David Xie

40065595

November 15, 2020

COMP352 – Assignment 3 Written Questions

Question 1

a)

Algorithm:

Do depth-first search on tree T with another parameter to record depth

Algorithm input the tree, node, and depth.

Algorithm output collection of depth values for each nodes.

- Input the tree and the start node. Initialize depth parameter to 1.
- Add the depth value to the collection of depth values.
- In the execution of depth-first search, each recursive call increments the depth parameter by 1.
- At the end, should obtain the collection of depth values for each nodes.

Time complexity: O(n + m) time where n is the number of nodes and m is the number of tree branches/edges.

Space complexity: O(n), since at most n space will be allocated for each collection.

b)

Do depth-first search on binary tree T with a variable to record the count for full nodes (nbFullNodes).

-Initialize nbFullNodes to 0.

-Each depth-first search recursive call checks if the node has both a left child and a right child. If that is the case, increment nbFullNodes by 1.

-Return nbFullNodes.

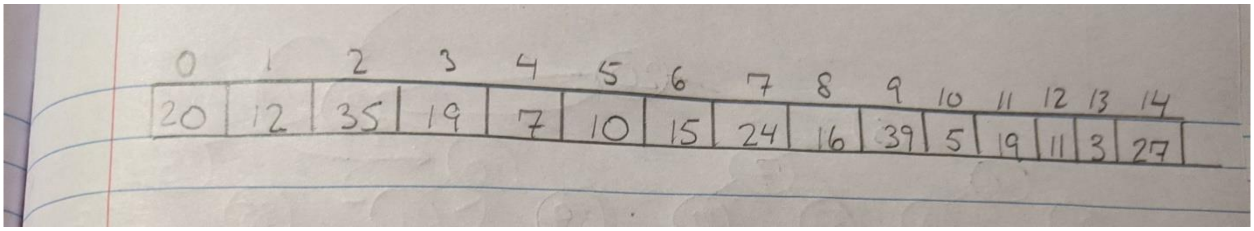
Time complexity: O(n + m) time where n is the number of nodes and m is the number of tree branches/edges.

Space complexity: O(n) since recursion acts similarly to stacks and occurs n times.

Question 2

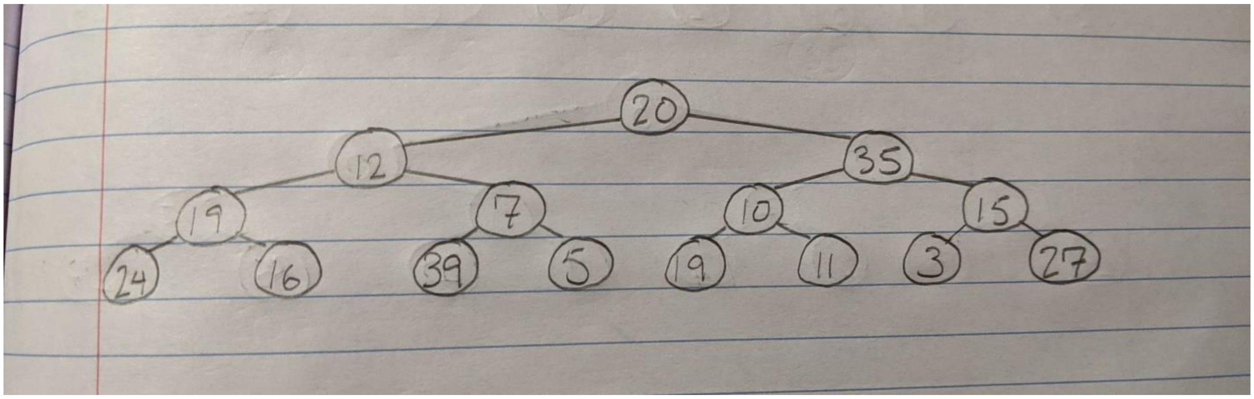
a)

Assuming:

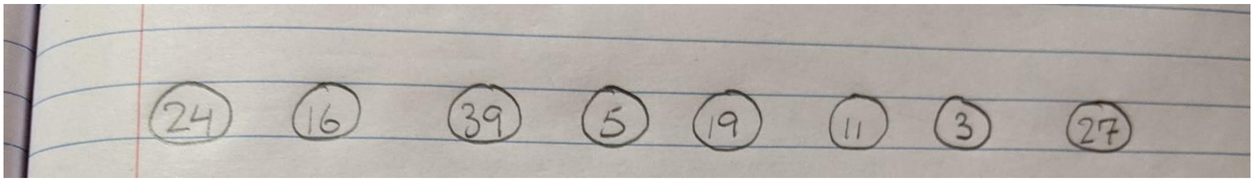


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	12	35	19	7	10	15	24	16	39	5	19	11	3	27

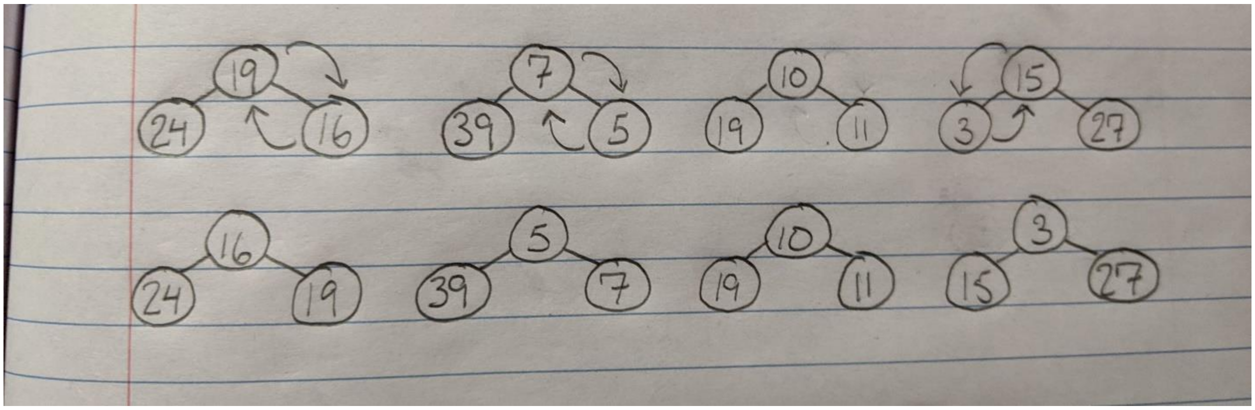
Then the corresponding tree is:



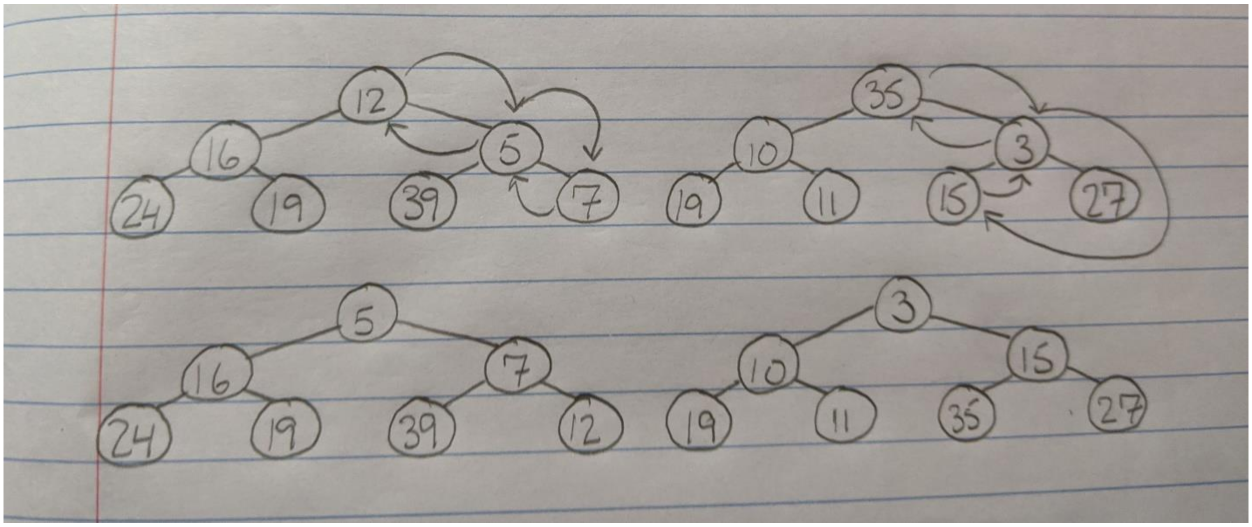
Doing bottom-up construction, start with 1-entry heaps on bottom level:



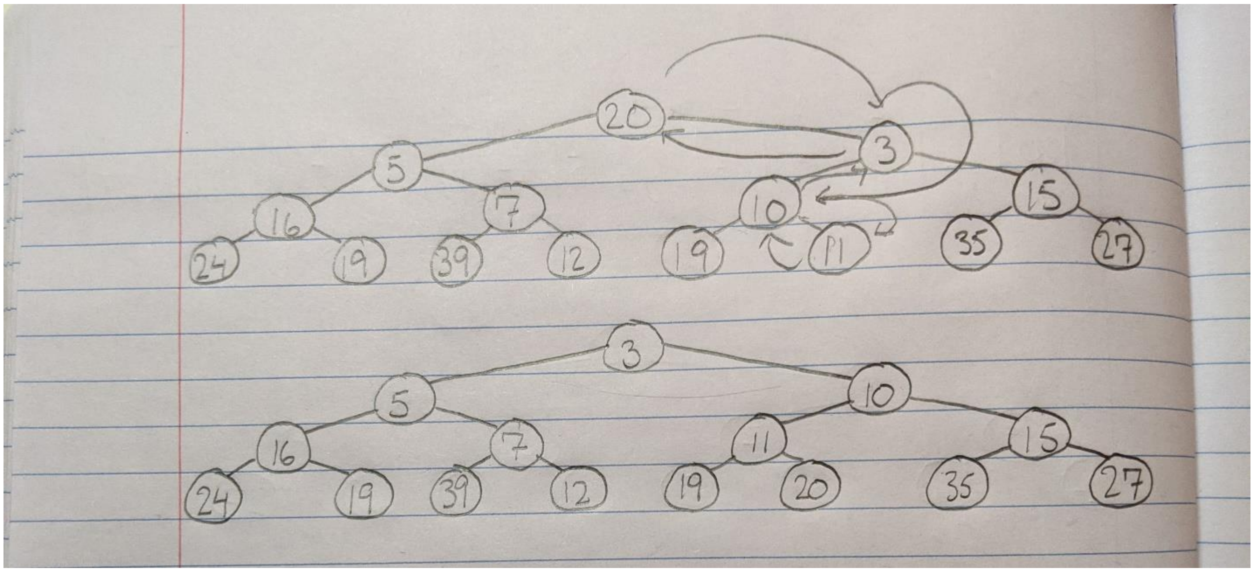
Combine into 3-entry heap and downheap:



Combine into 7-entry heap and downheap:

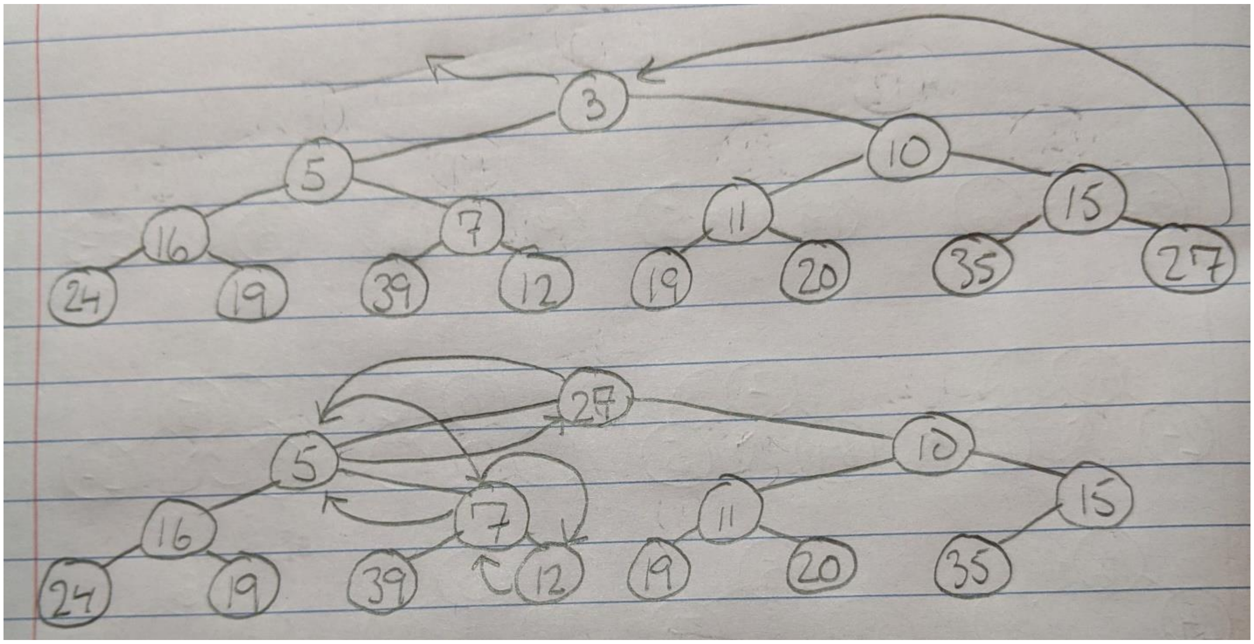


Final heap and downheap:

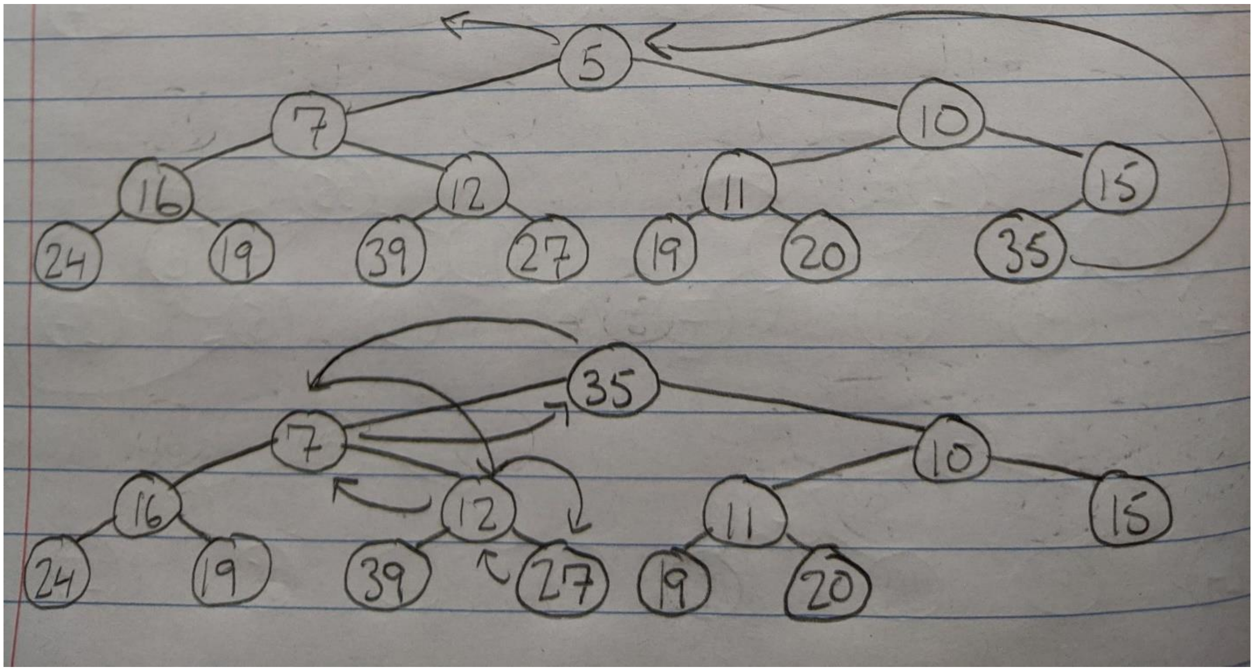


Next, perform 6 removeMin

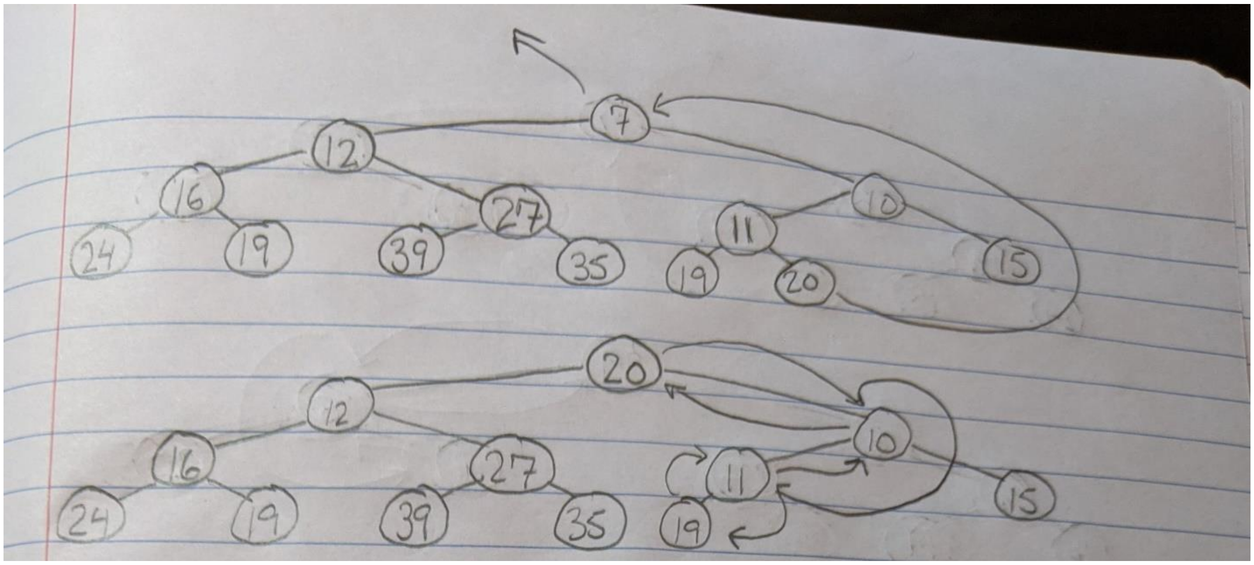
1st removeMin:



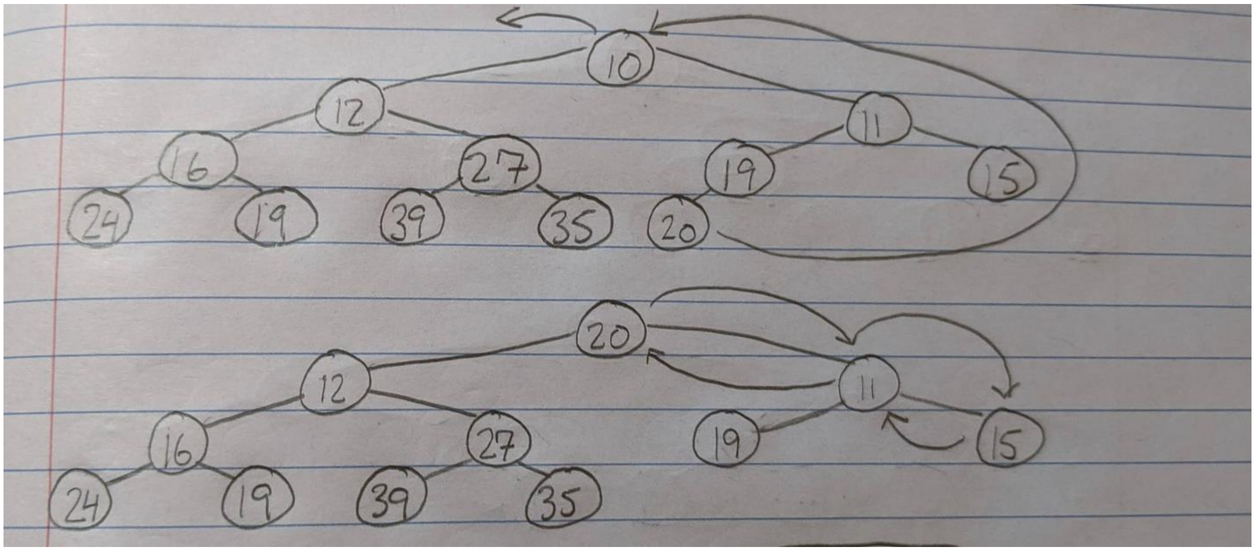
2nd removeMin:



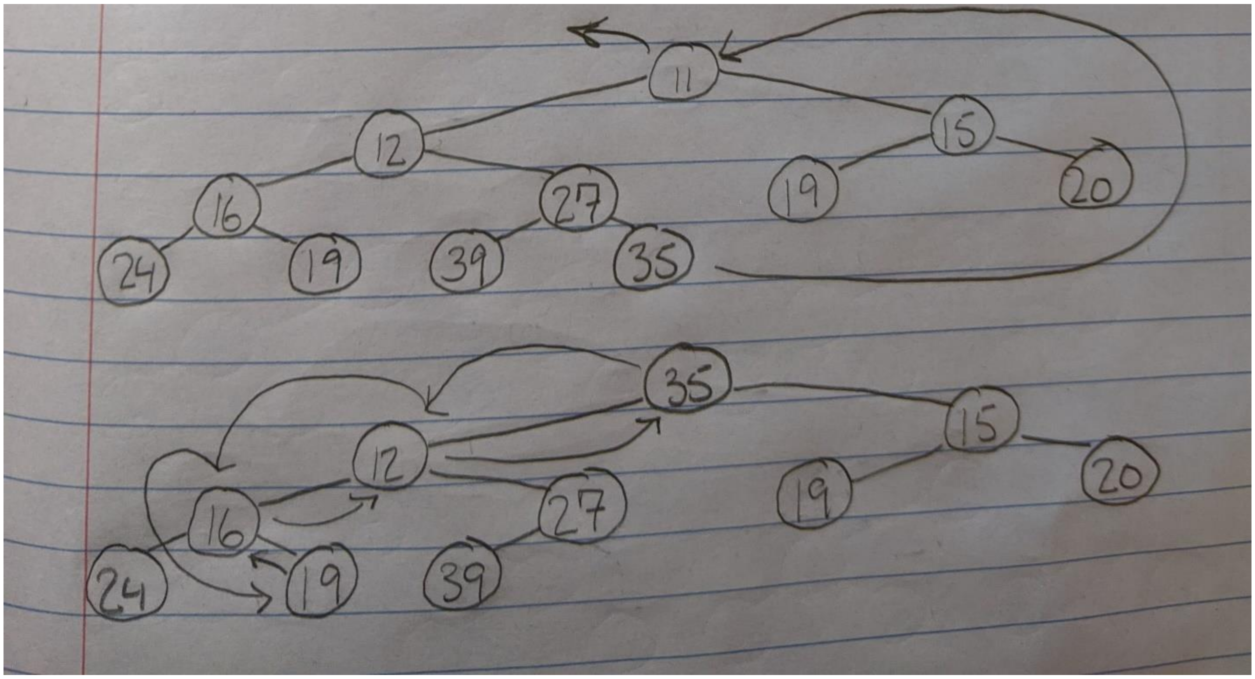
3rd removeMin:



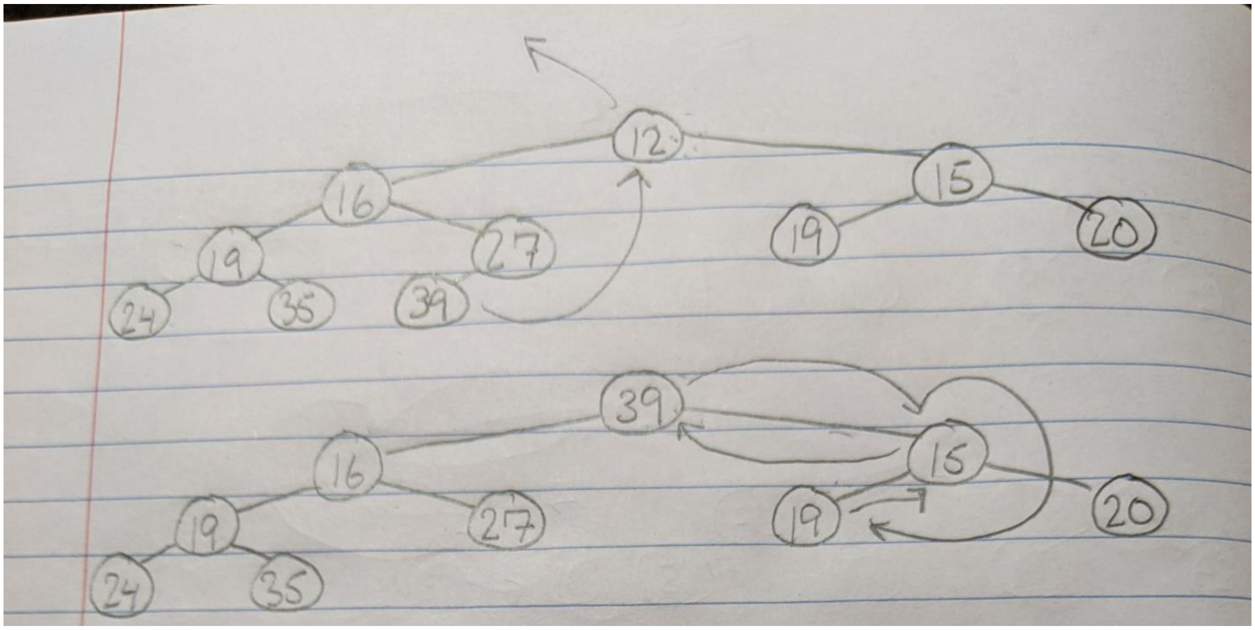
4th removeMin:



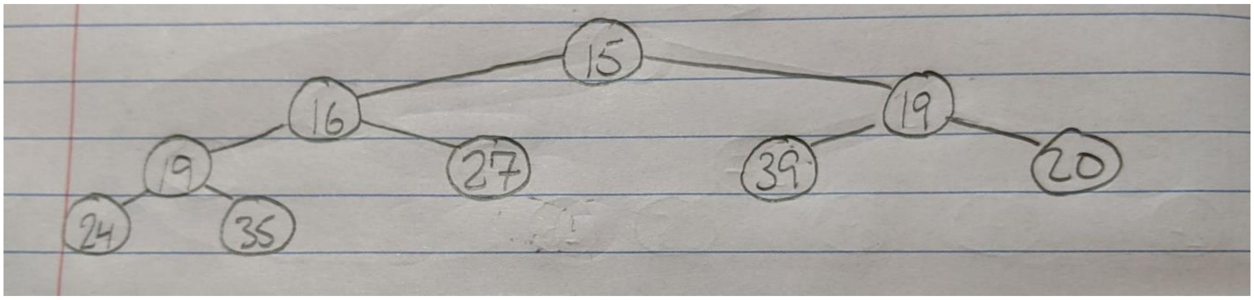
5th removeMin:



6th removeMin:

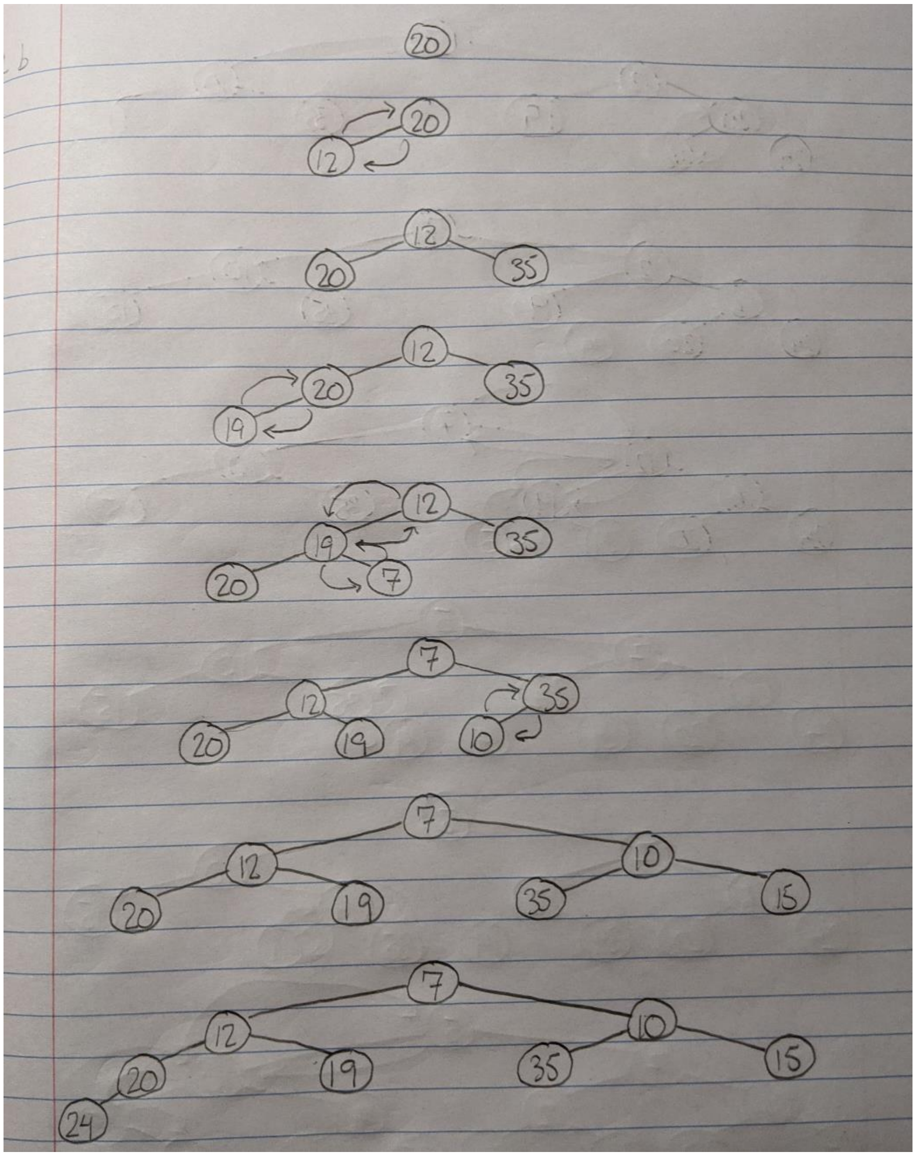


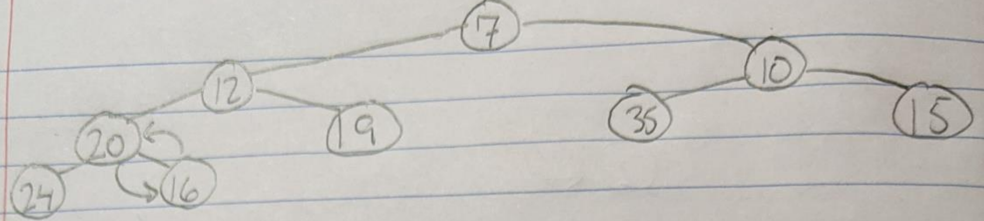
Final tree:

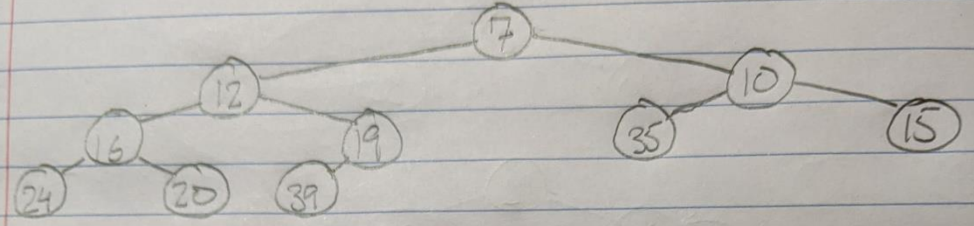


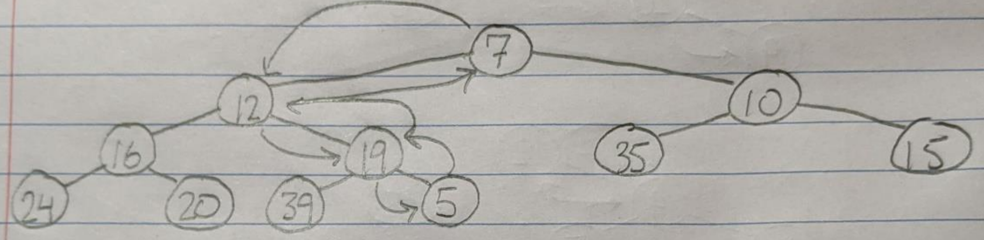
b)

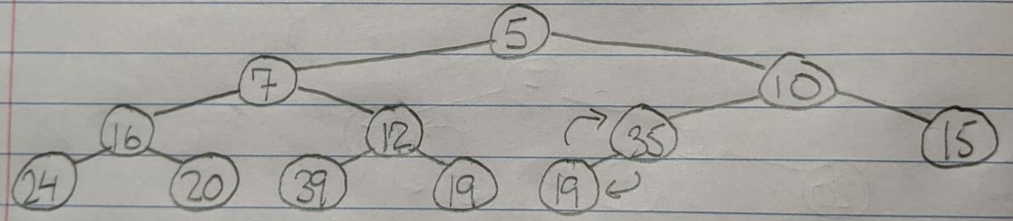
Create min-heap using insertion.

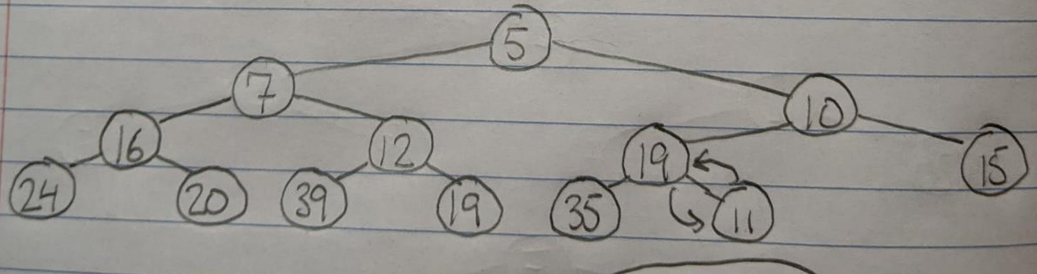


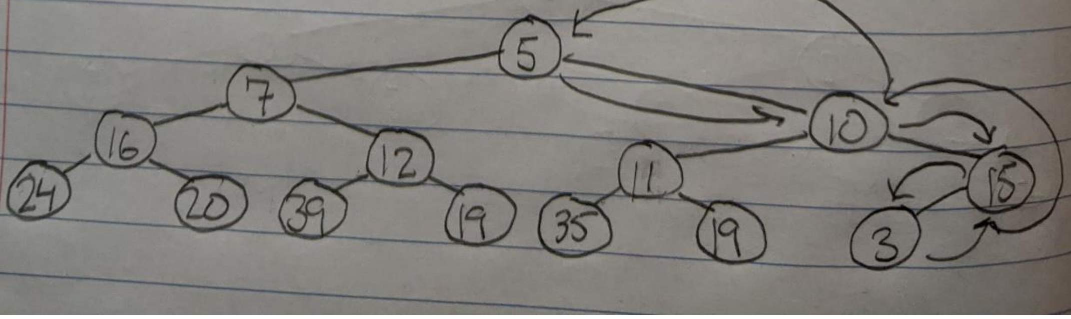

7
12
10
20
19
35
15
24
16


7
12
10
16
19
35
15
24
20
39

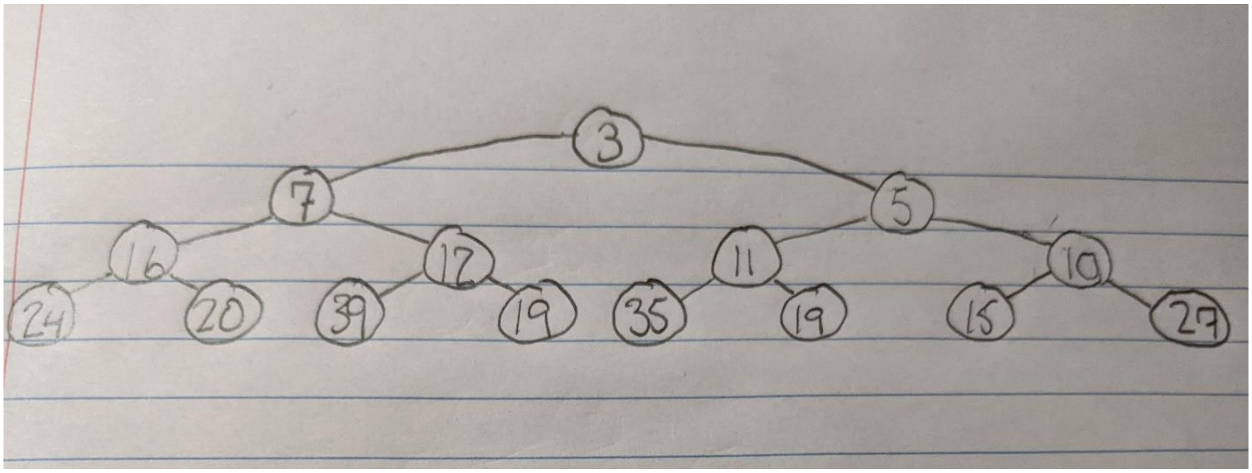

7
12
10
16
19
35
15
24
20
39
5


5
7
10
16
12
35
15
24
20
39
19
19


5
7
10
16
12
19
15
24
20
39
19
35
11


5
7
10
16
12
11
15
24
20
39
19
35
19
3

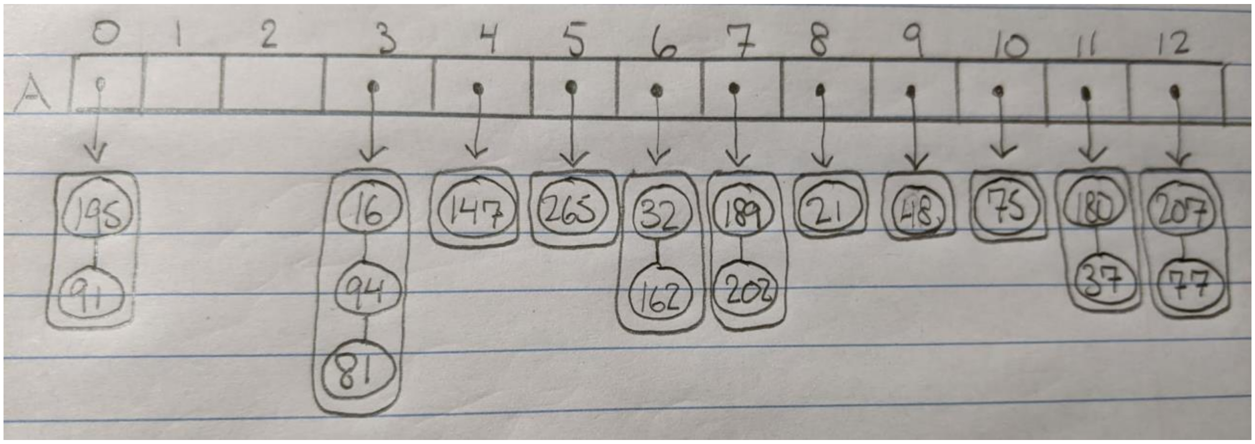
Final tree:



Question 3

i)

Do $h(k) = k \, mod \, 13$ and use that value to insert the key into the correct bucket.



ii)

Based on the above representation, there is a maximum of 7 collisions, as a same h occurs 7 times.

Question 4

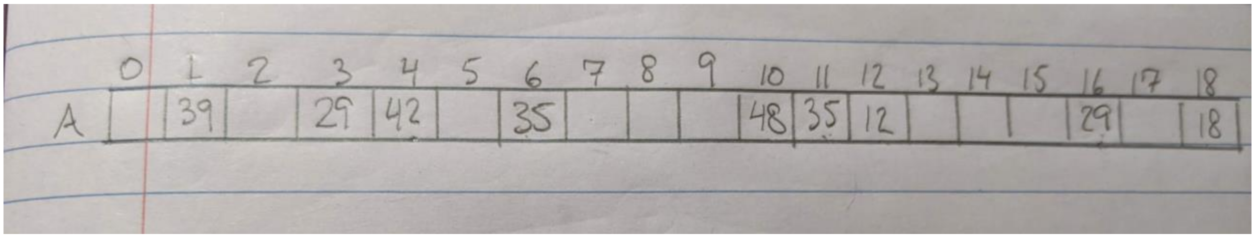
At first glance, the proposal seems to hold some validity. As it explains, for separate chaining, it is preferable for the load factor, $\lambda = n/N$, be less than 1. Increasing N from 13 to 15 does reduce the load factor, thereby supposedly reducing the risk of collisions.

However, when put into practice, the resulting contents show a maximum of 8 collisions. The proposal is senseless because it suggests using N = 15, which is not a prime number, for its compression function. Using a non-prime number for N increases the risk of repeat hash values, resulting in collisions.

Question 5

i)

To find the correct position, do $\left(i + jd(k)\right)\, mod\ N$ with $i = h(k)$, $j = 0, 1, \dots, 18$ and $d(k) = 7 - k\ mod\ 7$. j increments for that key search every time there is a collision.



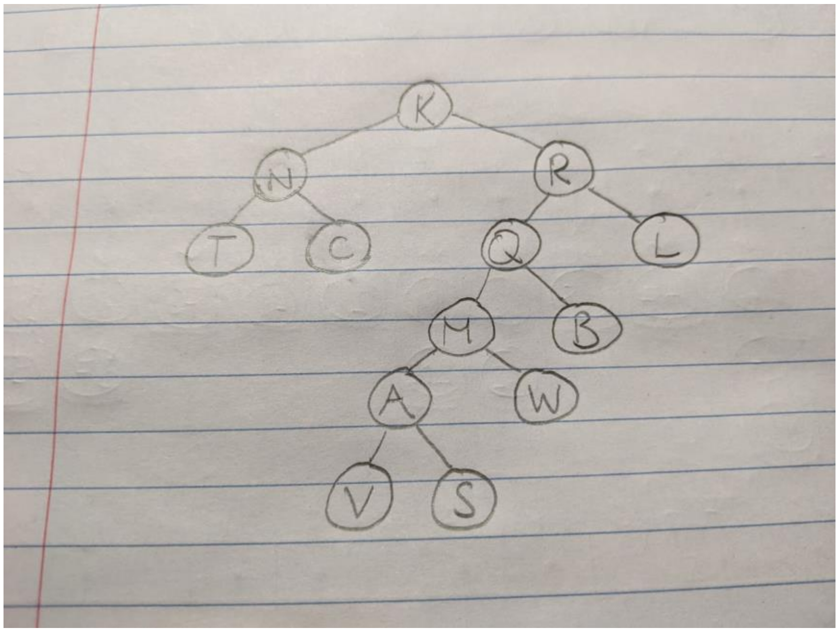
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A		39		29	42		35				48	35	12				29		18

ii) Longest cluster = 2.

iii) There were 11 collisions as a result of the operations.

iv) Load factor, $\lambda = n/N$, where n is the number of entries and N is the size of the array. Hence, $\lambda = \frac{9}{19} = 0.4737$

Question 6



K
N
R
T
C
Q
L
M
B
A
W
V
S