
Insert 15:



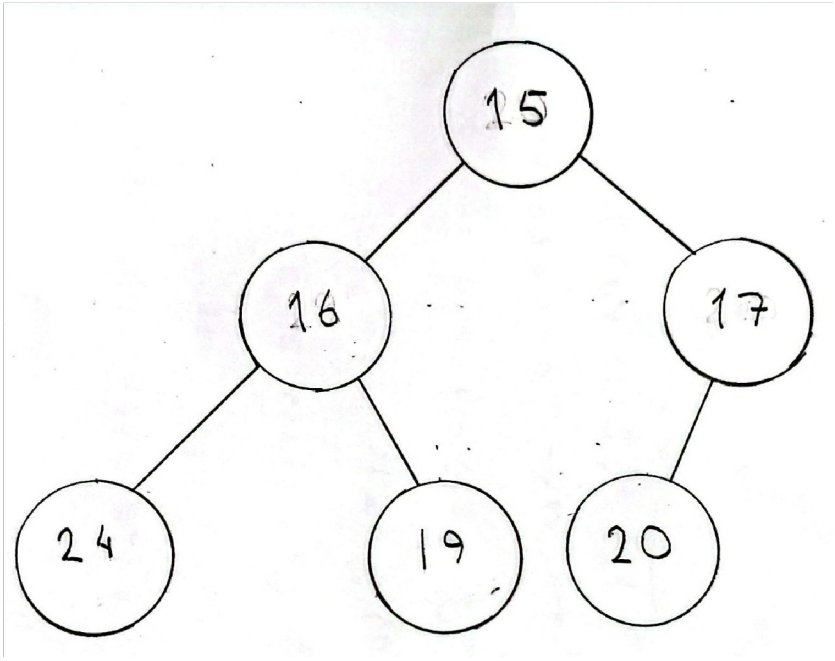
Insert 19:



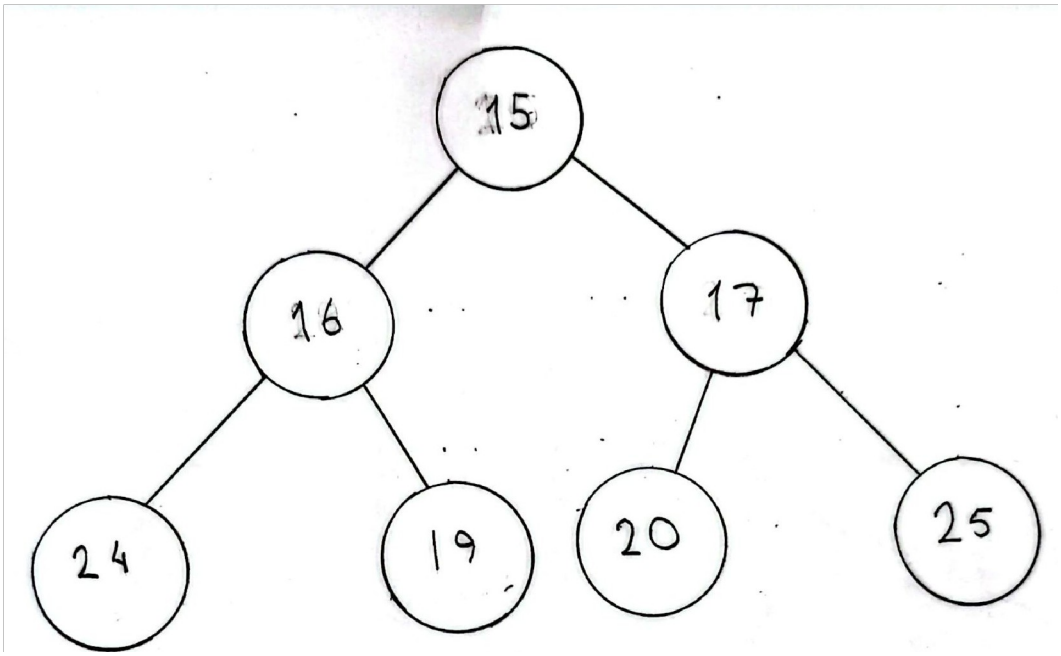
Insert 16:



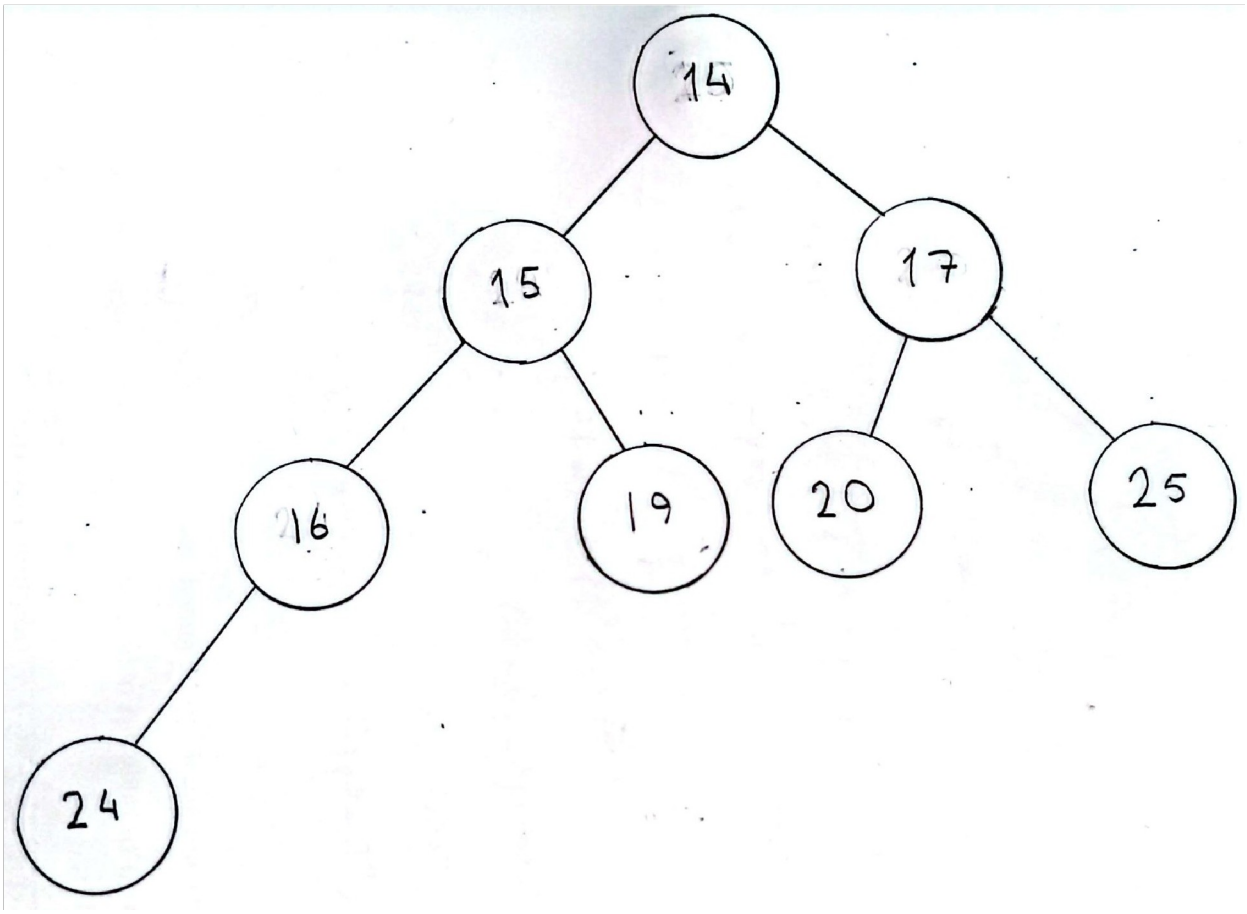
Insert 17:



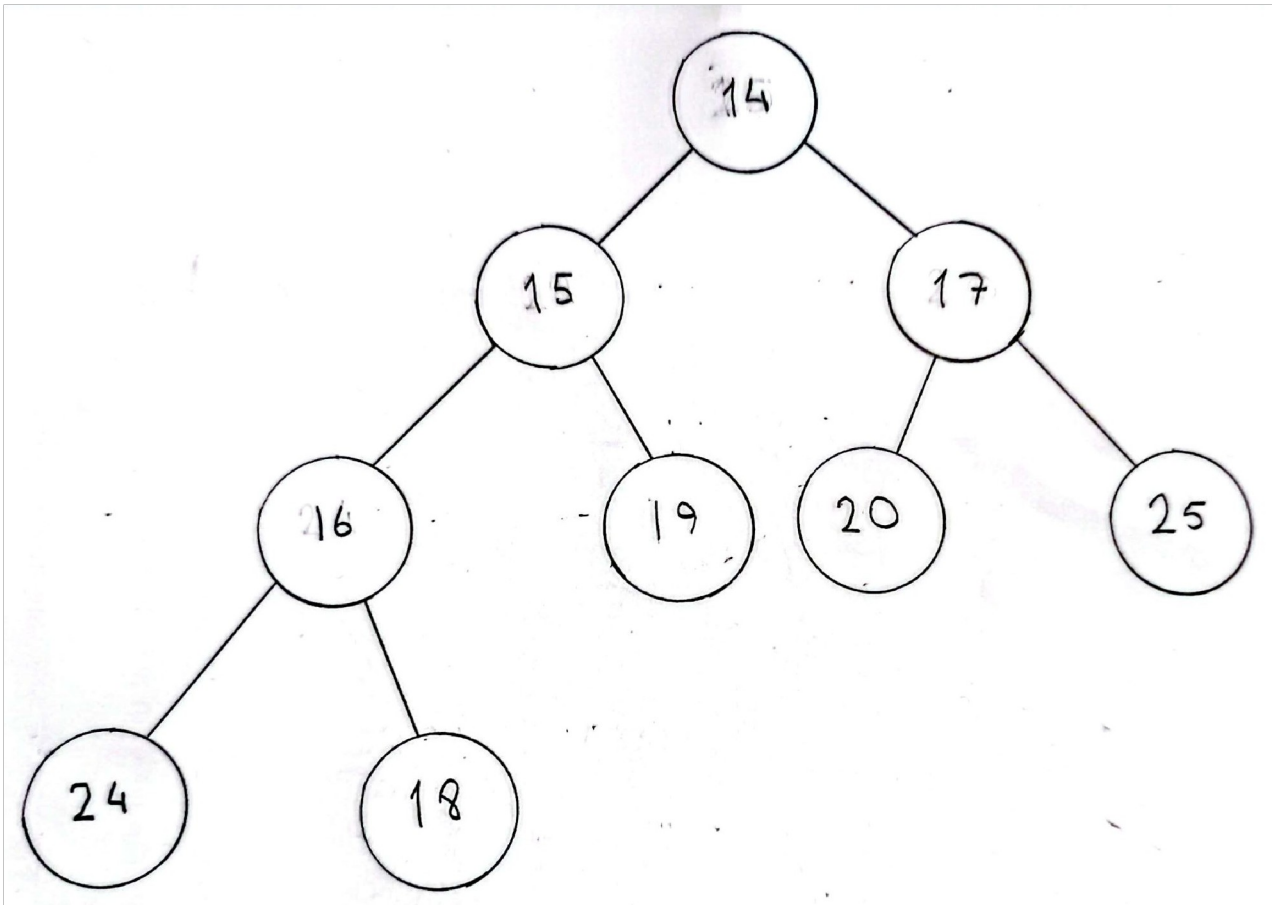
Insert 25:



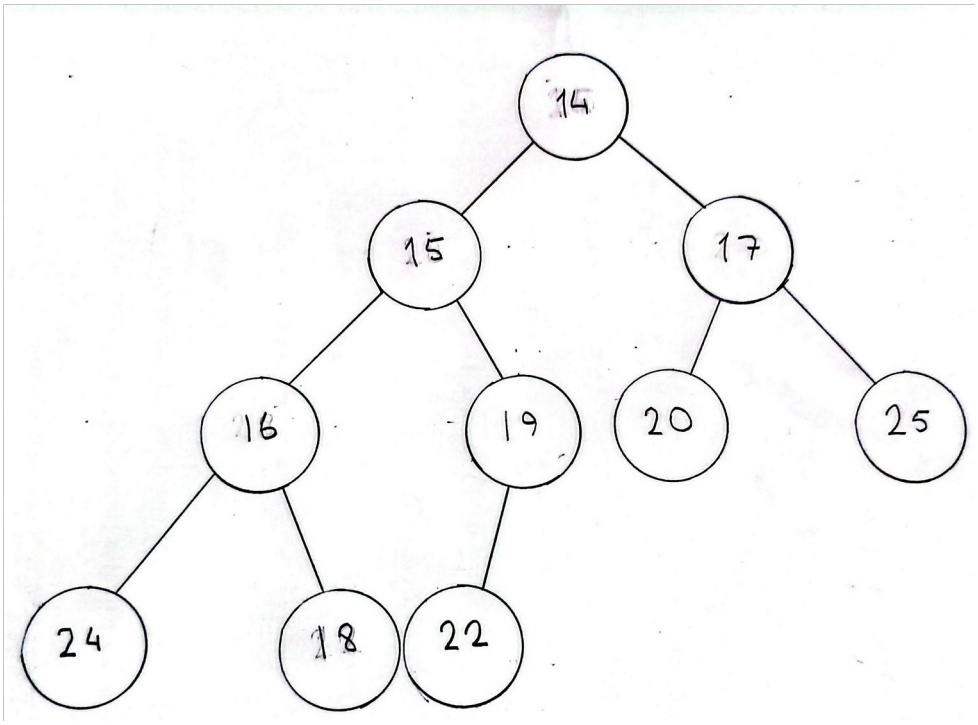
Insert 14:



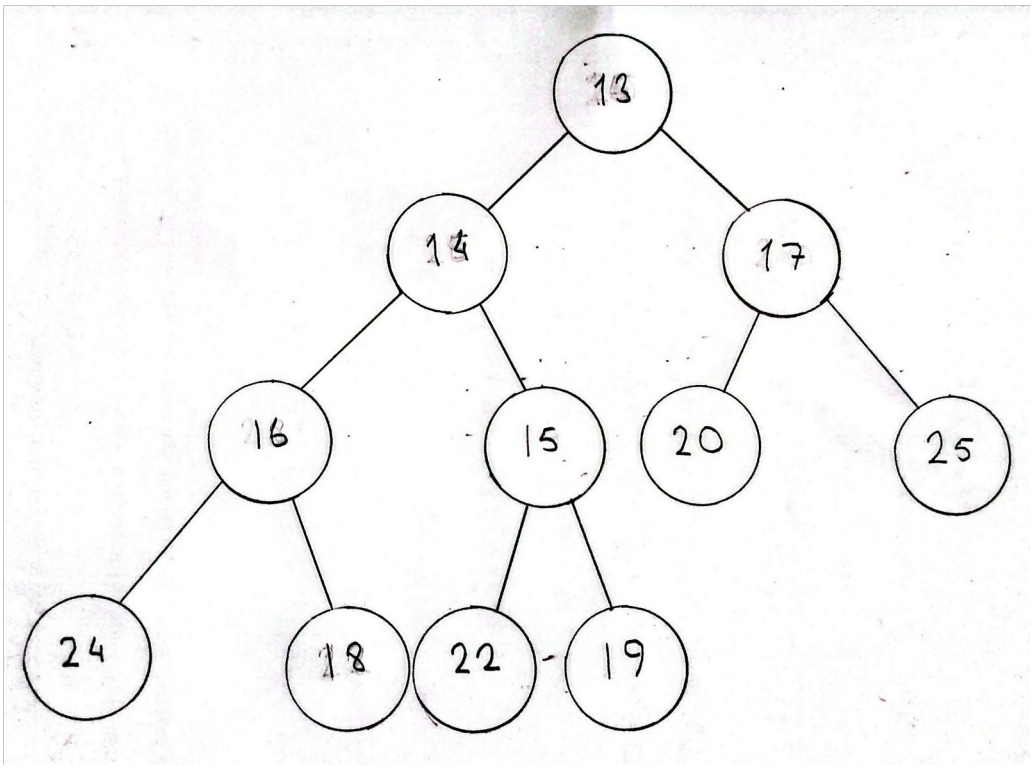
Insert 18:



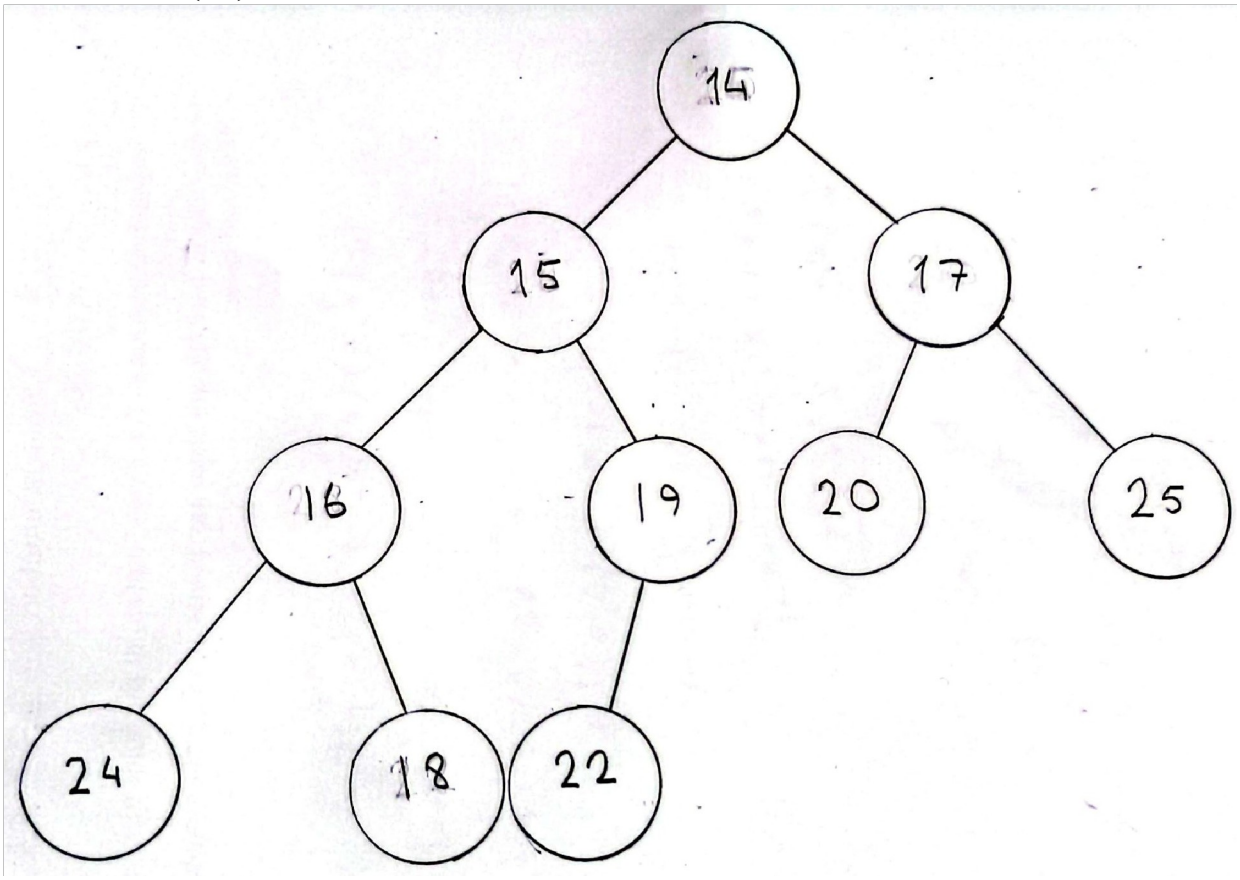
Insert 22:



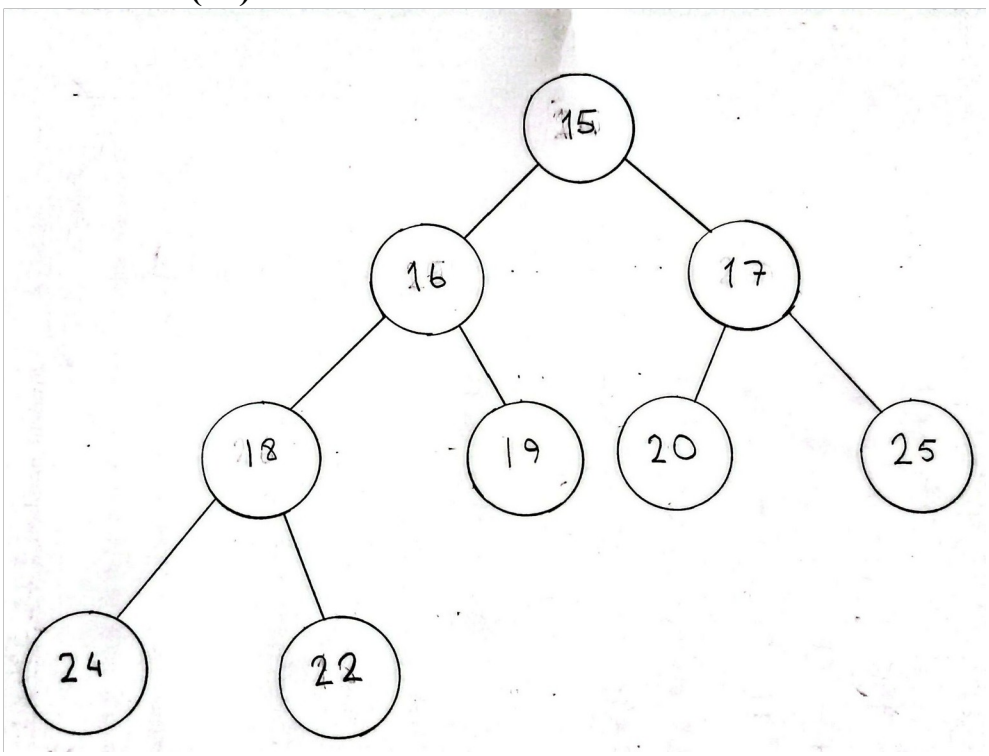
Insert 13:



Delete Min (13):



Delete Min (14):



Part B

Let's consider the following example:

- Heap 1 is created by inserting 1,2,3.
- Heap 2 is created by inserting 1,3,2.

Both of these heaps have 1 as their root. However, 2 is left and 3 is right children in heap 1 and the other way around in heap 2.

1. Heap 1
   - Preorder Traversal: 1 – 2 – 3
   - Inorder Traversal: 2 – 1 – 3
   - Postorder Traversal: 2 – 3 – 1
2. Heap 2
   - Preorder Traversal: 1 – 3 – 2
   - Inorder Traversal: 3 – 1 – 2
   - Postorder Traversal: 3 – 2 – 1

As it can be seen from this example, preorder traversal of a min heap may or may not be sorted since relationship between children of a heap node is not restricted. (One children can be greater, or smaller than the other.)

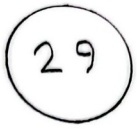
In addition, inorder traversal of a min heap whose size is greater than or equal to 3 can never be sorted, since minimum element will always be in the middle.

Finally, postorder traversal of a min heap may be sorted in descending order or may not be sorted at all.

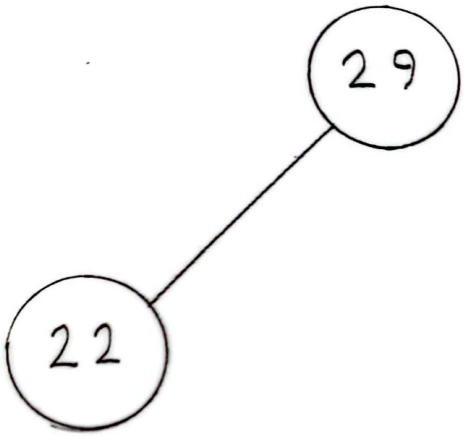
Even though sometimes preorder and postorder traversals may give heap elements sorted, it is less likely to get a sorted sequence with larger heaps since children are not restricted in terms of each other in heap data structure.

Part C

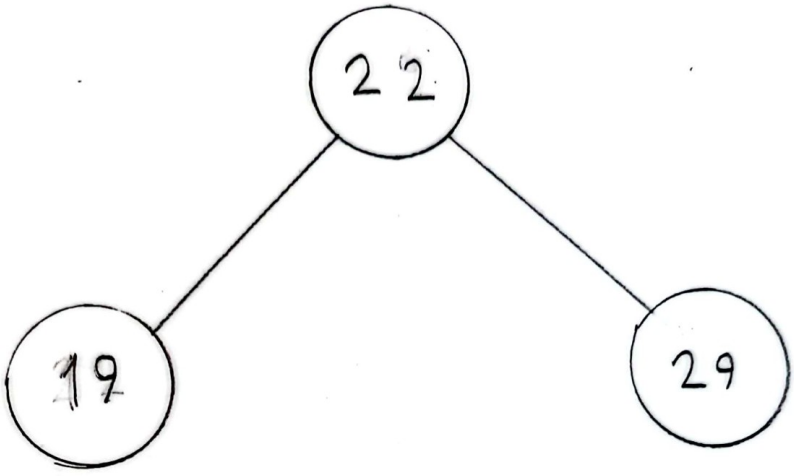
Insert 29



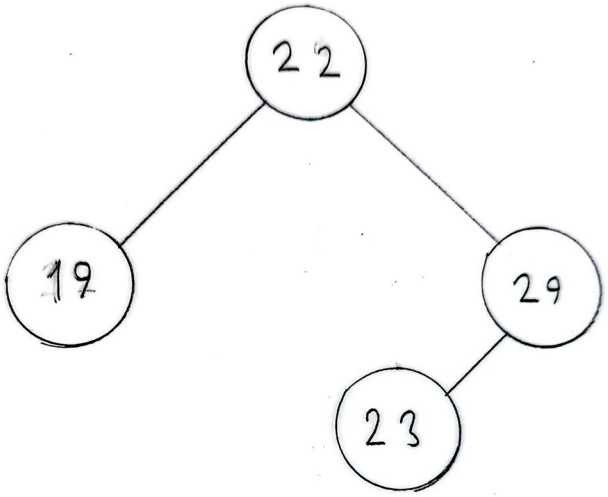
Insert 22



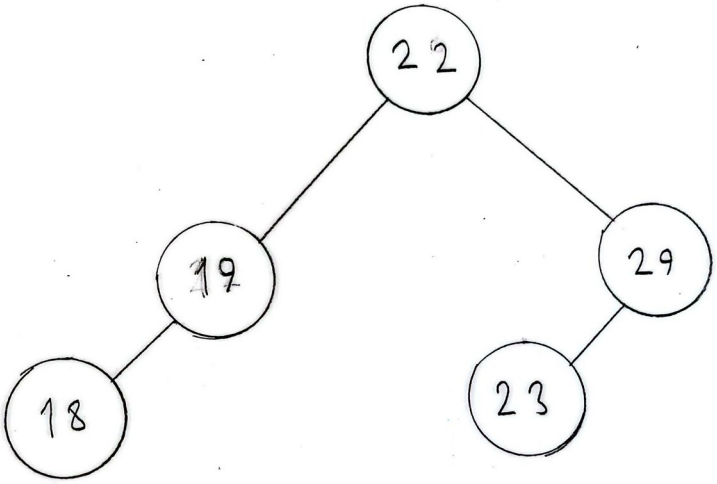
Insert 19



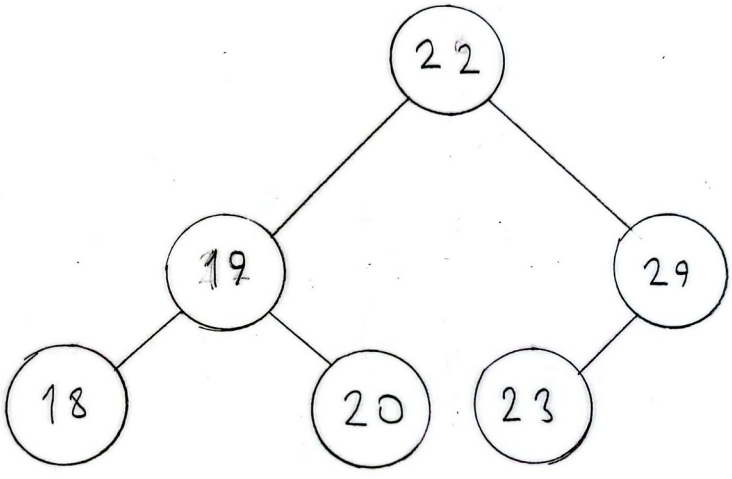
Insert 23



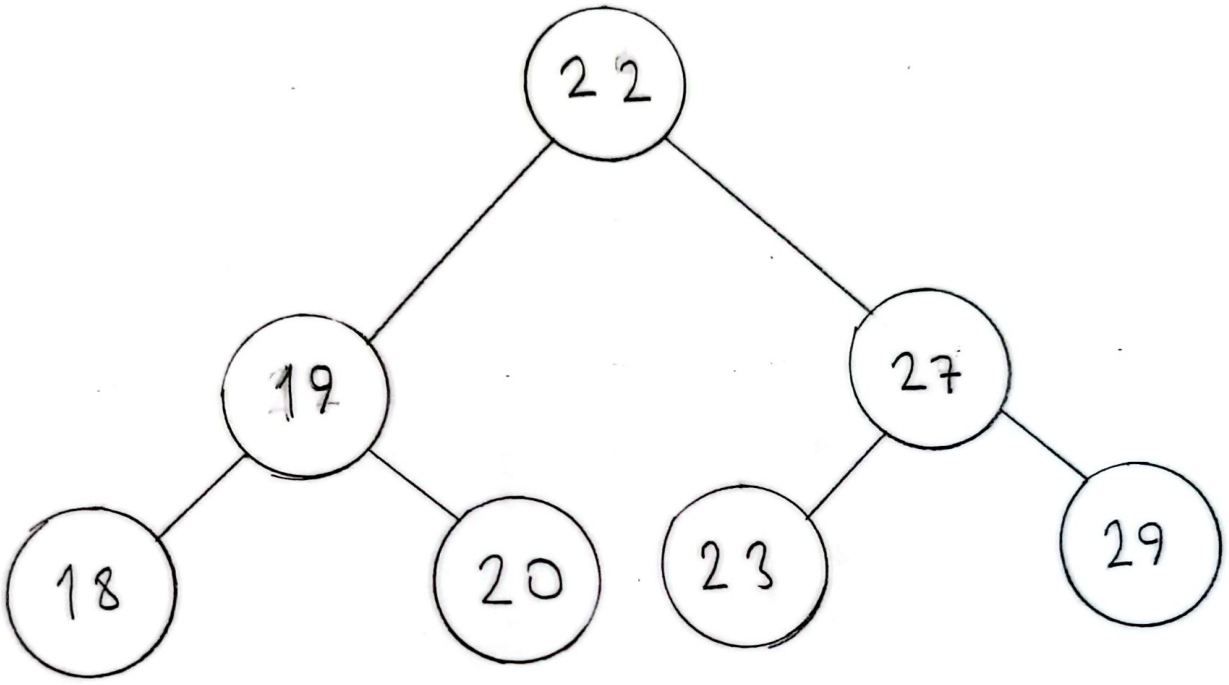
Insert 18



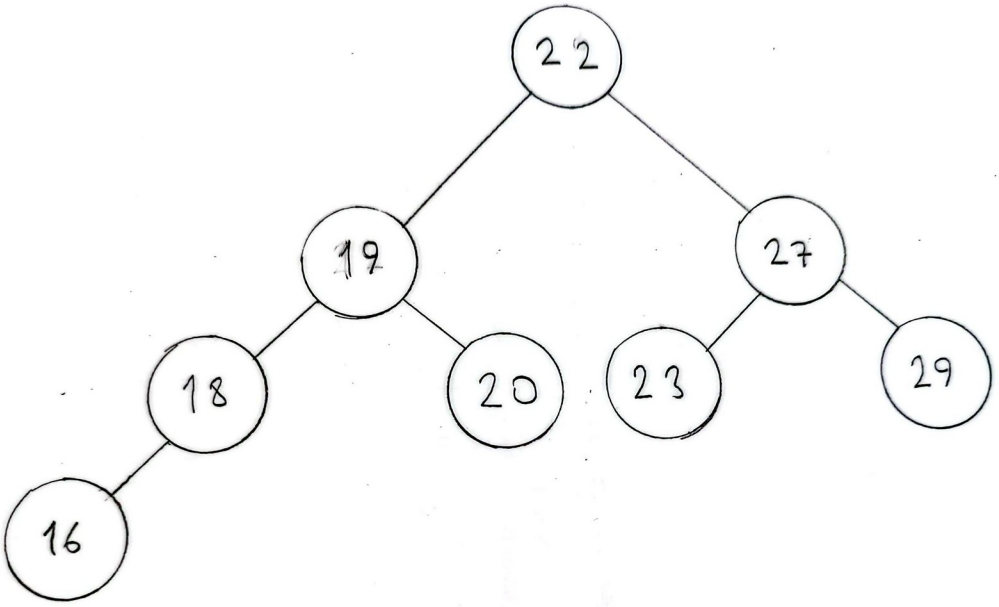
Insert 20



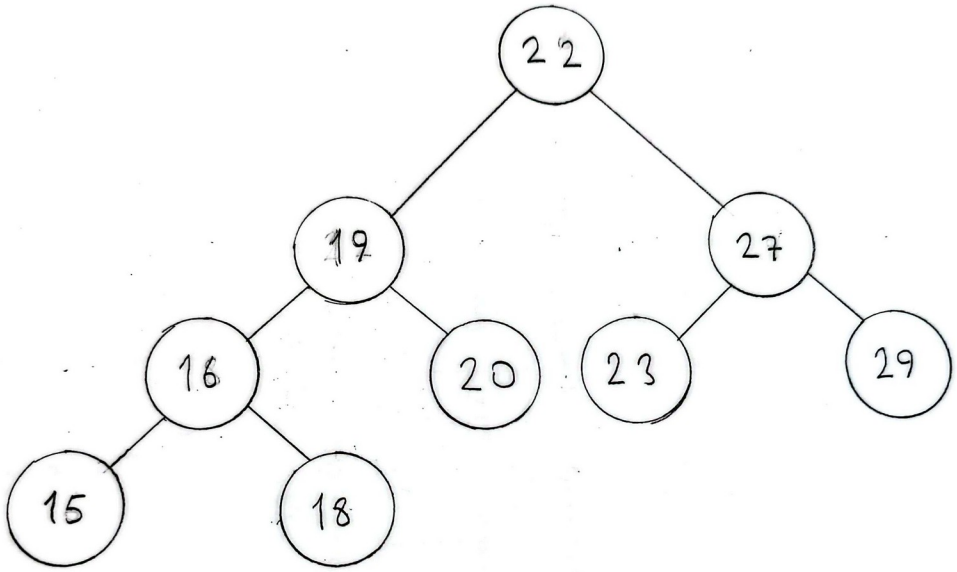
Insert 27



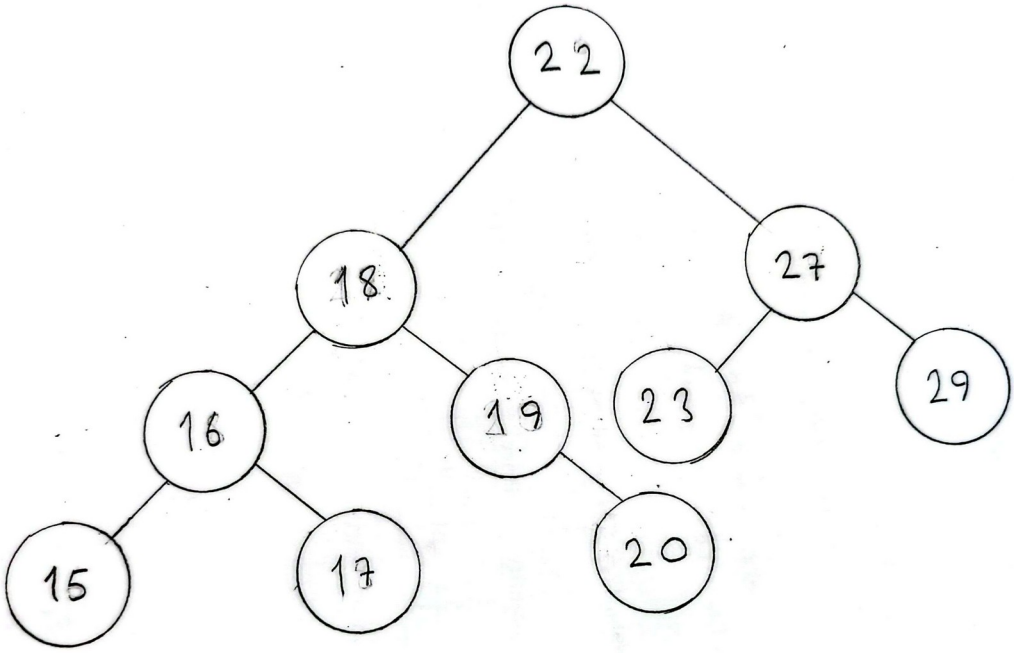
Insert 16



Insert 15



Insert 17



Question 3 Answer

For large systems, deciding on optimal printer number K could be done like searching a key in a sorted array with binary search.

Let's say the potential number of printers is 1000 (N = 1000). We should run the simulation with K = 500. If the resulting average waiting time is greater than acceptable average waiting time, then we should run the simulation with K = 750. However, if we have an acceptable time, we should run the simulation with K = 250 in order to optimize the printer count. Repeatedly doing these steps would find the optimal printer count more efficienly than doing it with a linear search manner.