CS 202 - Fundamental Structures of Computer Science 2

Homework 3 – Heaps, Priority Queues, and AVL Trees

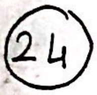
Student Name: Esad İsmail Tök

Student ID: 21801679

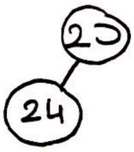
Section: 2

Question 1 Part a)

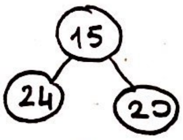
Insert 24 :



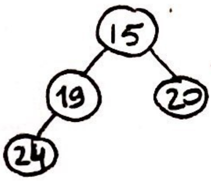
Insert 20 :



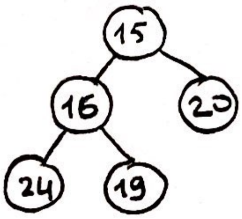
Insert 15 :



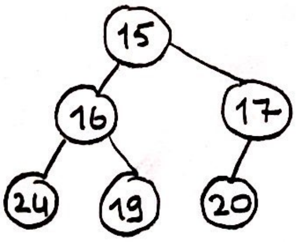
Insert 19:



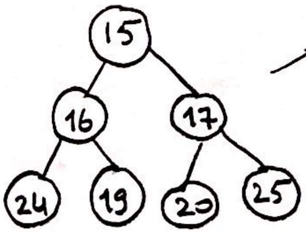
Insert 16 :



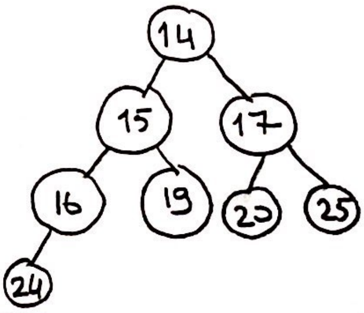
Insert 17:



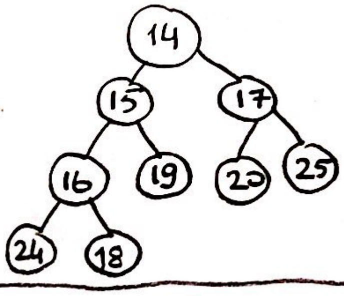
Insert 25:



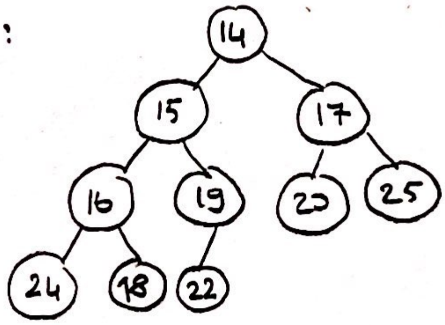
Insert 14:



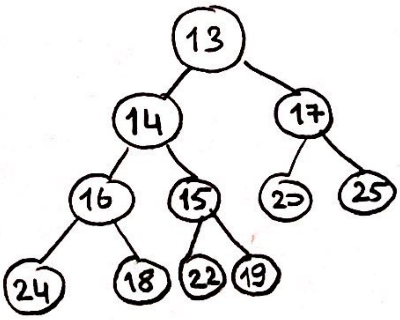
Insert 18:



Insert 22:

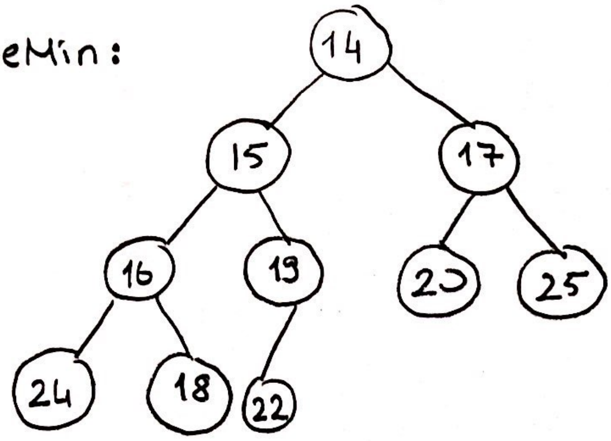


Insert 13:

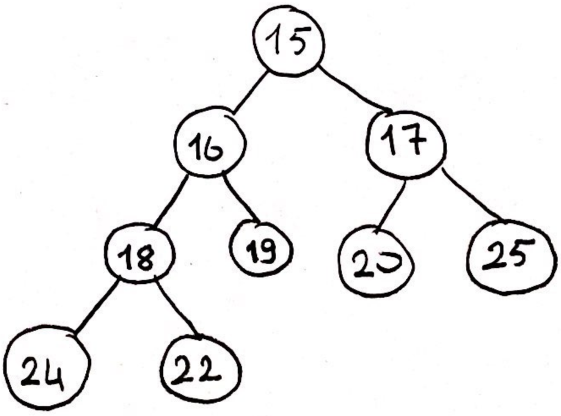


Question 1 Part a) Continue

DeleteMin:

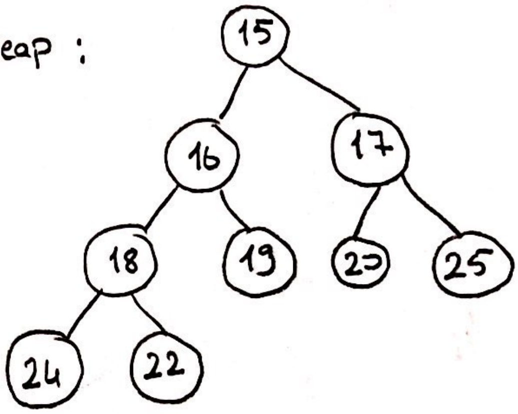


DeleteMin:



Question 1 Part b)

Consider the binary Min-Heap :



I) The preorder travers of this heap is :

[15, 16, 18, 24, 22, 19, 17, 20, 25]. As it can be seen, the preorder traverse of the binary min-heap is not sorted. Even though the parent node is smaller than it's child nodes in a binary min-heap, there is no relation between the child's. Therefore this traverse is not sorted.

II) The inorder traverse of this heap is:

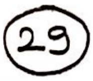
[24, 18, 22, 16, 19, 15, 20, 17, 25]. As it can be seen, the inorder traverse of the binary min-heap is not sorted just like the preorder traverse. It is again because the min-heap has parent nodes which are smaller than its' child nodes but it is solely not enough for the min-heap to be sorted-

III) The postorder traverse of this heap is:

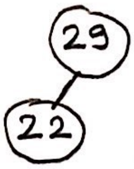
[24, 22, 18, 19, 16, 20, 25, 17, 15]. As it can be seen, the postorder traverse of the min-heap is again not sorted just like the preorder and inorder traverses. The reason is again, a min-heap does not have a strict construction as BST, but it has only the condition that a parent is smaller than it's children. And that is not enough for it to be sorted in any of the traversals.

Question 1 Part c)

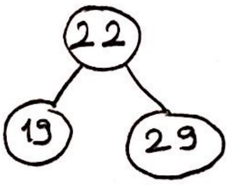
Insert 29 :



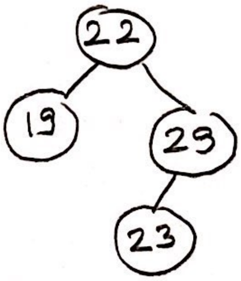
Insert 22 :



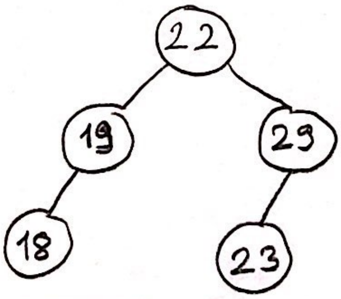
Insert 19 :



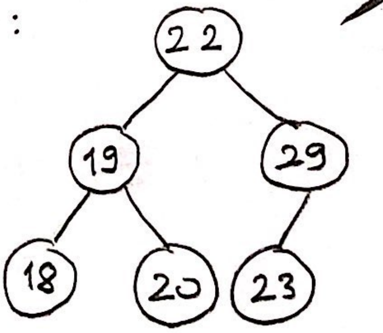
Insert 23 :



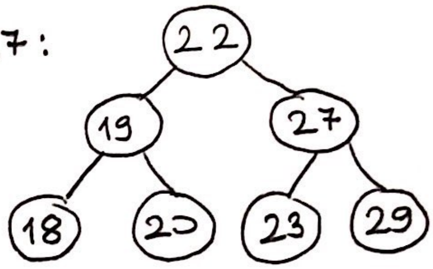
Insert 18 :



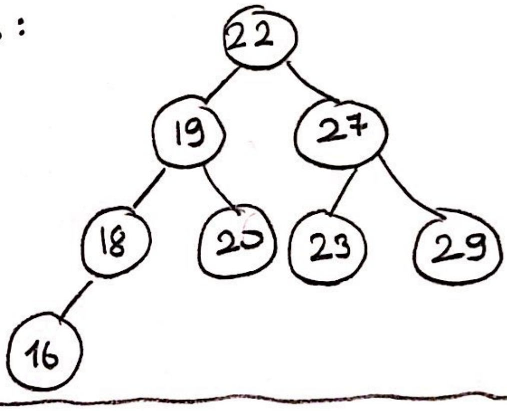
Insert 20 :



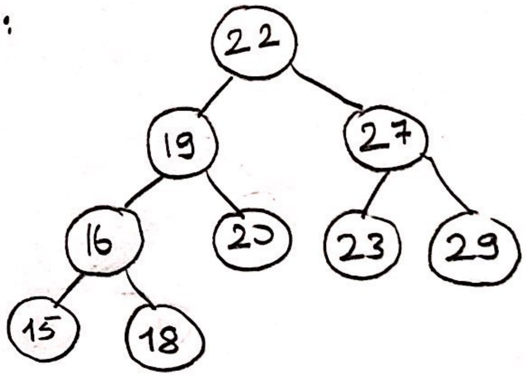
Insert 27 :



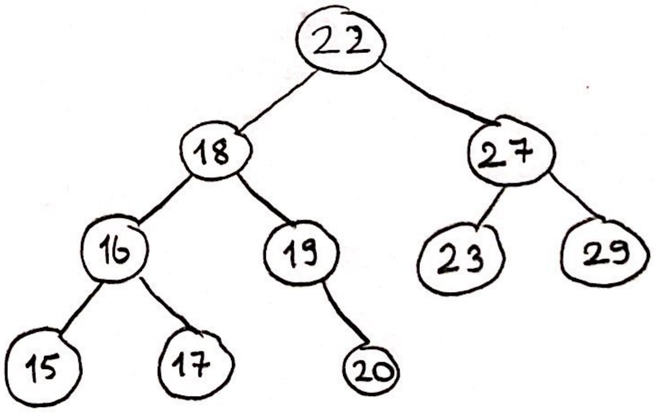
Insert 16 :



Insert 15 :



Insert 17 :



Question 3

- For small number of printers and print requests, starting from one printer and increasing the number of printers to find the correct number of printers was a reasonable strategy. However applying the same strategy for large numbers of printers and print requests would spend huge amount of resource, especially time, because in this scenario, we would work on the same request file with many requests over and over each time by increasing the printer count only one by one.

- Therefore if we consider the potential total number of printers as *"N"* and the number of printers that we need, as *"K"* according to the prompt of *Question 3,* it would be a bad idea to make the simulation starting with 1 printer and incrementing it one by one. We need to find a better and more efficient way to implement the simulation for the large number of printers and requests.

- A possible better strategy can be applied by using an algorithm that is similar to the binary search logic. Since we know that the available number of printers that we have is *"N",* we can initiate the simulation by using *"N/2"* printers and according to the average waiting time that we get from that simulation with "*N/2"* printers*,* we can decide the new number of printers needed in the next simulation. For example if we get an average waiting time that exceeds the limit, we can then pick the new printer count as the new middle value which is ((N/2 + N) / 2), discarding the lower half of the possible printer numbers. Therefore we do not have to increase the number of printers one by one anymore. In such a strategy, a possible drawback is that we may choose a number of printers that allow us to get an average waiting time but this number of printers may not necessarily be the minimum number since we might miss other possible printer counts in between. To prevent this scenario, after finding the first correct number of printers that gives us a waiting time below the limit, we can start checking the opposite way to find the first incorrect number of printers that gives us the waiting time above the limit. And we can proceed this strategy until we find the minimum number of printers, namely *"K"* that gives us a waiting time that is below the limit as we want.

- Thus, by applying this new strategy it is possible to find the correct number of printers, namely *"K",* in a logarithmic time according to the potential number of printers, namely *"N".* Therefore it is a better strategy than just incrementing the printer count one by one, for large number of printers and print requests.