Mohammad Ali Zahir (40077619)

COMP 352: Data Structures & Algorithms

March 30th, 2020

Assignment 3

1. Category 1: We would have to use the **List ADT,** since we have to use all the methods, which are in this ADT, and it is easier for this to do the indices.

   Category 2: Since we have to add the container at a certain position, we would have to use the **Position ADT**. This allows us to access any position in the data type and put a container there.

   Category 3: The only one left would be the **Sequence ADT**, since it follows a certain order, no matter if we put the element before or after. So here, the container would automatically be sorted in alphabetical order once we put it.

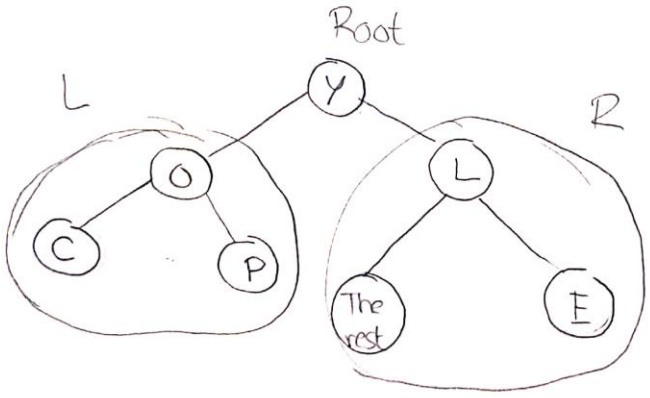
2)

a) Inorder: C O P Y R I G H T A B L E

Postorder: C P O R G I T H I B A E L Y

Remember Inorder: Left - Root - Right → starts with C - O - P

The most bottom left should be C

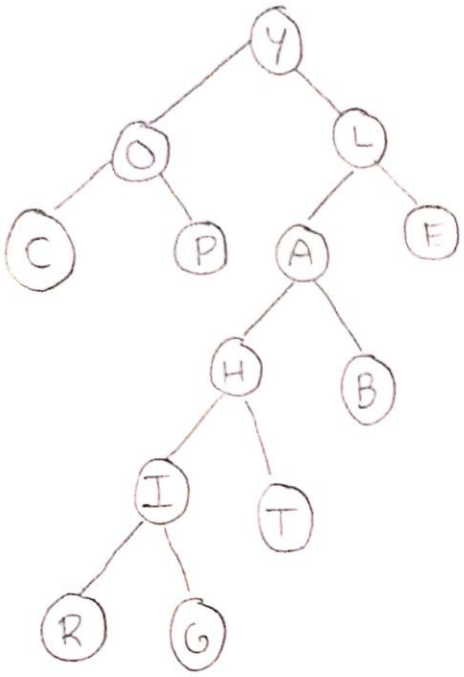


If we continue,

for the rest, continue from the left

every second element from the root is the left element ($i = 2$), the element before is the right element from the root

From L, right (element before) = E ✓
left (two elements after) = A



From A, right = B
left = H

From H, right = T
left = I

from I, right = G
left = Y
but since this is the root = R

So, now we check for post-order

Post-order : Left Right Root

C - O - P - R - G - I - T - H - B - A - E - L - Y

Hence, this tree works for both

b) For this to be a complete binary tree, we would have to insert in the ArrayList as follows:

Y-O-L-C-P-A-E-H-B-I-T-R-G

2.

a)

Algorithm depth(TreeNode t1,TreeNode t2)

Assume a tree t1
Initialise bottom left node as lnode
Initialise bottom right node as rnode

Create maxdepth(TreeNode t1, method

If t1 is empty, return 0

else

recurse maxdepth(lnode) until there is no more left node
recurse maxdepth(rnode) until there is no more right node

depth = (maxdepth(lnode), maxdepth(rnode) +1)

return depth

This method takes O(n) time since we would have to go the whole length of the tree to find the depth.

b)

Algorithm countFullNodes()

Create an empty stack stk
Initialise int count to 0
Assume tree is created making stacks in the stack tr

stk.push(root node of tree)

do

tr.pop while stk is not empty

if tr.pop is not empty (does not return an error), then count++

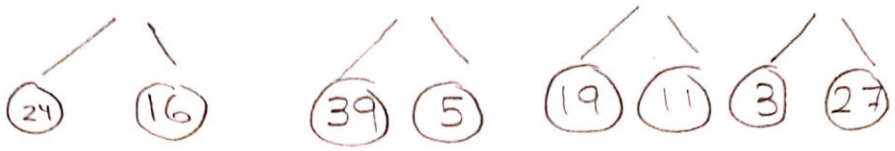
stk.push(lnode) if not empty
stk.push(rnode) if not empty

return count

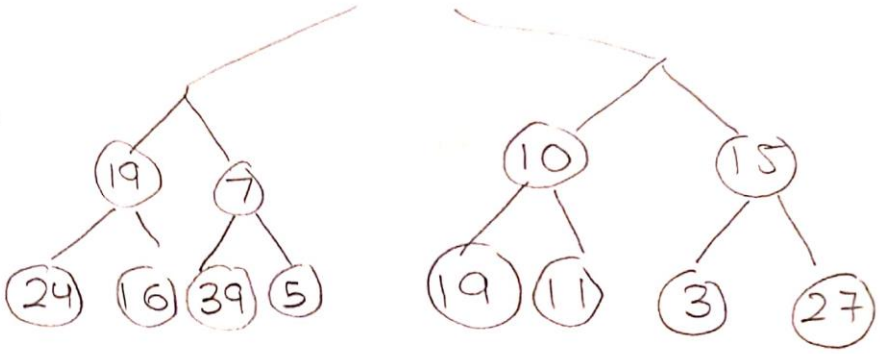
The algorithm will take O(n) time since it has to go through the whole tree atleast once

4) a) 20, 12, 35, 19, 7, 10, 15, 24, 16, 39, 5, 19, 11, 3, 27

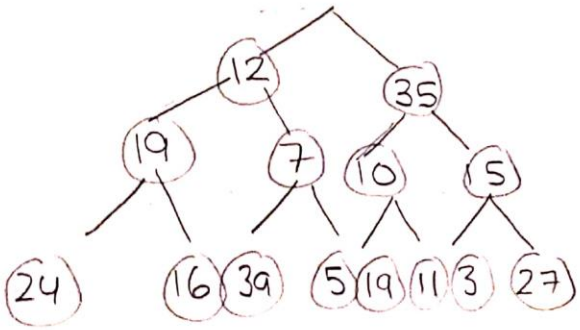
Step 1 :



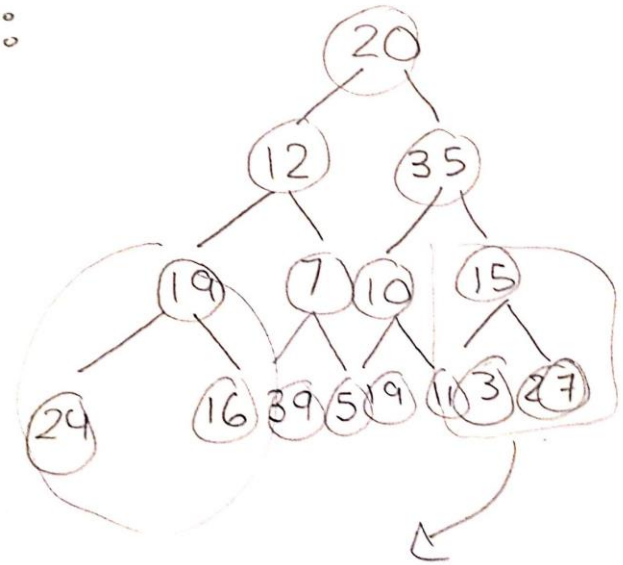
Step 2 :



Step 3 :

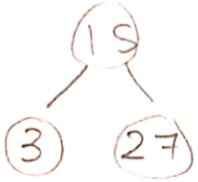


Step 9:



Take this subtree

Take this subtree

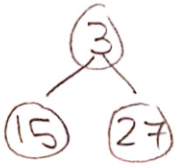


Check if child > parent

R > Root → 27 > 15 ✓

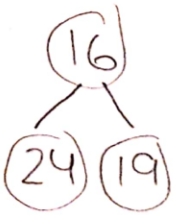
L > Root → 3 > 15 ✗ swap

new heap

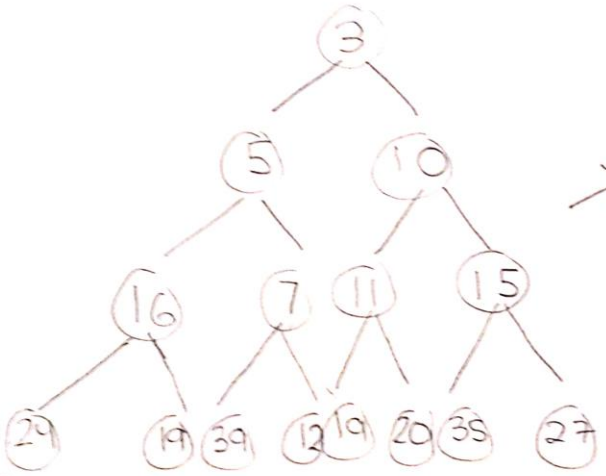


R > Root ✗ swap

L > Root ✓



By repeating these intermediate steps we get a final min-heap of



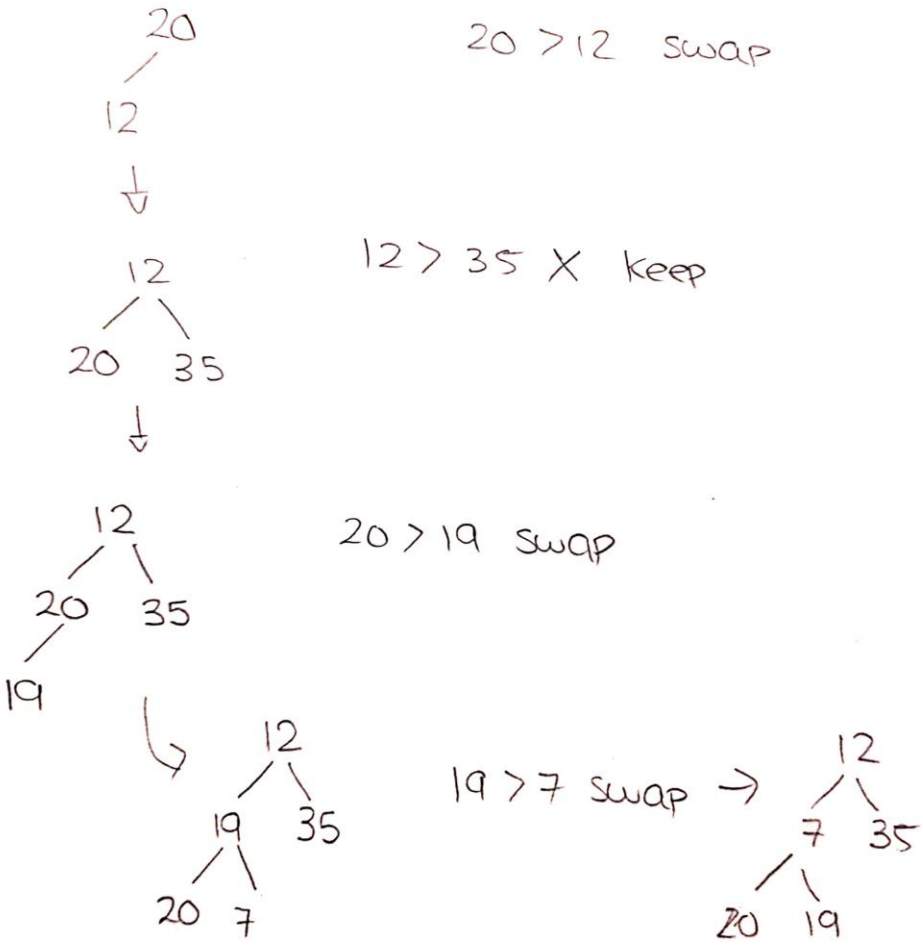
→ final min heap

b)

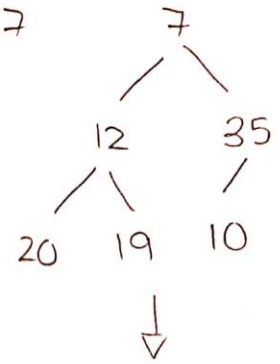
20 > 12 swap

(27)

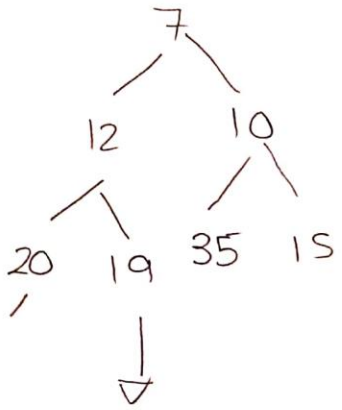
b)


20
12
20 > 12 swap
12
12 > 35 X keep
20
35
12
20
35
19
20 > 19 swap
12
19
35
20
7
19 > 7 swap →
12
7
35
20
19

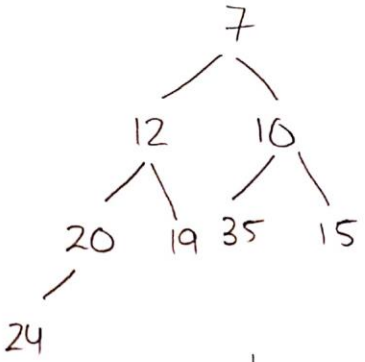
12 > 7



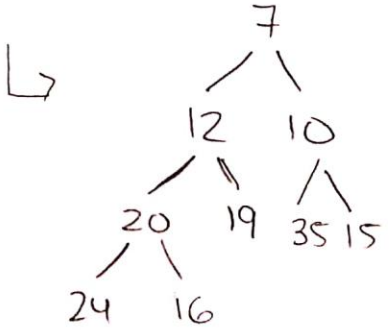
35 > 10 swap



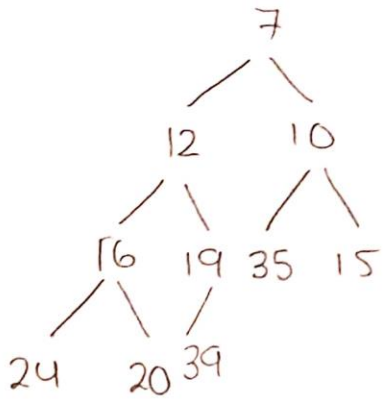
10 > 15 × keep



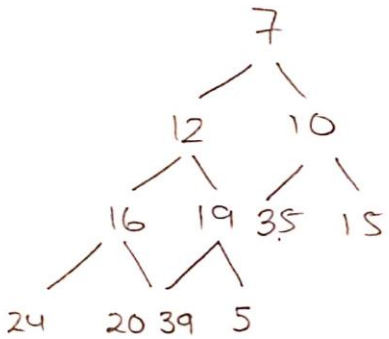
20 > 24 × keep



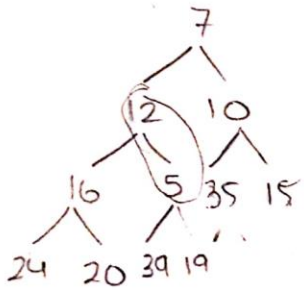
20 > 16 ✓ swap



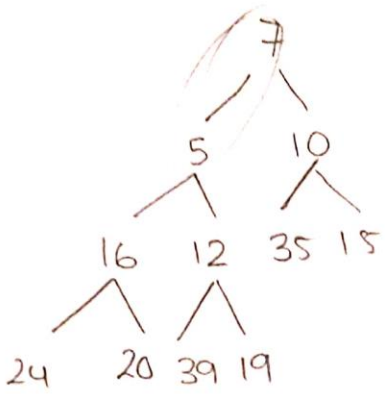
19 > 39 × keep



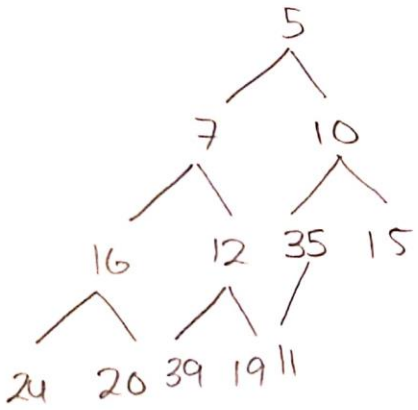
19 > 5 swap



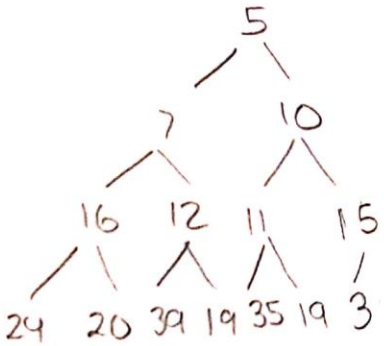
12 > 5 swap



7 > 5 swap

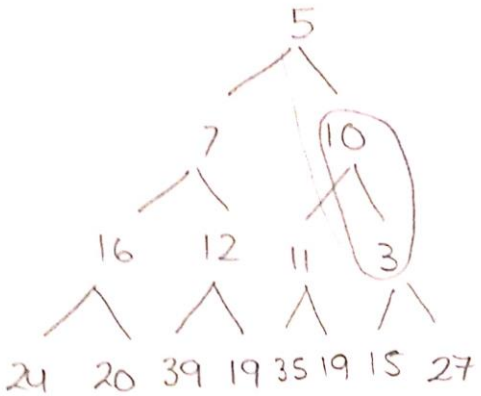


35 > 11 swap



11 > 19 X keep

15 > 3 swap



10 > 3 swap

5 > 3 swap

Hence, the final min-heap should look like