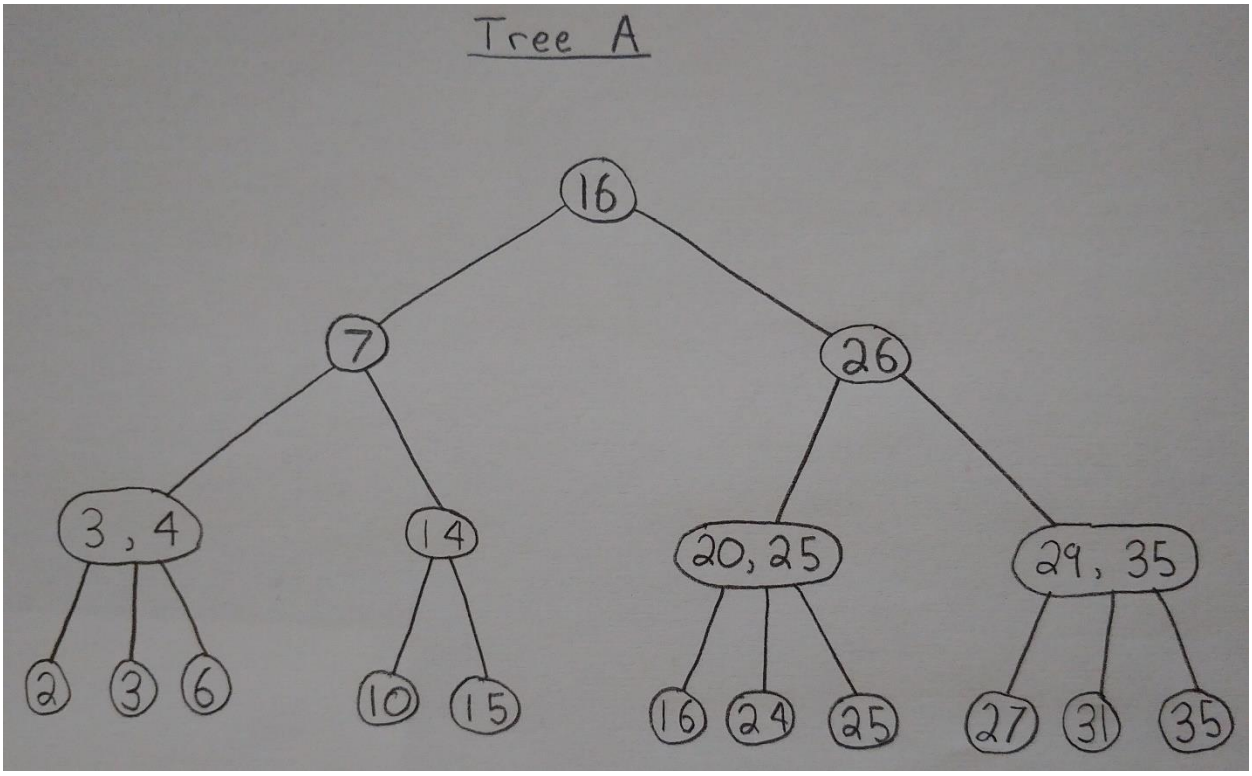
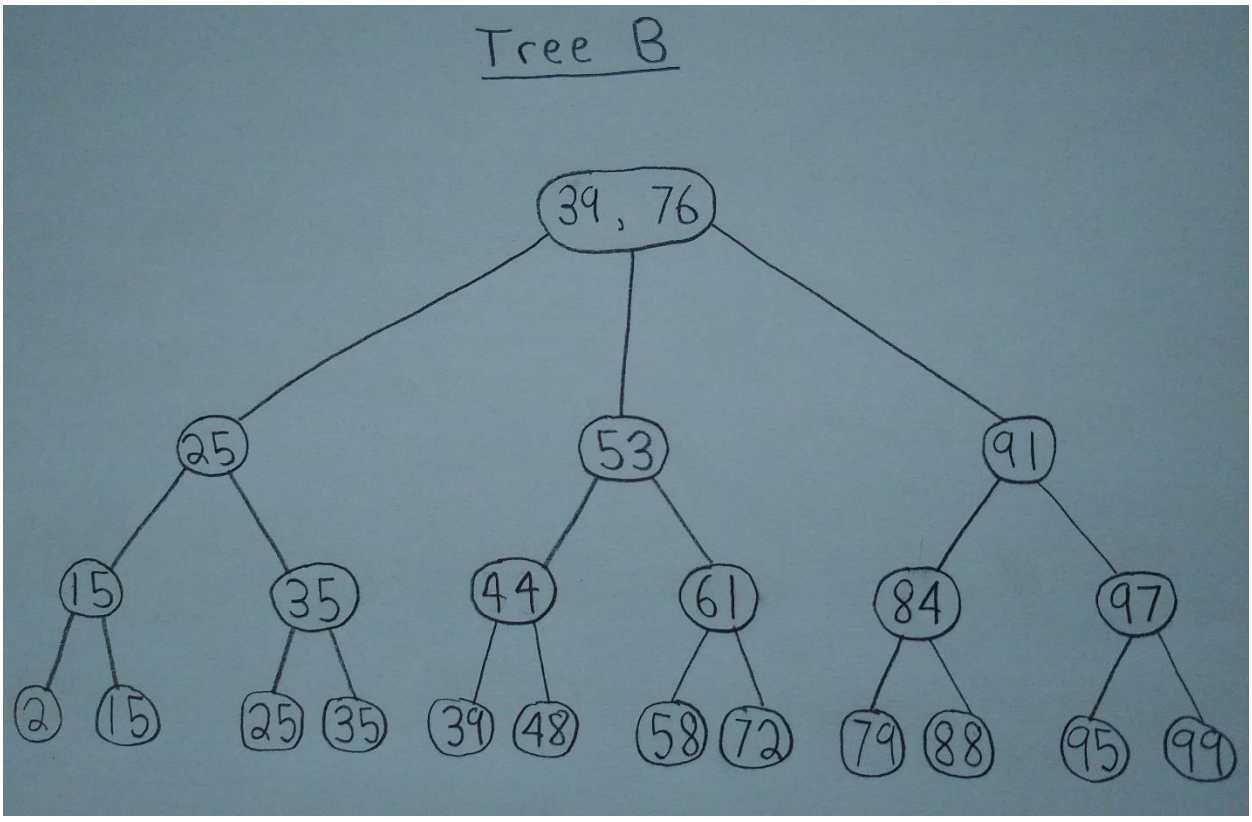
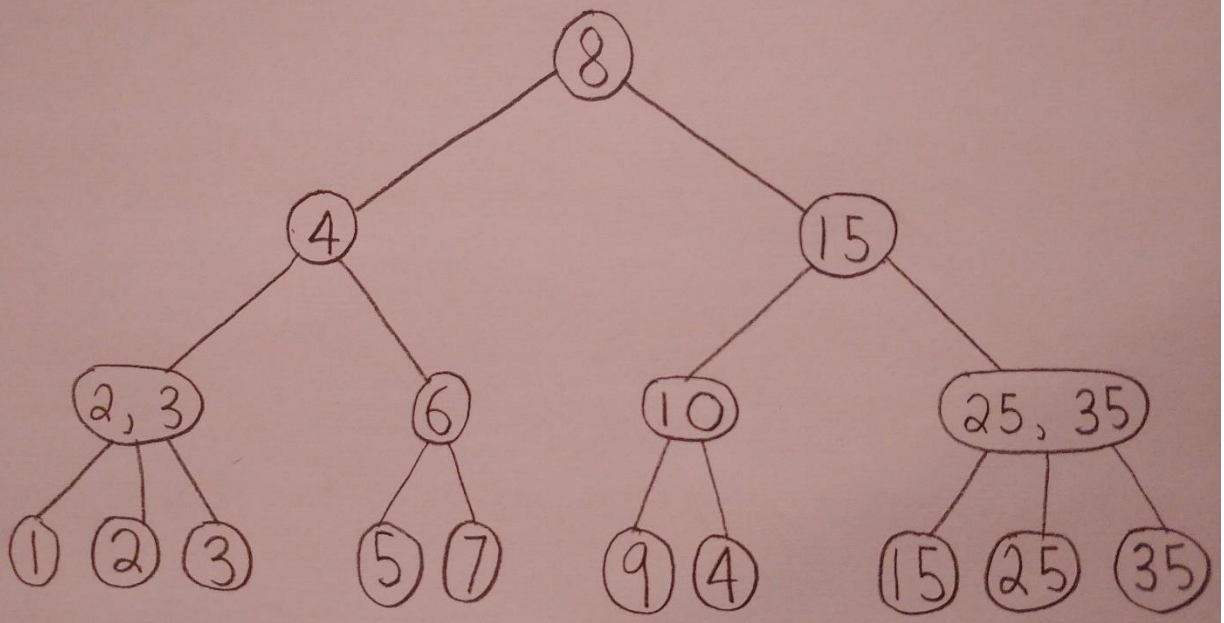
Part 2: Leaf-Based 2-3 Trees

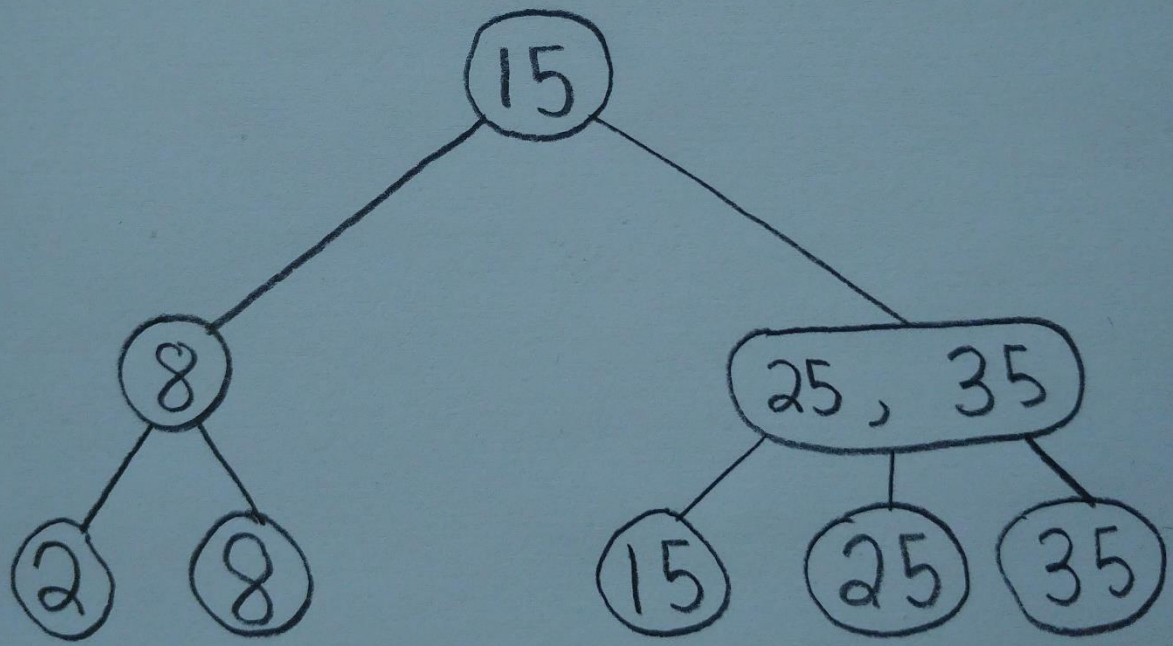




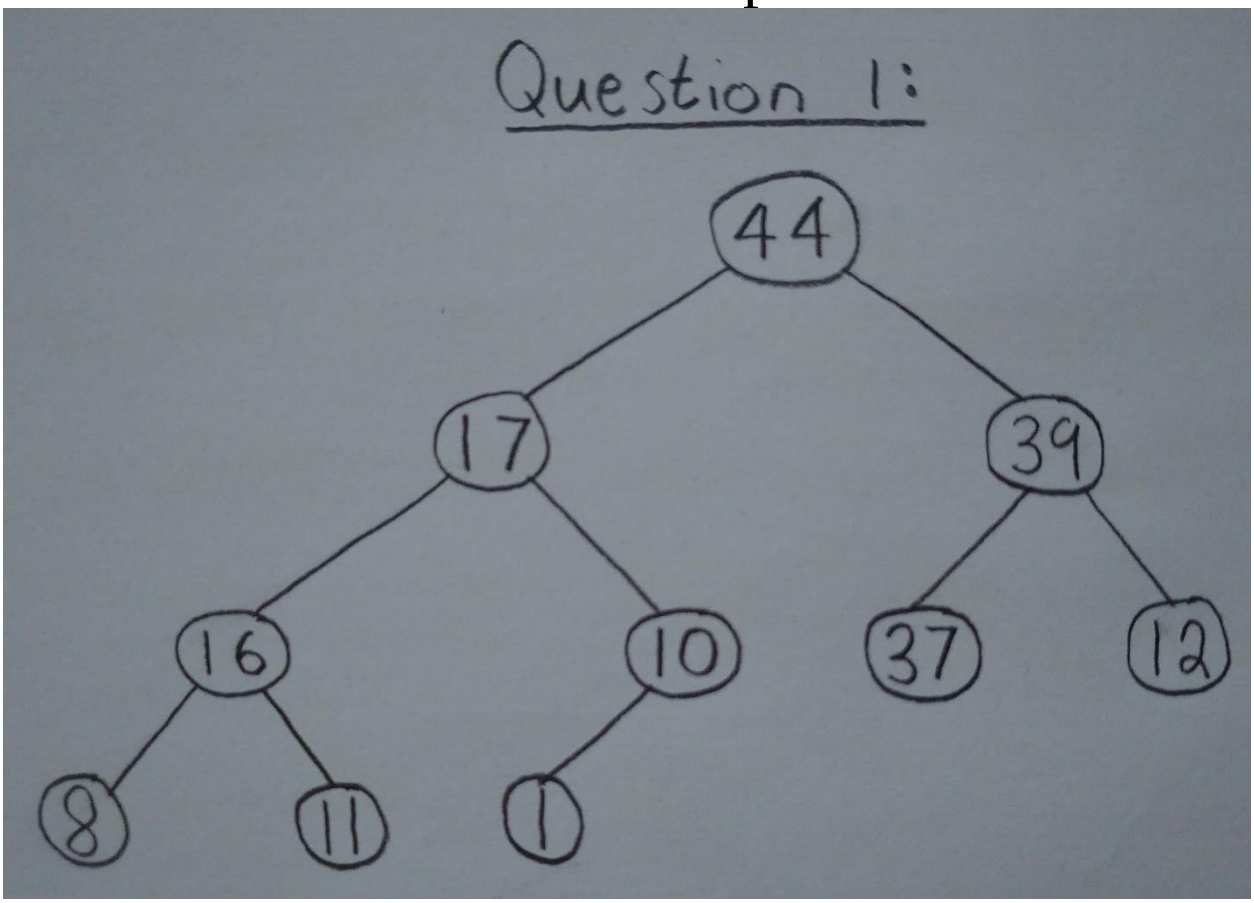
Tree C

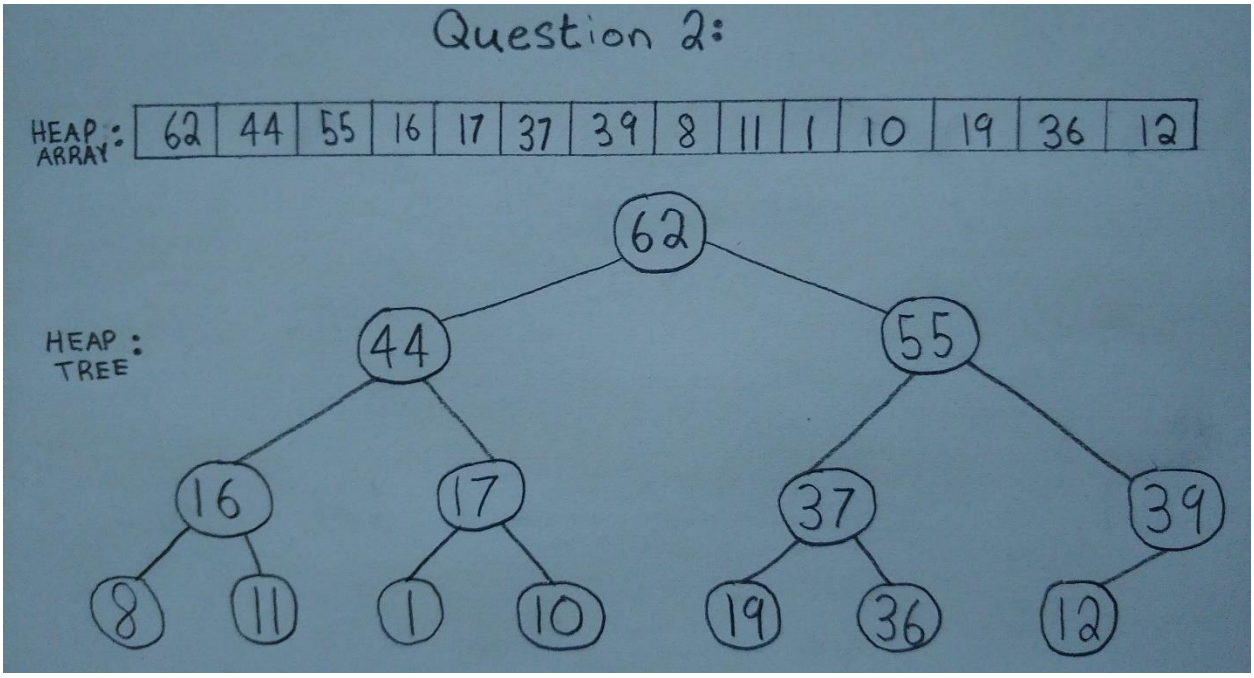


Tree D



Part 2: Heaps


Question 1:
44
17
39
16
10
37
12
8
11
1


Question 2:
HEAP ARRAY:
62 44 55 16 17 37 39 8 11 1 10 19 36 12
HEAP TREE:
62
44
55
16
17
37
39
8
11
1
10
19
36
12

Question 3

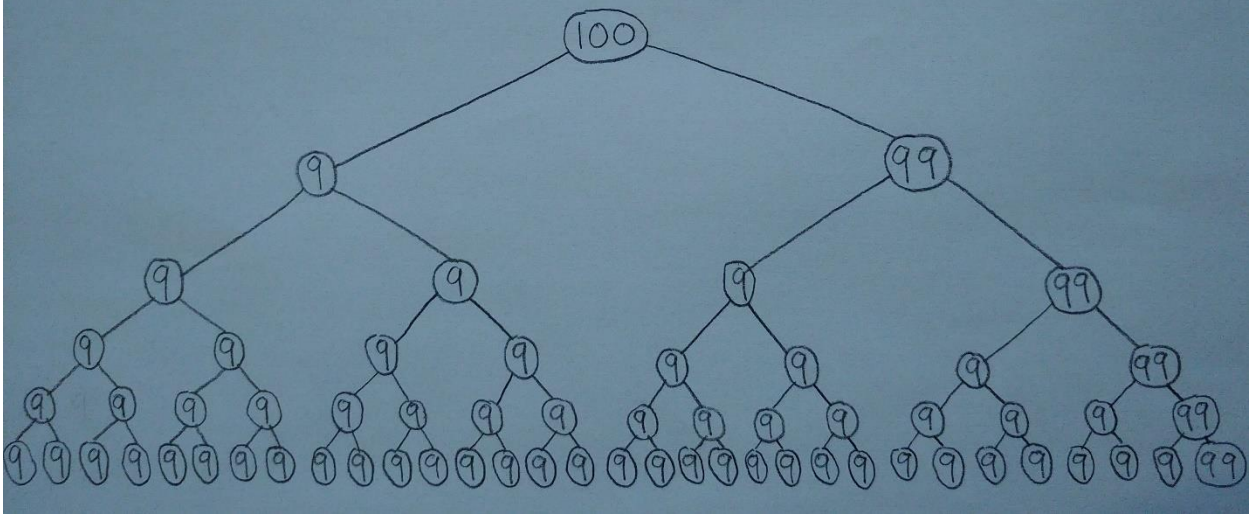
Heap array after one deleteMax() operation:

18	12	17	9	3	16	2

Heap array after two deleteMax() operations:

17	12	16	9	3	2

Question 4:



Question 4:

Statement: “The deleteMax() operation in a heap of size n, where n is a large number, can take O(1) in the best case.”

I will prove that this statement is true.
If we take the above heap of size n=63, and perform a single deleteMax() operation, the root node containing priority 100 will be replaced with the leaf node containing priority 99. The new root node does not have a child that is greater than it, so we cannot sift it down. This heap requires only one swap to put the leaf into the root position and heap order is restored. Performing a single deleteMax() operation on this heap runs at O(1) time. This scenario of having the right-most leaf be the same value as all of its ancestors up to the root, and the other child of the root being smaller than it, will make deleteMax() run at O(1) time for any large heap size n.