

1. Where in a min heap the largest element resides? (Assume all elements are distinct) Explain.

When $n = 1$ the largest element is trivially a leaf. Suppose that we have a min heap with n elements where $n > 1$. Suppose further that the largest element was an internal node, then that means it has atleast one child. This would be a contradiction since the largest element would be larger than its children which contradicts the min-heap property. we conclude that the largest element must be a leaf.

2. Make the min heap by successive insertions into an initially-empty min heap. Re-draw the heap each time an insertion causes one or more swaps.

$[-4, 2, 10, 12, -1, -3, 15, 76]$

Iteration tree

1.

(-4)

2.

(-4)
6

3

(-4)
6
10

4

(-4)
6
10
12

5

(-4)
6
10
12
14
16

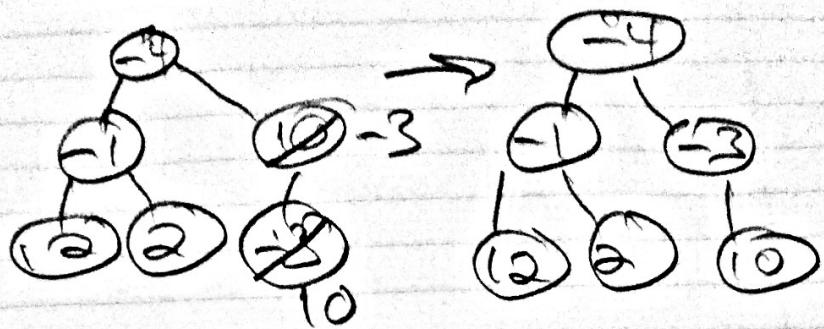


(-4)
-1
12
2
10

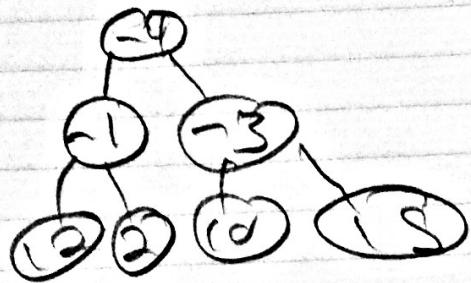
4 iteration

tree

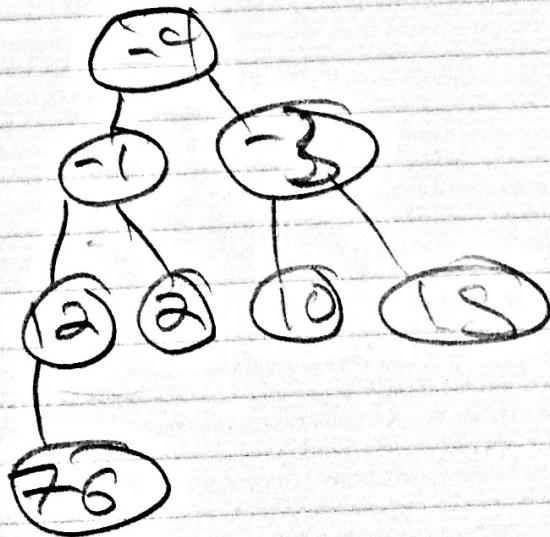
6



7



8



3. Make the max heap by successive insertions into an initially-empty max heap, Re-draw the heap each time an insertion causes one or more swaps

1. -4,2,10,12,-1,-3,15,76
2. 61,25,-12,16,20,97,-1,100

3.1

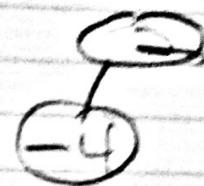
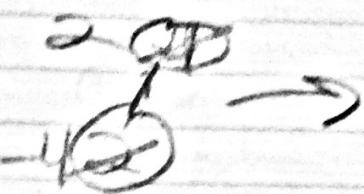
Minimum

Max

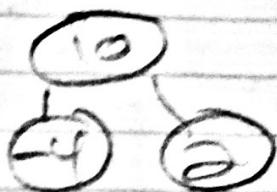
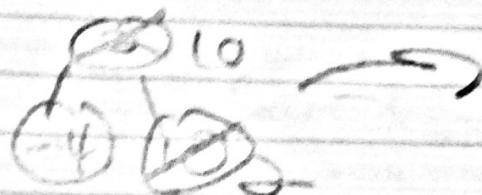
1

4

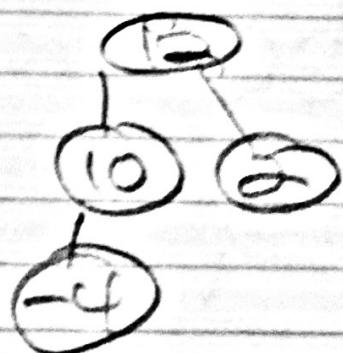
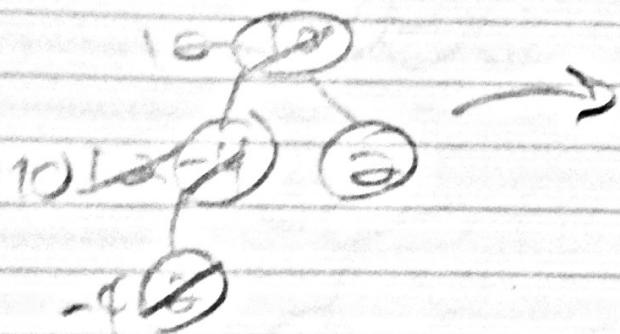
2



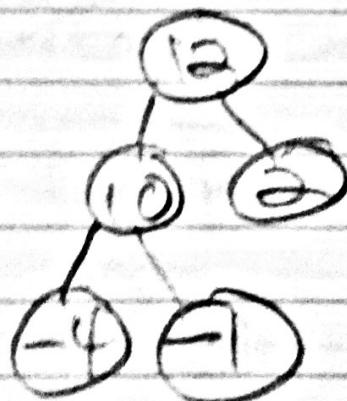
3



4



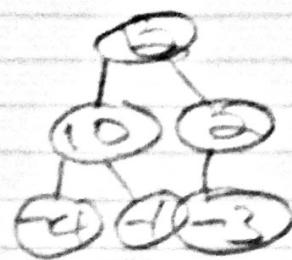
5



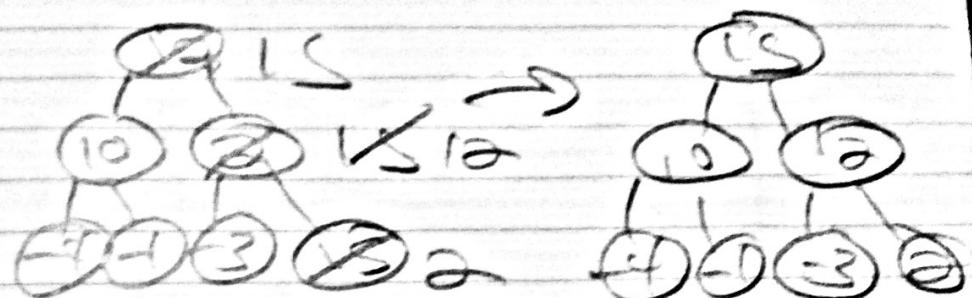
question

heap P

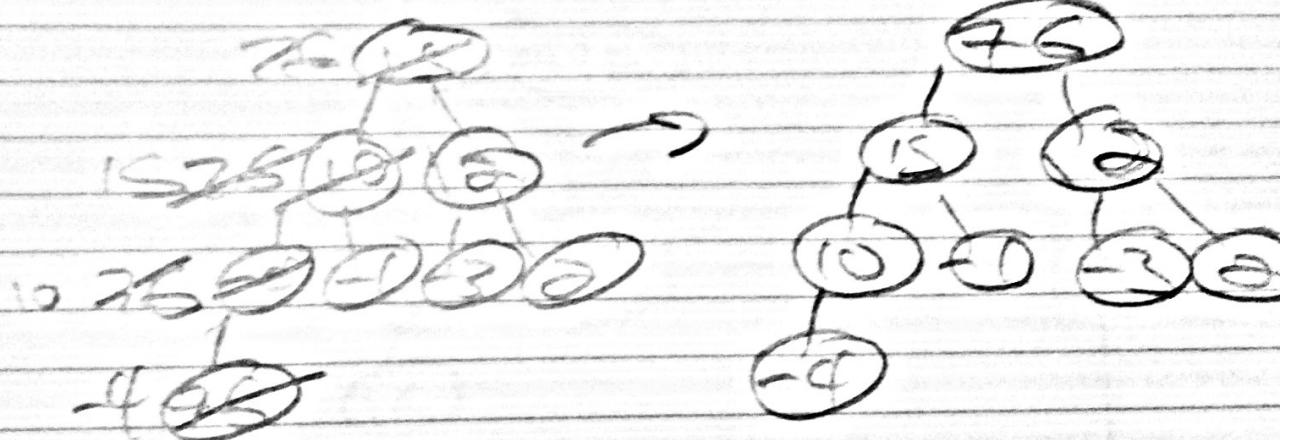
6



7



8



(3.2)

Iteration

heap

1

(61)

2

(61)
(23)

3

(61)
(23) - (15)

4

(61)
(23) - (15)
(15)

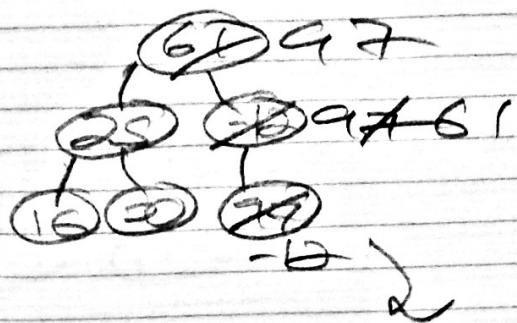
5

(61)
(23) - (15)
(620)

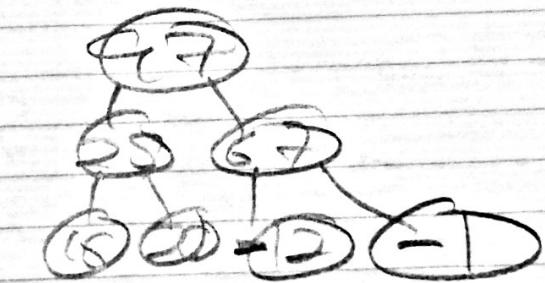
Iteration

6

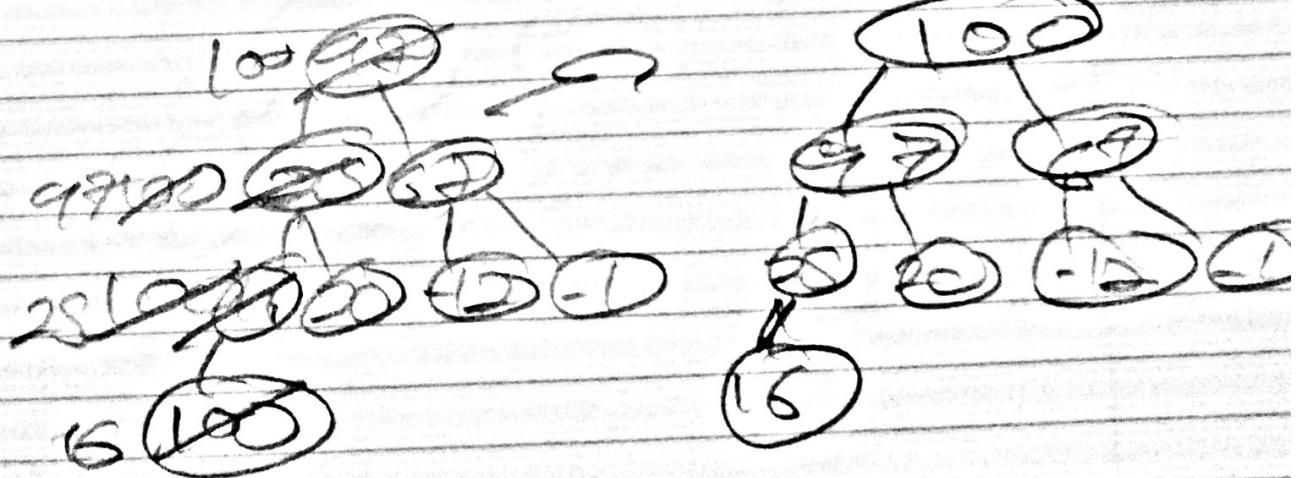
Heap



7

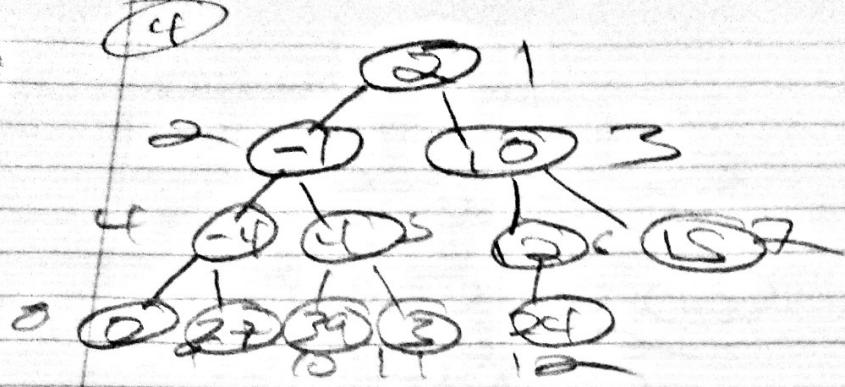


8



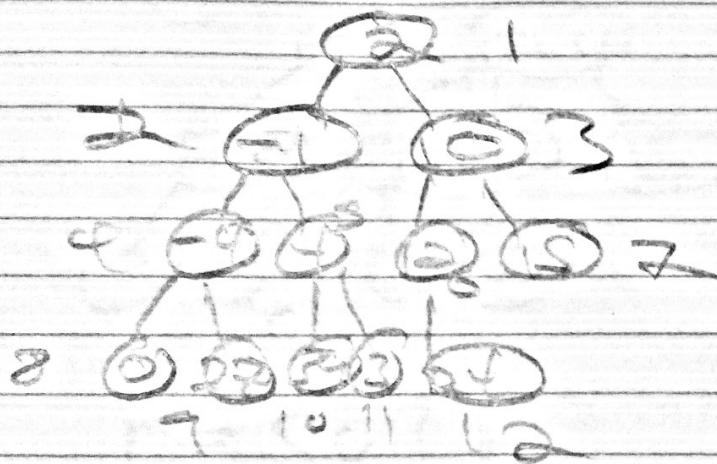
- 4.** Is this a max-heap? if not, apply max-heapify function to change it to a max heap.

[2, -1, 10, -4, 4, 12, 15, 0, 27, 39, 3, 24]

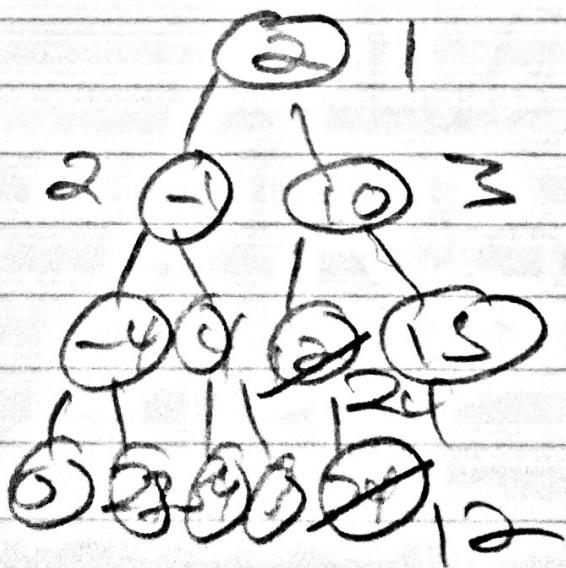


max-heapify(a , 7)

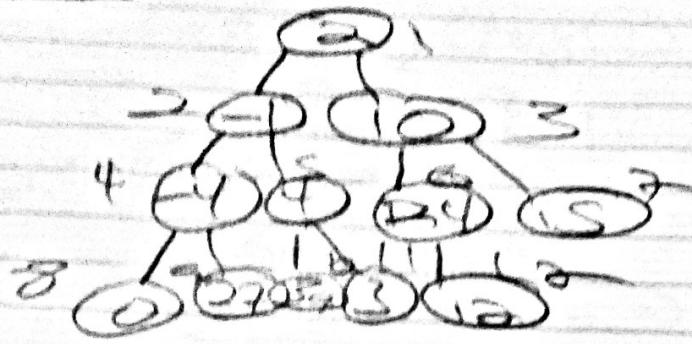
Result



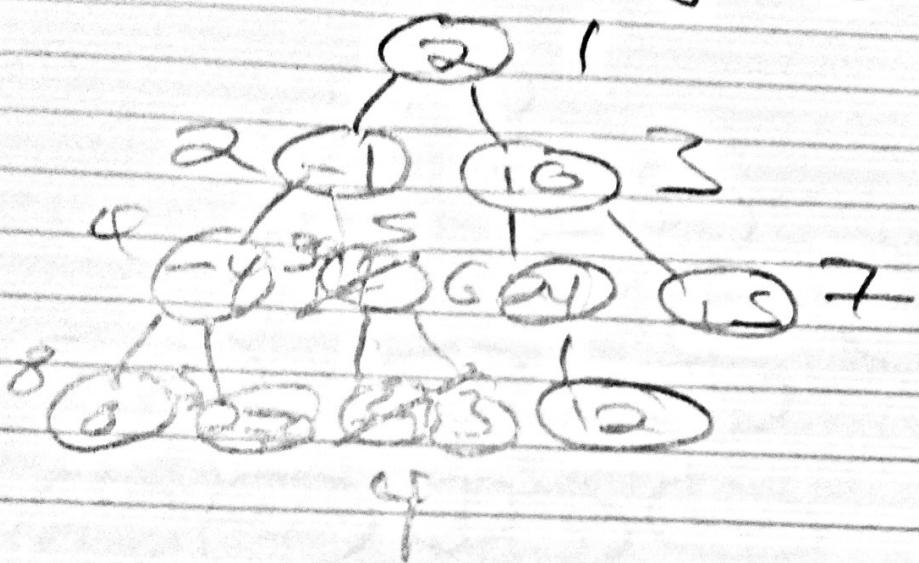
max-heapify(a , 6)



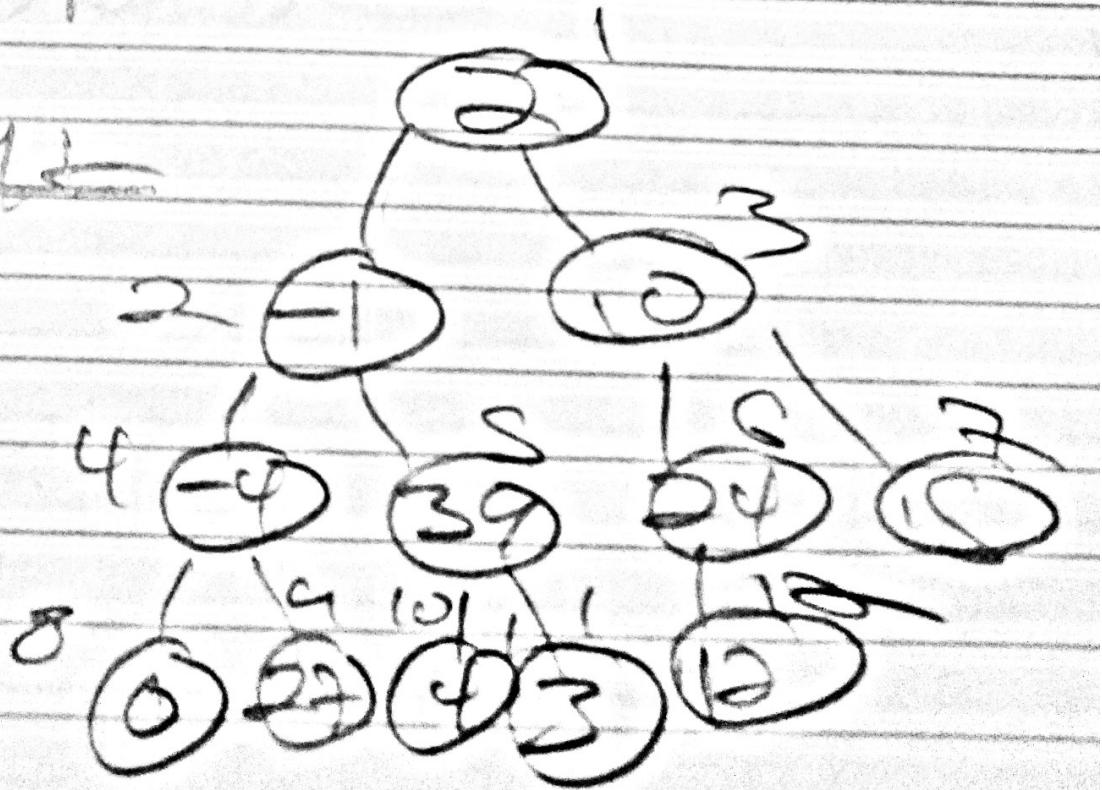
result



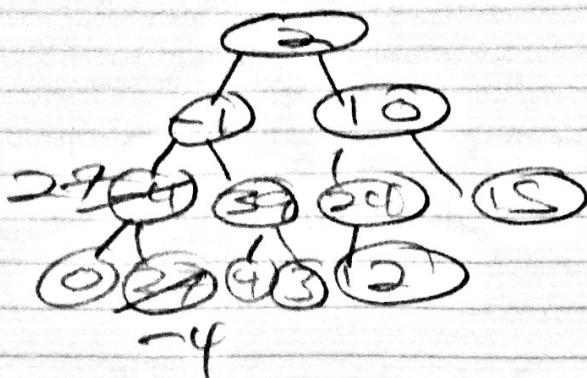
~~next-hearty (age 5)~~



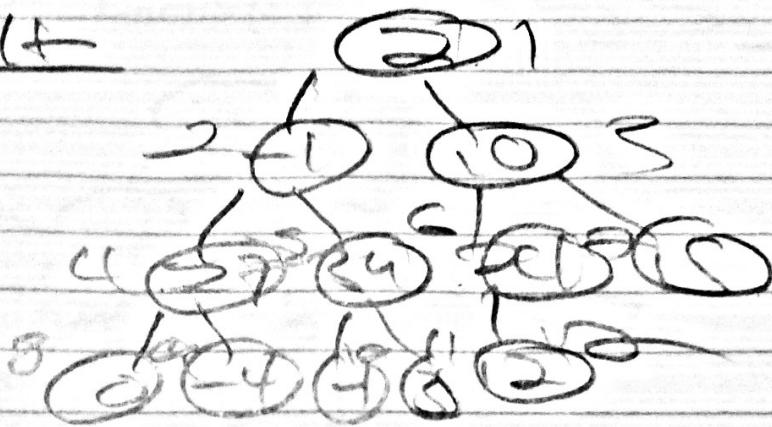
resell



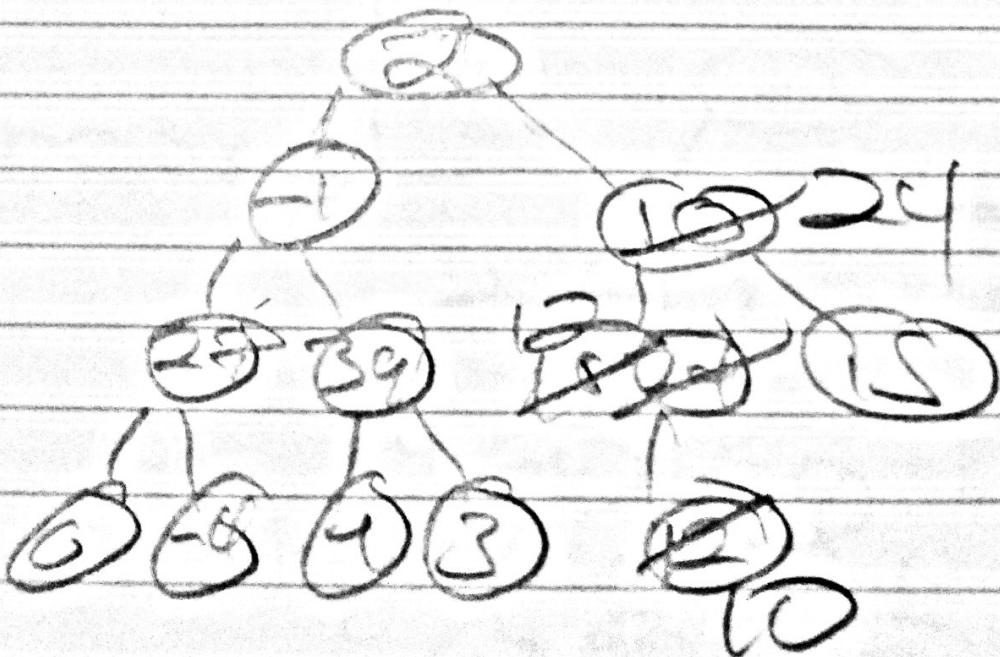
max heapify (A[3]) -



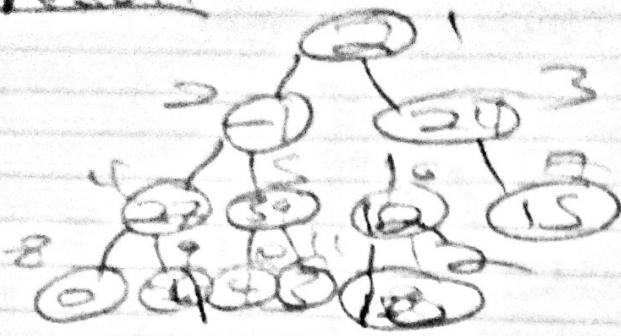
result



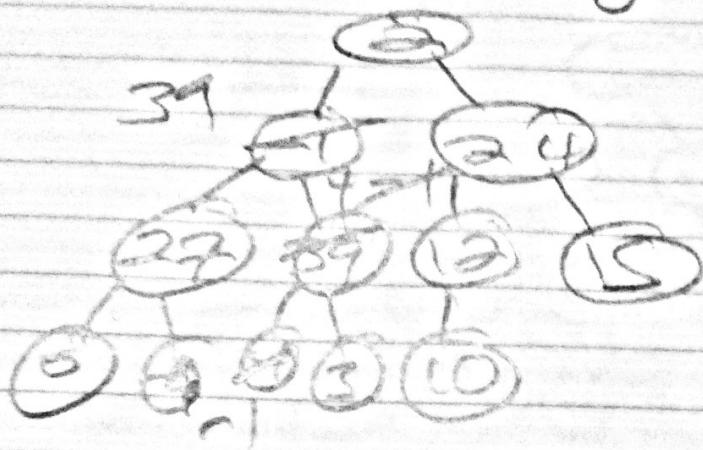
max heapify (A[3])



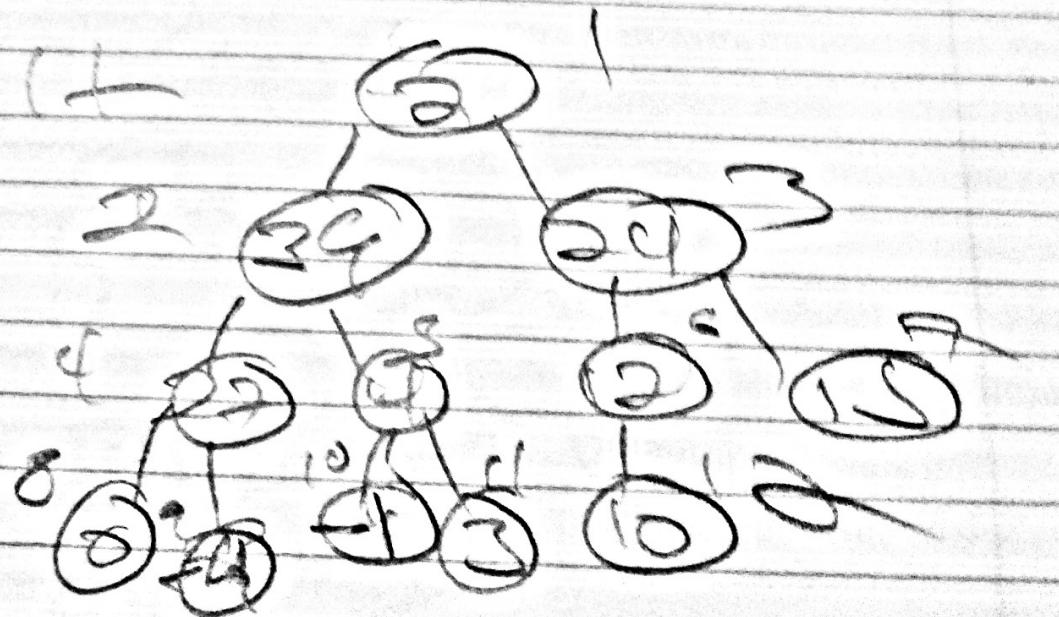
result



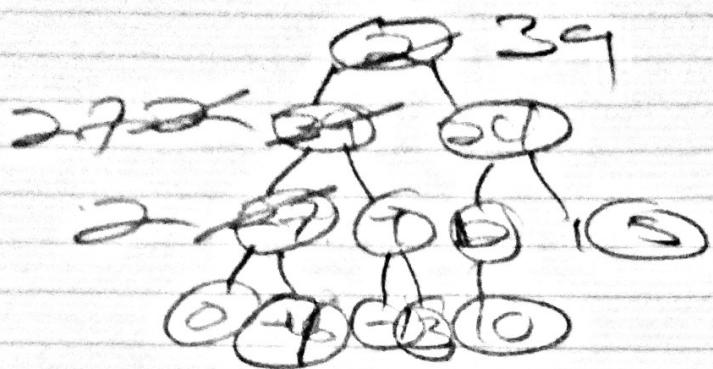
next-Leap by (A) 23



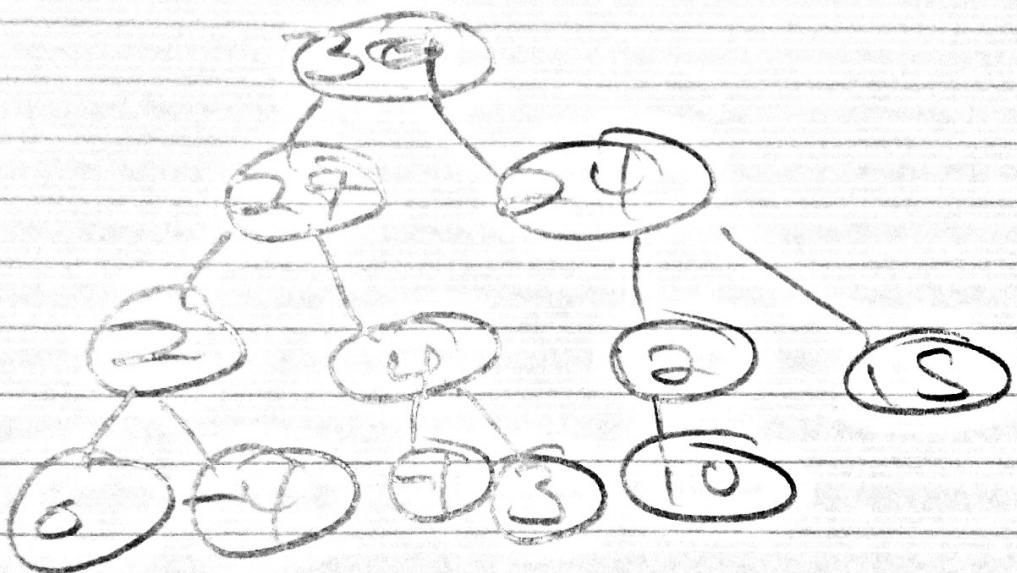
result



max-heaps by CBJD

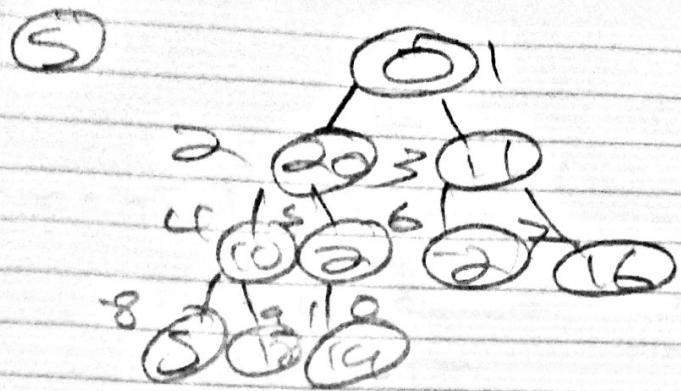


result

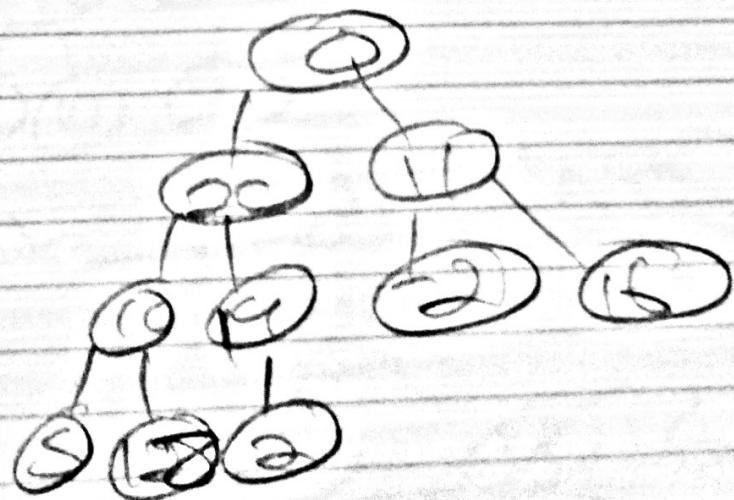
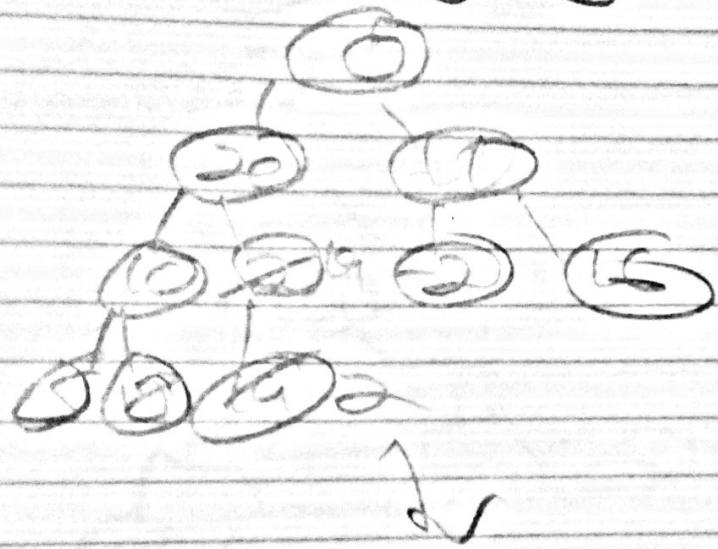


- 5.** Is this array max-heap? If not, change it to the max heap by putting the elements on a binary tree and then applying the max-heapify function.

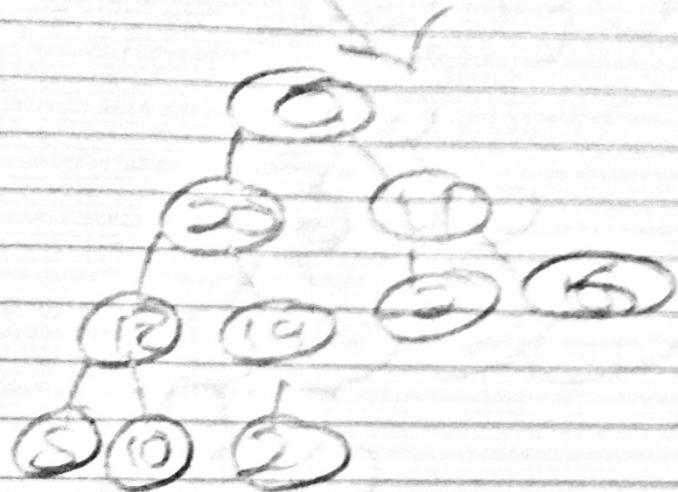
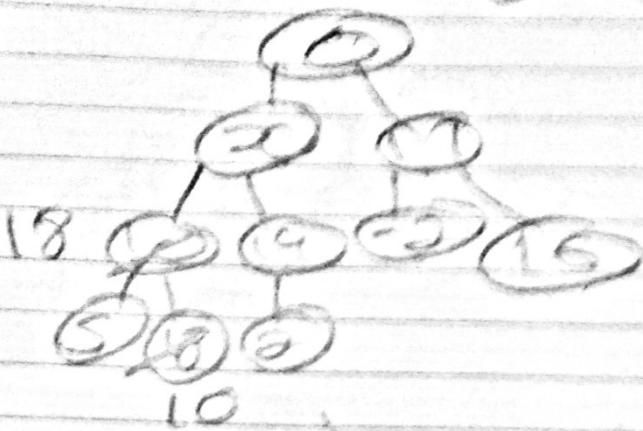
0, 2, 11, 2, -2, 16, 5, 18, 19



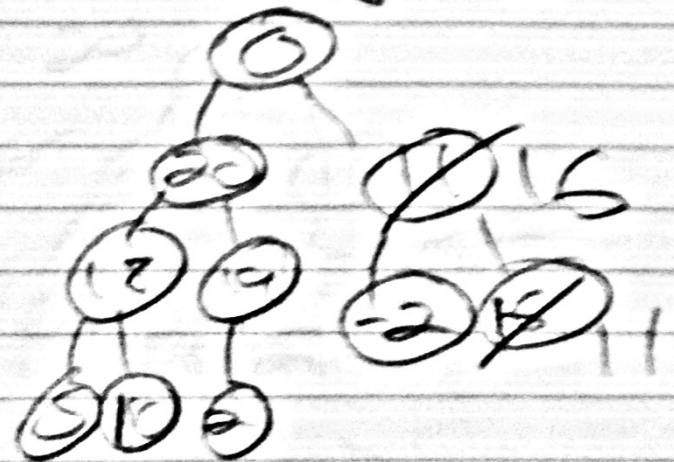
new leaf of A(5)



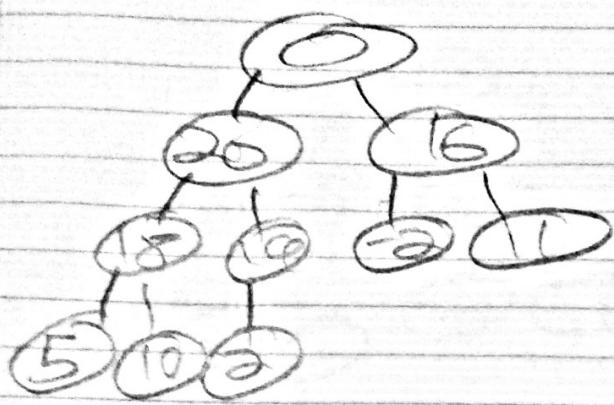
max heapify (by 1)



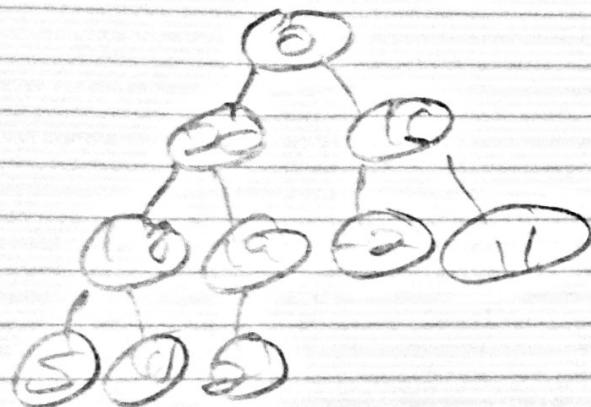
max heapify (A[3])



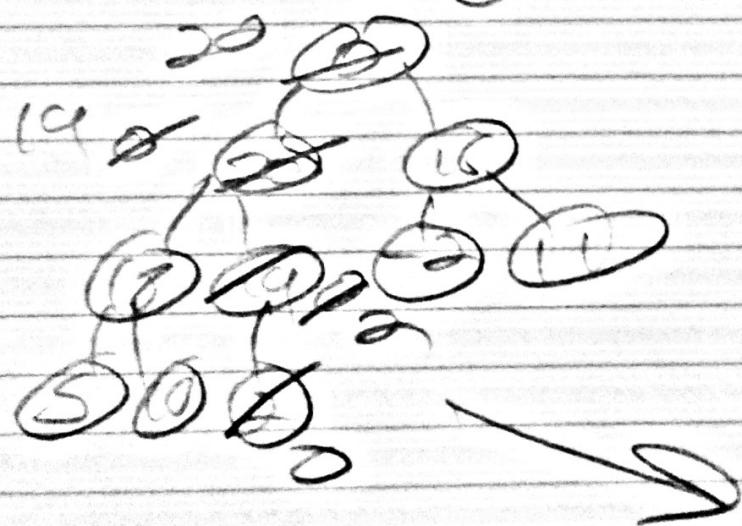
2

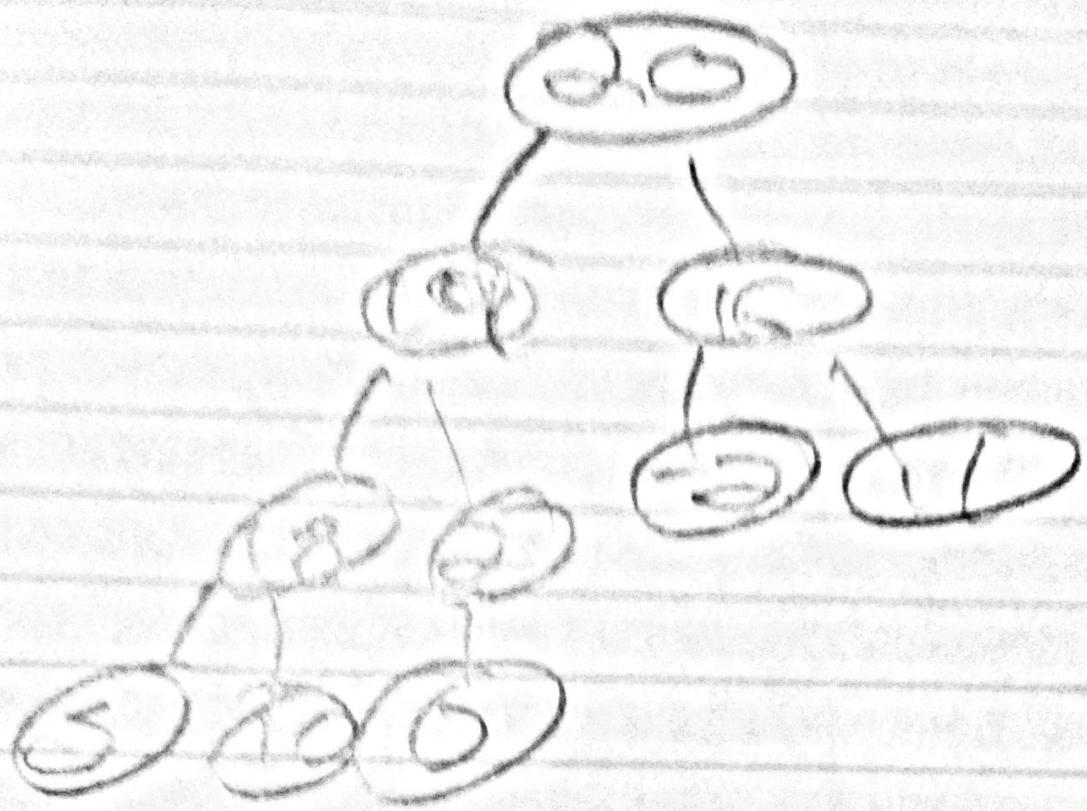


new-tree(A[2])



new-tree(A[1])





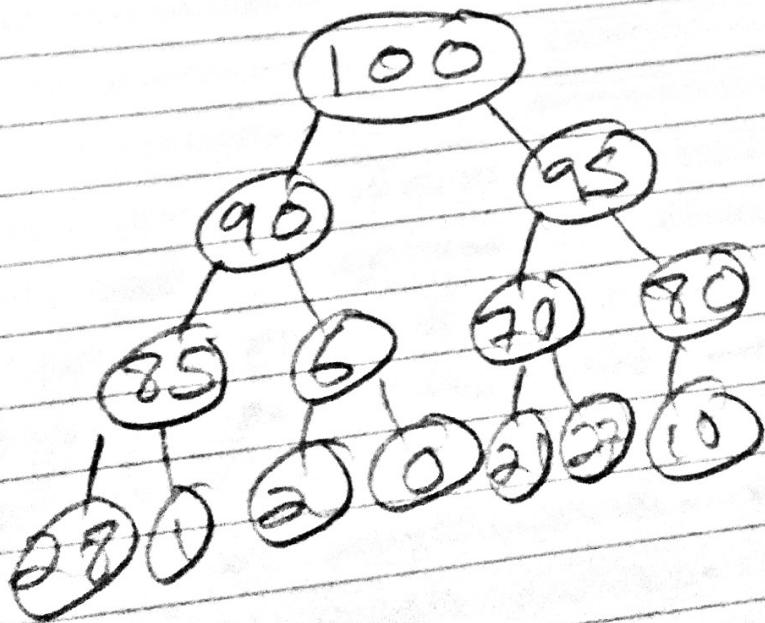
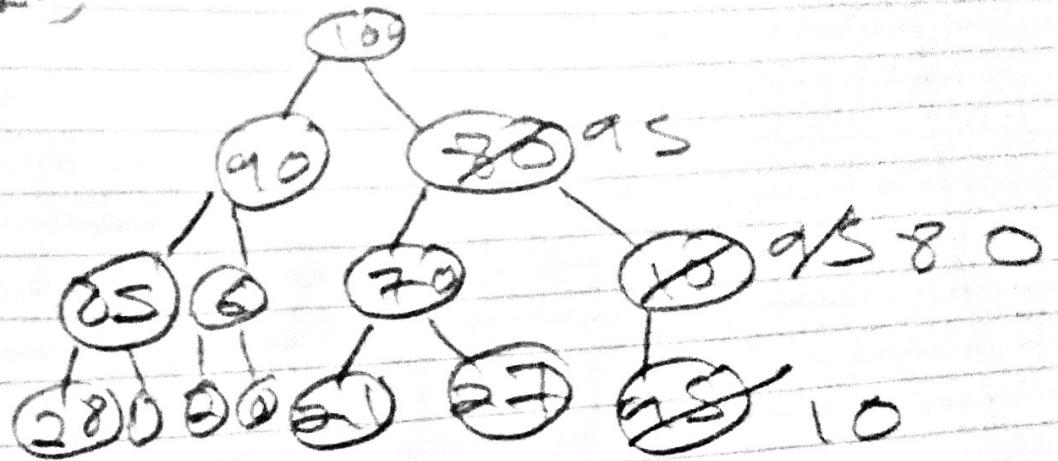
6. Insert

1. 95
2. 160
3. 37

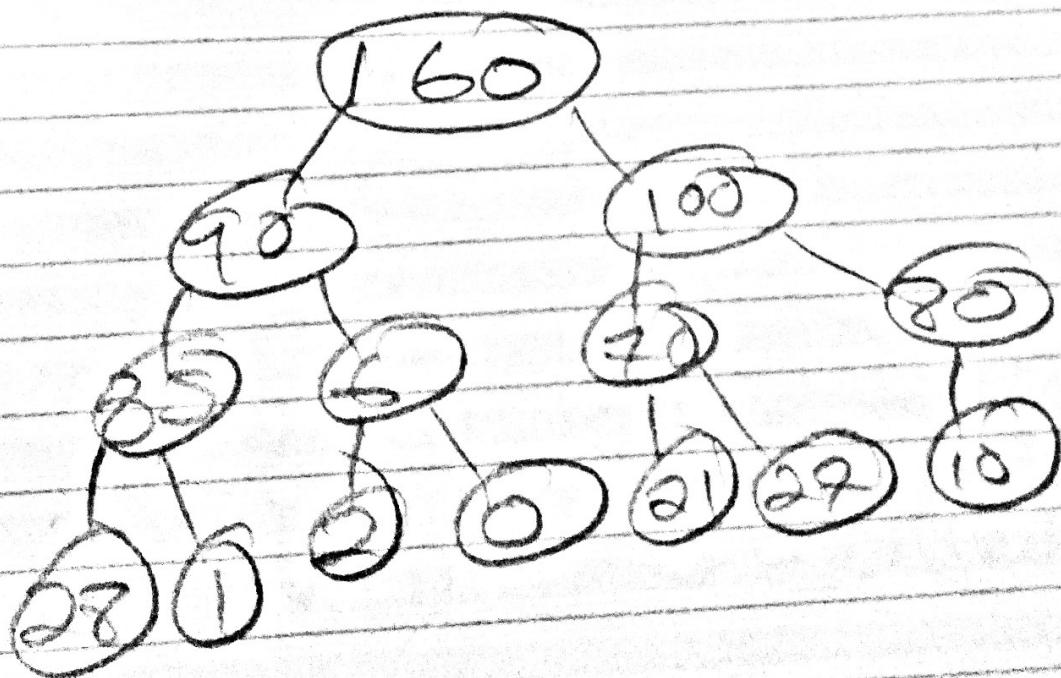
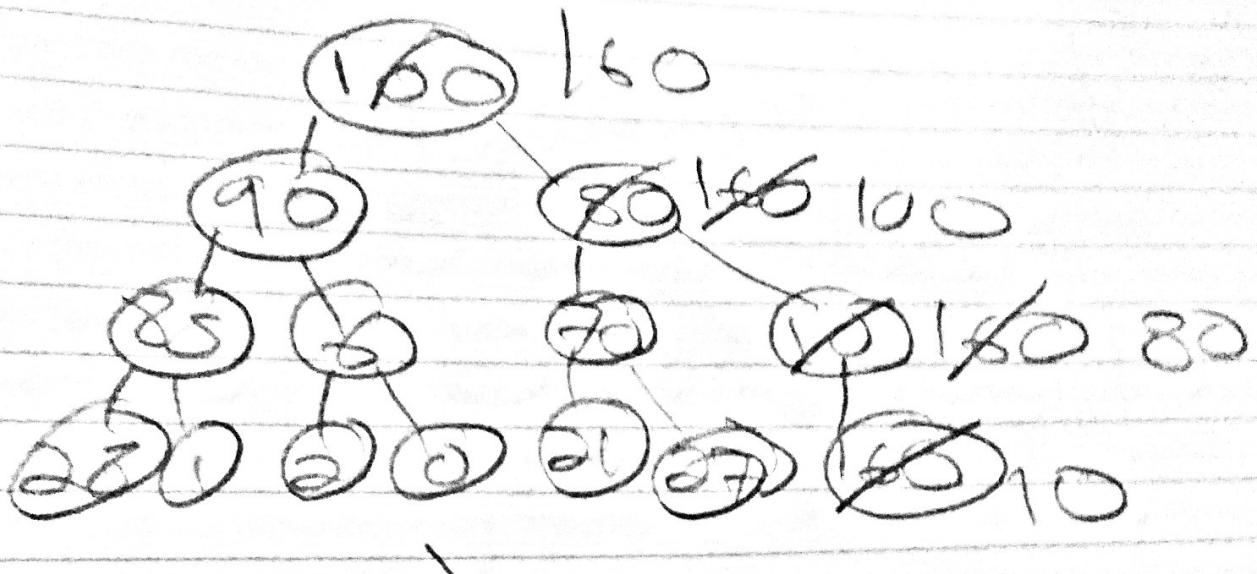
into the below max-heap

[100, 90, 80, 85, 6, 70, 10, 28, 1, 2, 0, 21, 27]

6.1 Insertion
95

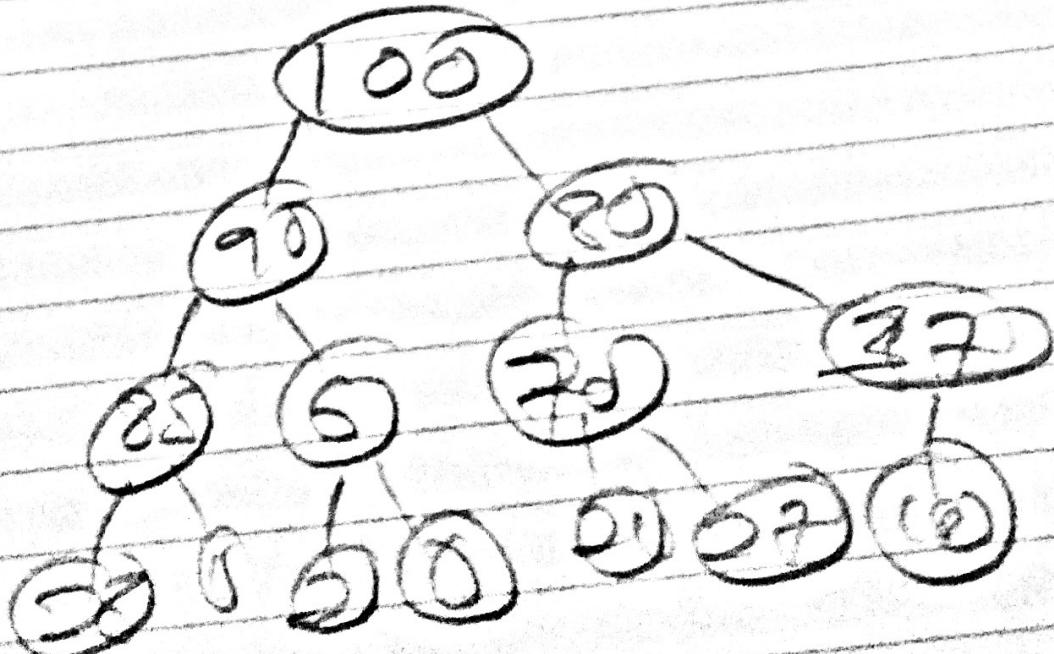
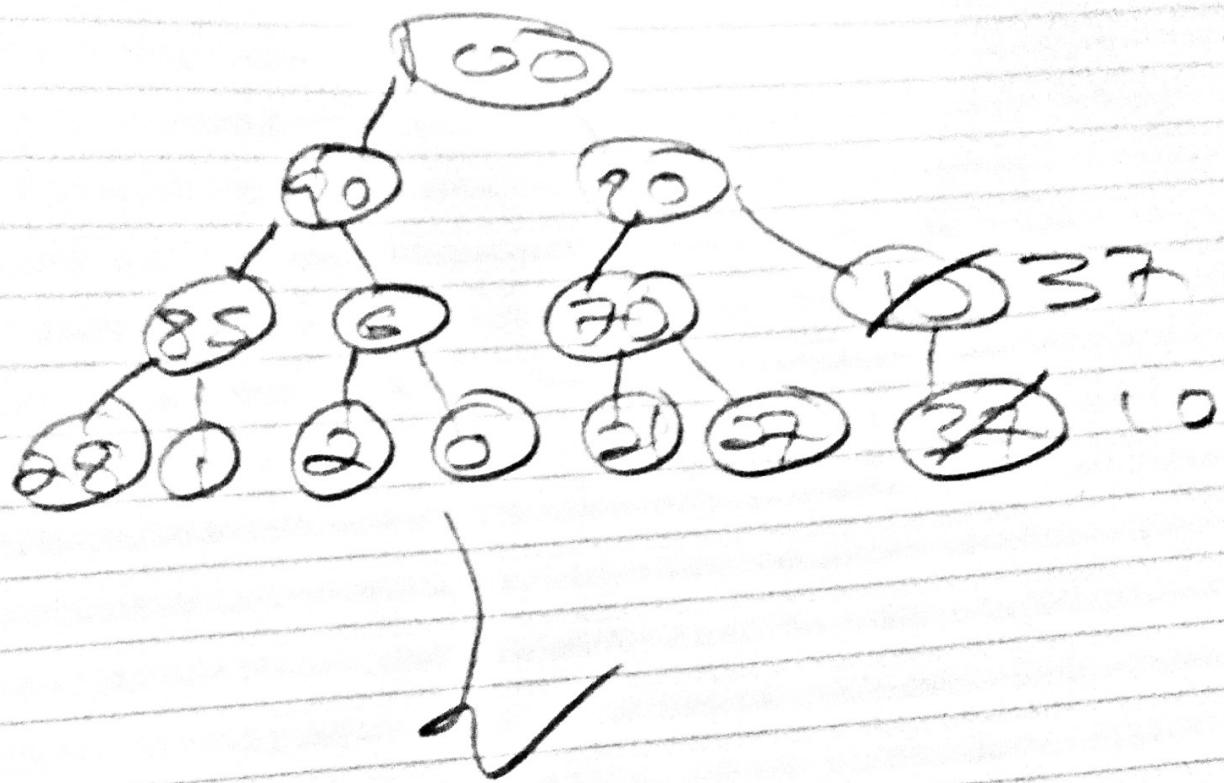


6.2 Inserting 160



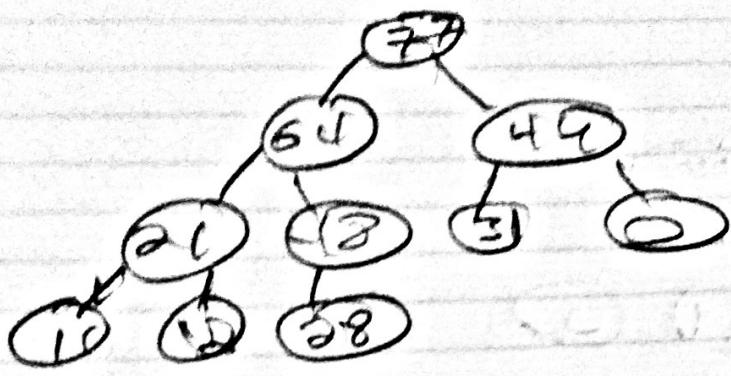
6.3

inserting 37

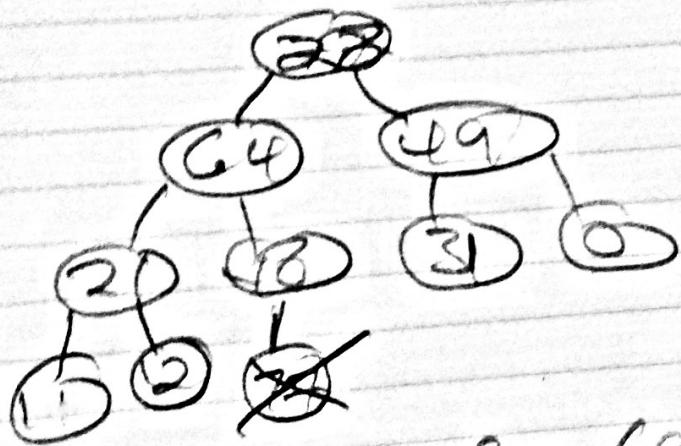


7. Delete the root of the below max-heap

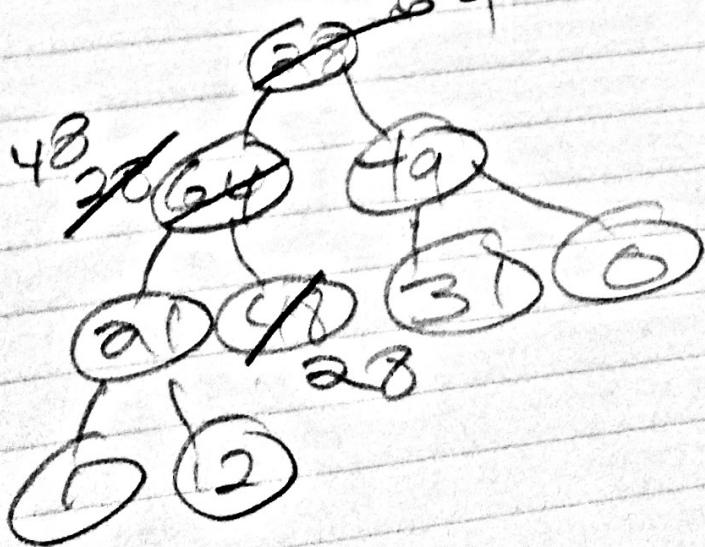
[77, 64, 49, 21, 48, 31, 0, 1, 12, 28]



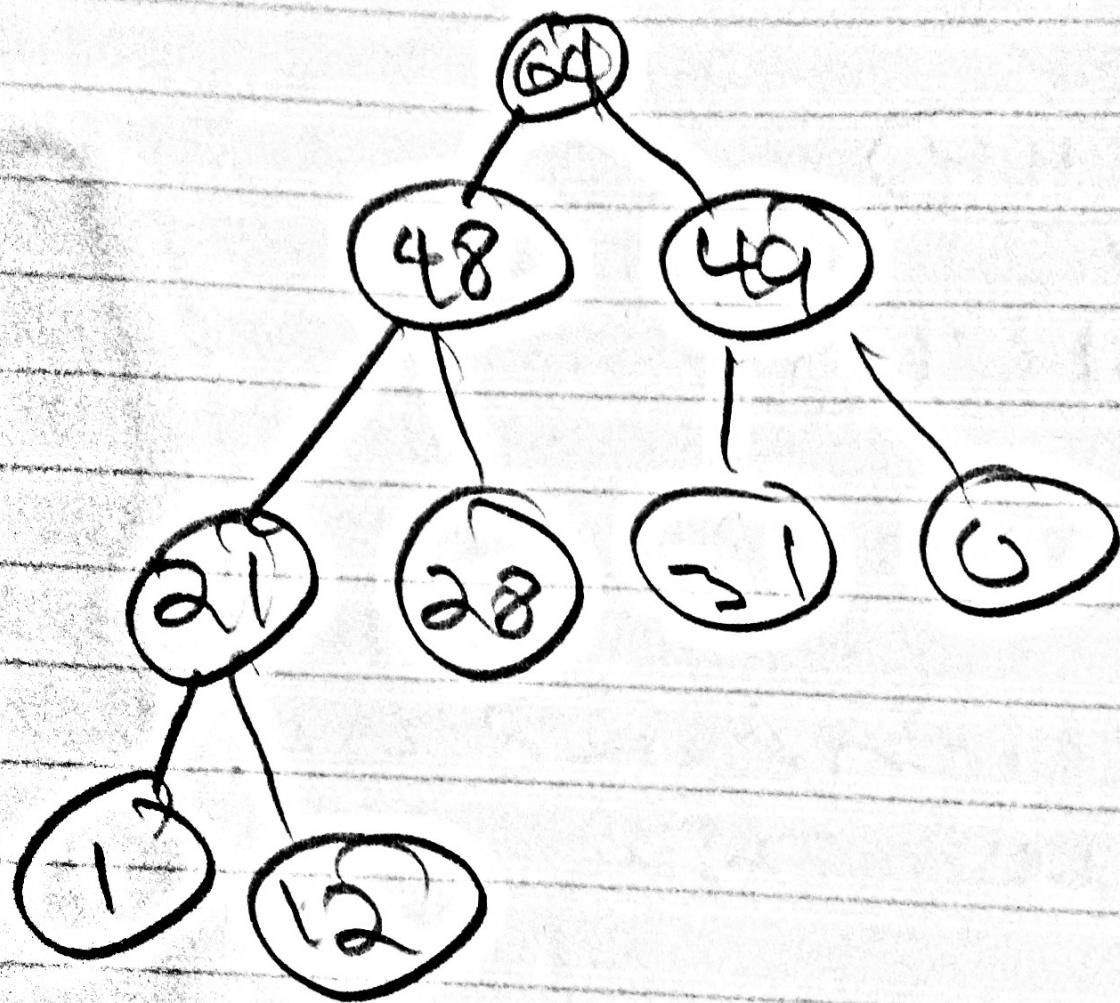
Swap 77 and 28
then reverse 72



now leafify (a) (b)



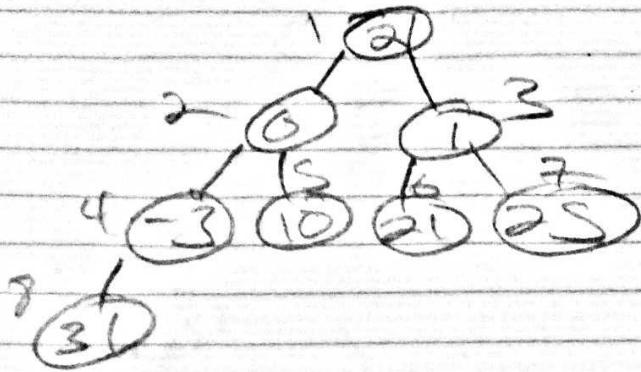
result



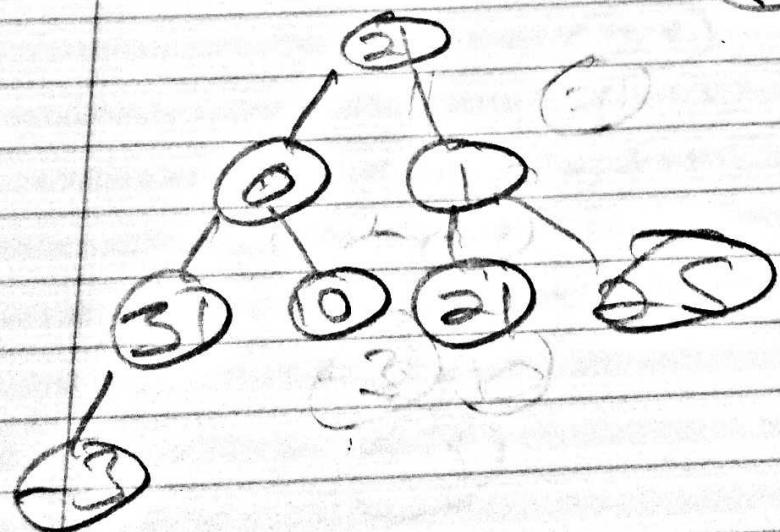
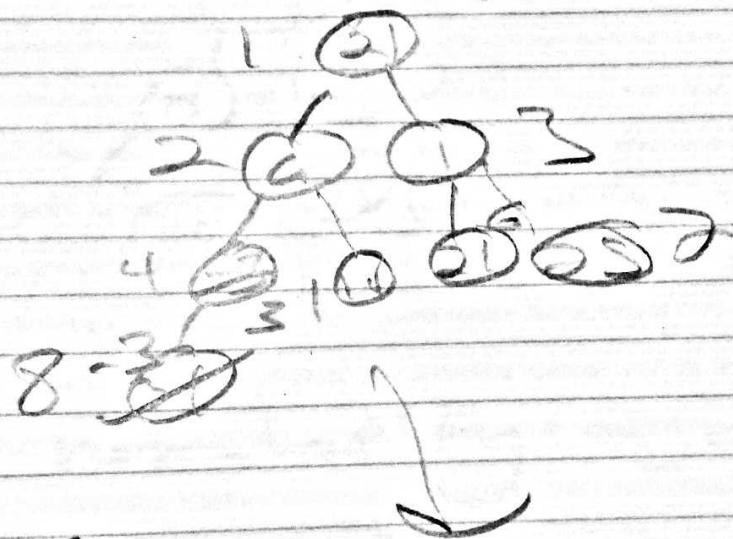
8. Sort the below array using heapsort

1. 21,0,1,-1,-3,10,21,25,31
2. 87,21,0,-1,-22,10,17,11,-3,10

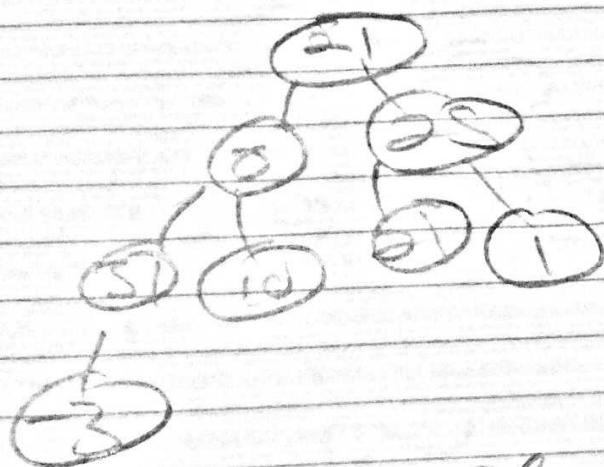
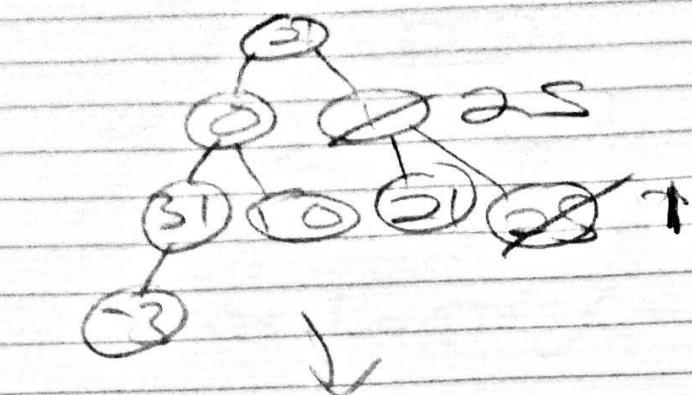
8.1 $21, 0, 1, -3, 10, 21, 25, 31$



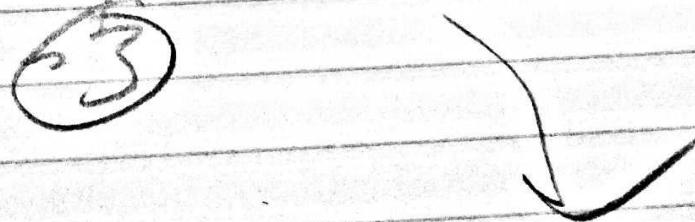
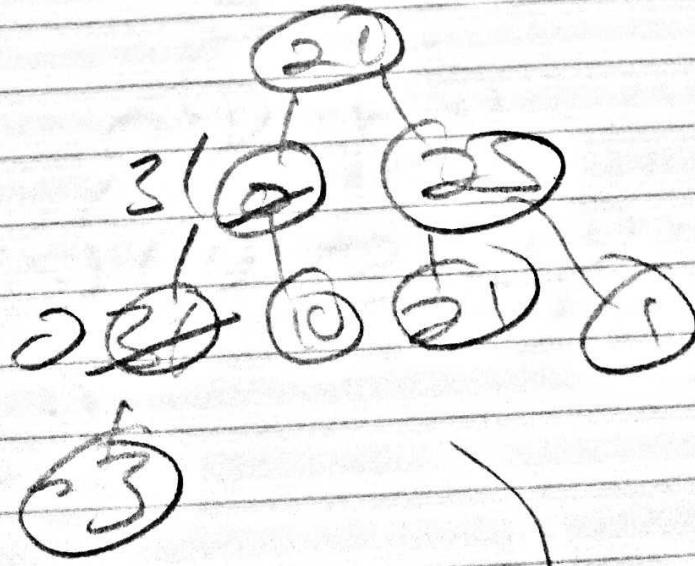
max-heapify($i=4$)

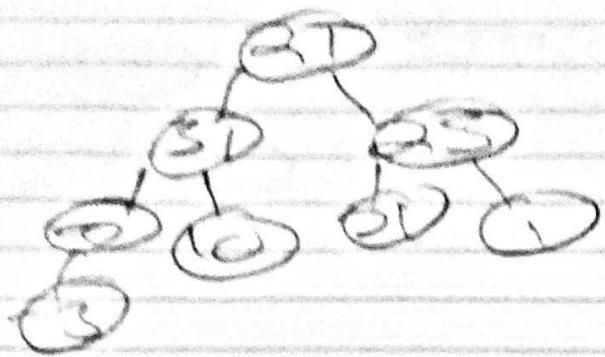


max-heapify($\alpha, j=3$)

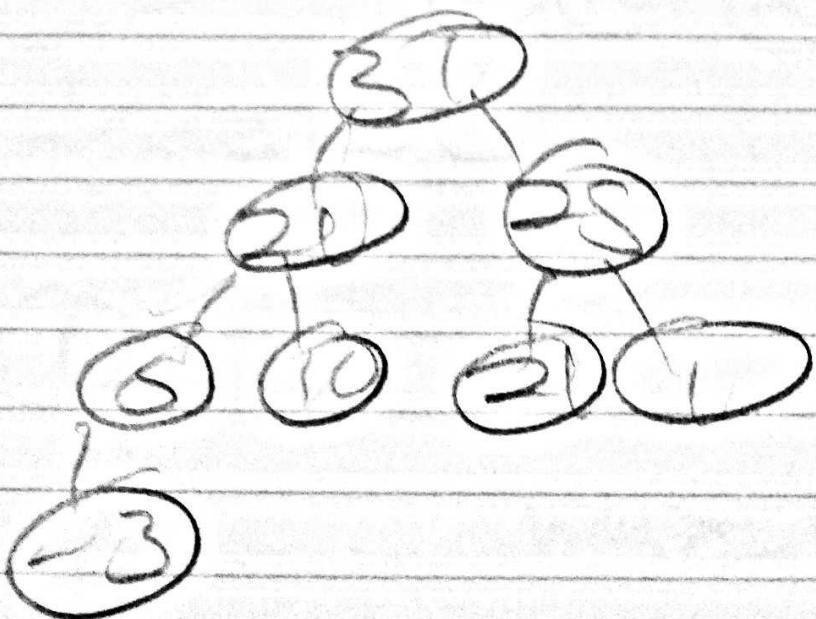
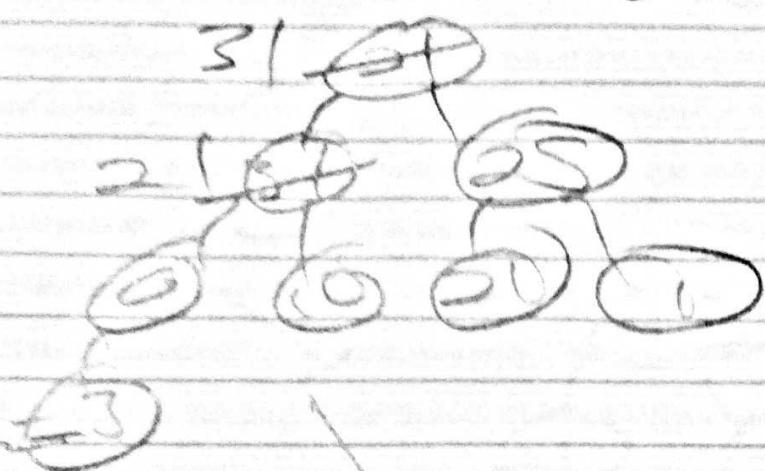


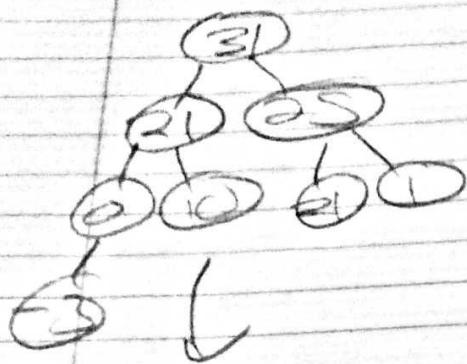
max-heapify($\alpha, j=2$)





max-leafs(max 1)

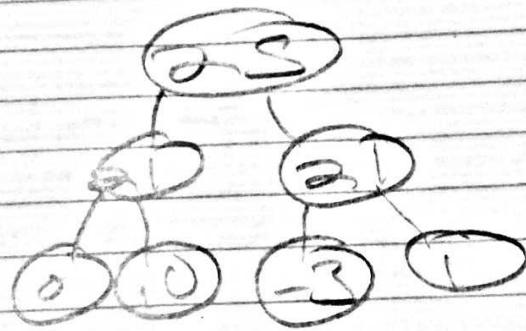




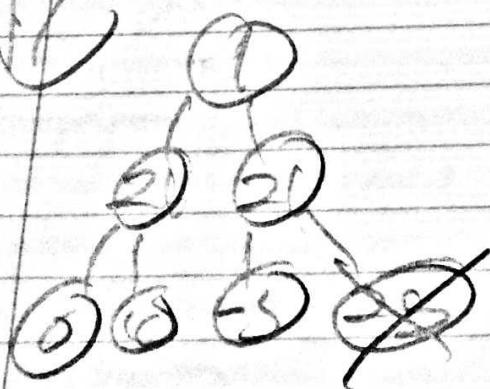
delete root



max-right-reg(c_{j0})

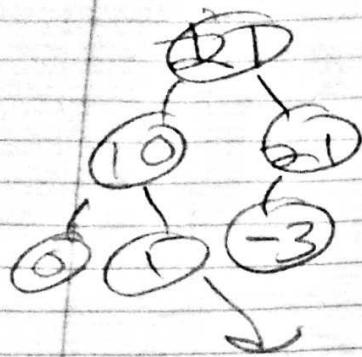


delete root



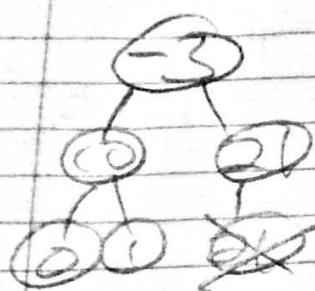
max-left-reg(c_{j0})



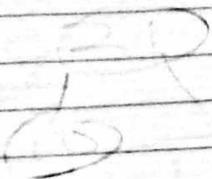


delete
root

-----1216531-----

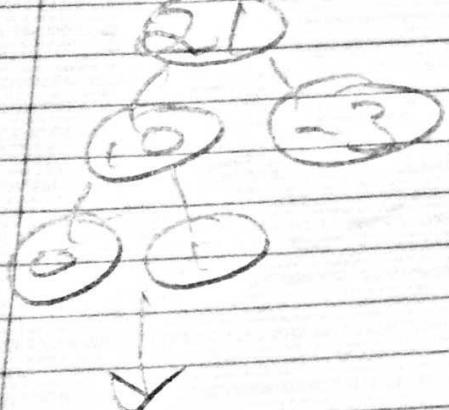


max-heapify(~~21~~)

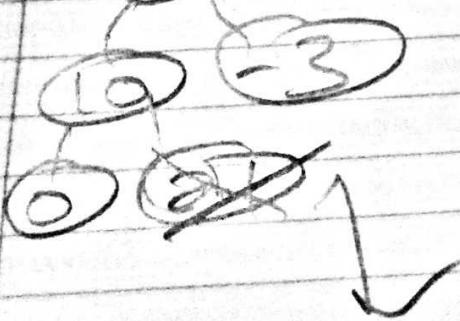


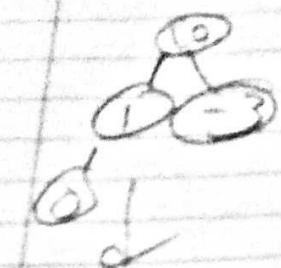
delete
root

-----216121523-----



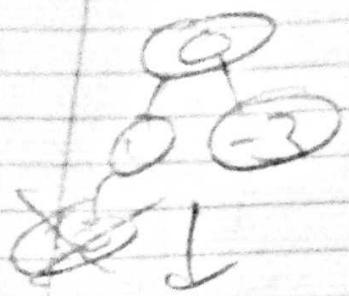
max-heapify(~~21~~)



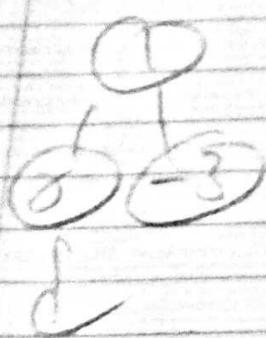


select
root

... 1 0 2 1 5 2 8 1

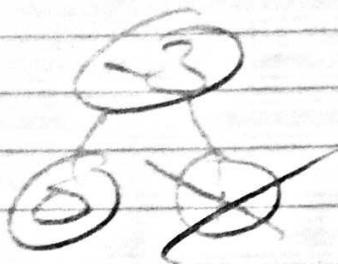


not least freq(1)

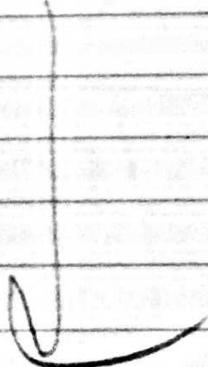


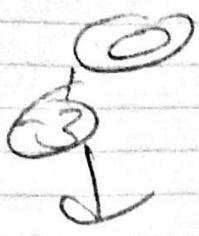
select
root

... 1 0 2 1 5 2 8 1 2 3 3

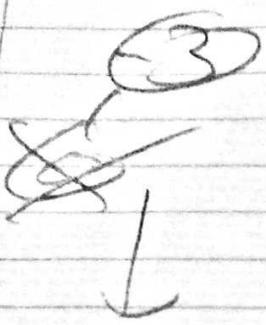


not least freq(1)





Delete
node



max heapify (c, j)



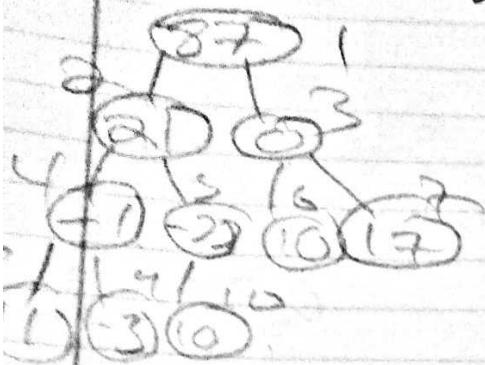
Delete node



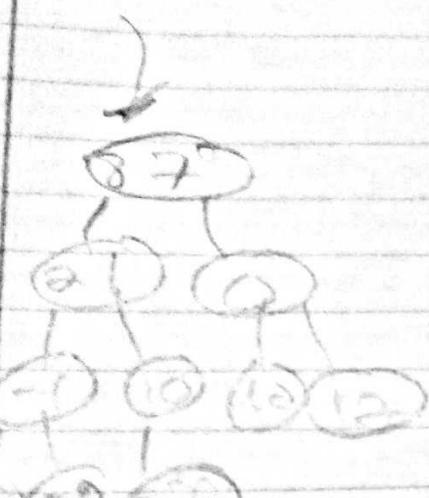
redistribute array

-3	0	1	10	2	2	2	2	3
----	---	---	----	---	---	---	---	---

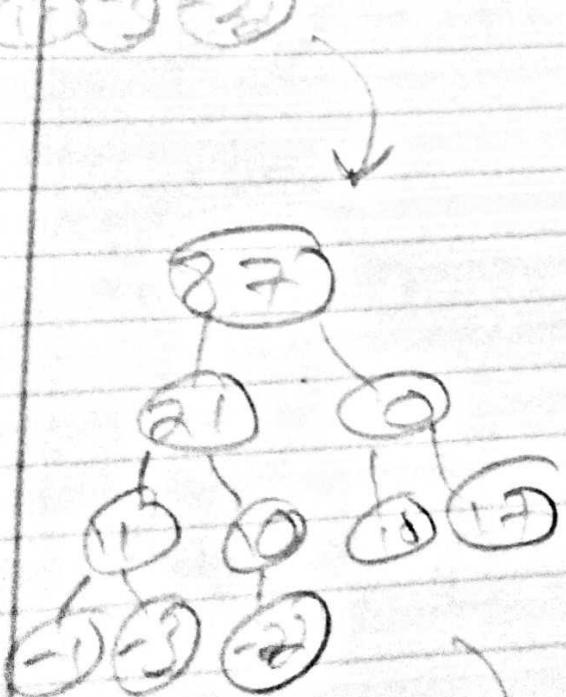
8.2 87, 21, 05 - 11-22, 19, 17, 11-3, 10



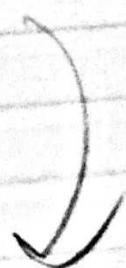
max-heapify($i, 5$)

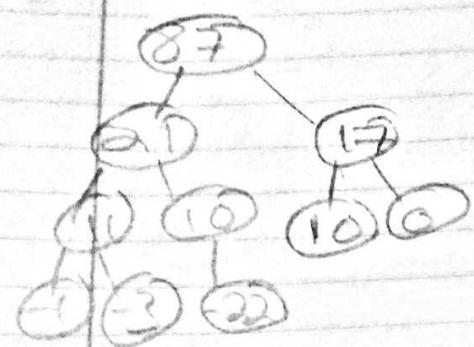


max-heapify($i, 4$)



max-heapify($i, 3$)

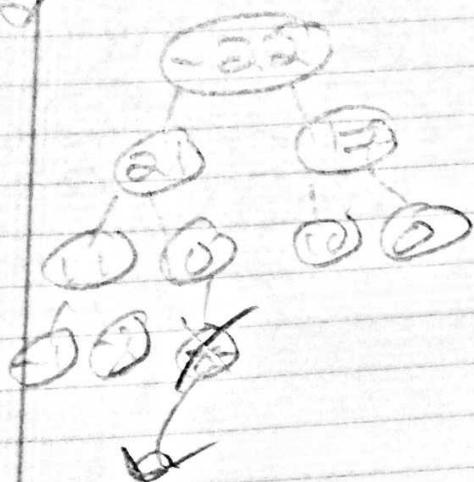




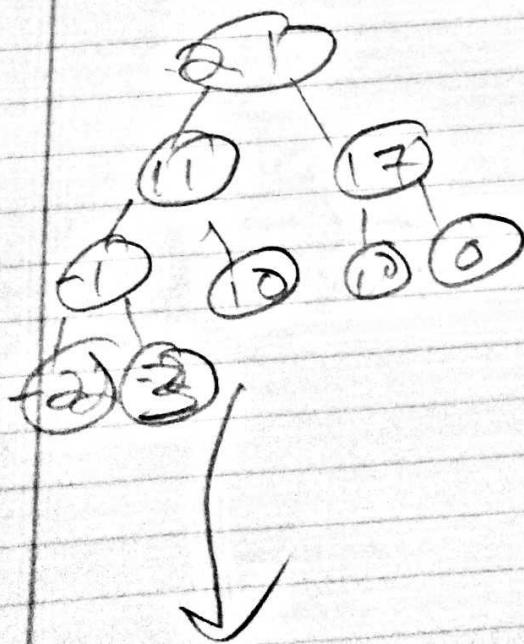
max-leafify(ay2) ✓
new-happy(ay1) ✓

delete root

heap [87]

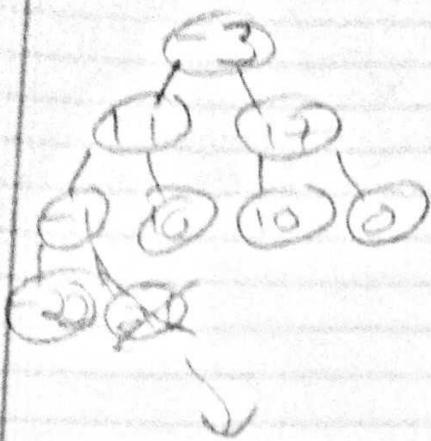


max-leafify(ay1)

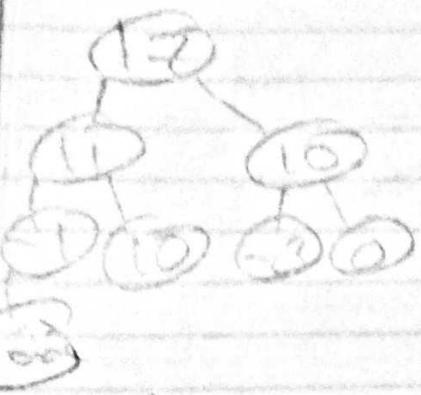


delete
root

heap2 [87]

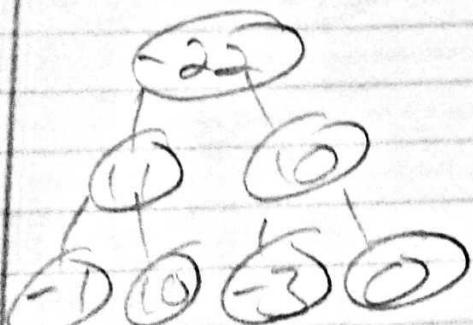


max-leftify(ayD)

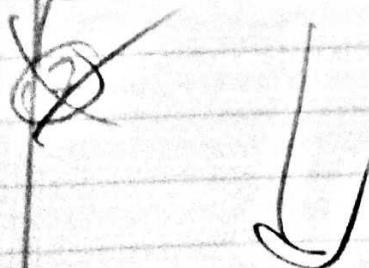


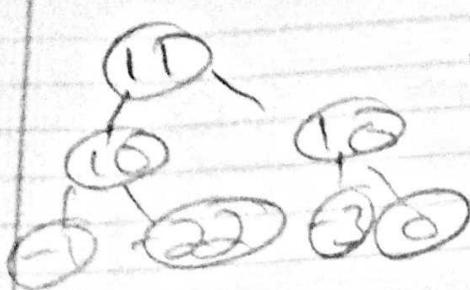
delete
root

Troop 172187



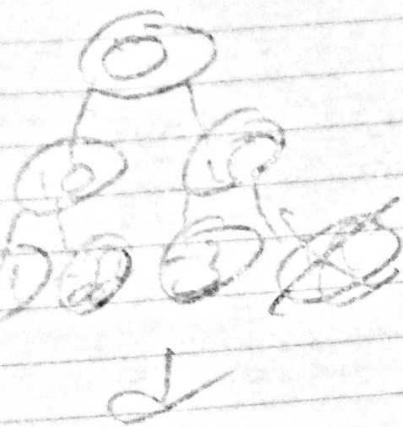
max-leftify(ayD)





delete
root

~~1 loop 11 17 21 87~~



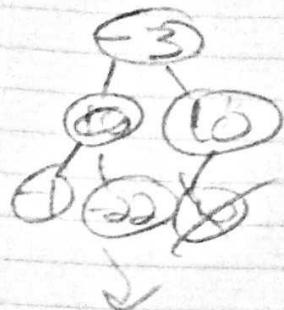
no delete yes D



desc. root

~~1 loop 10 11 17 21 87~~

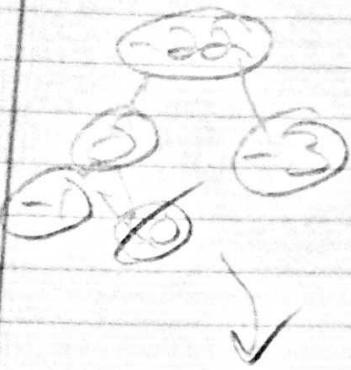
max-leafify (a)



delete
root

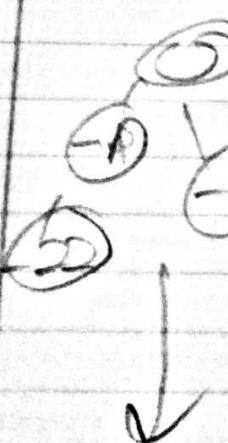
Tree	0	6	10	11	17	2	87
------	---	---	----	----	----	---	----

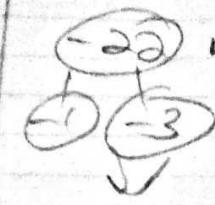
max-leafify (a)



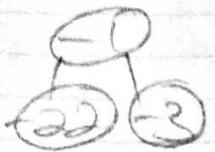
delete
root

Tree	0	6	10	11	17	2	87
------	---	---	----	----	----	---	----





new heapify [cyl]



delete
root

T heap = [-1 | 0 | 1 | 1 | 10 | 17 | 2 | 87]

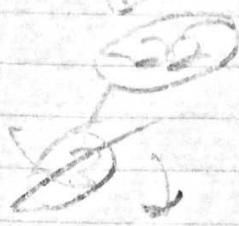
↓
p2



new heapify [cyl] ✓

delete root

T heap = [-3 | -1 | 0 | 10 | 1 | 17 | 2 | 87]



new heapify [cyl] ✓

delete root



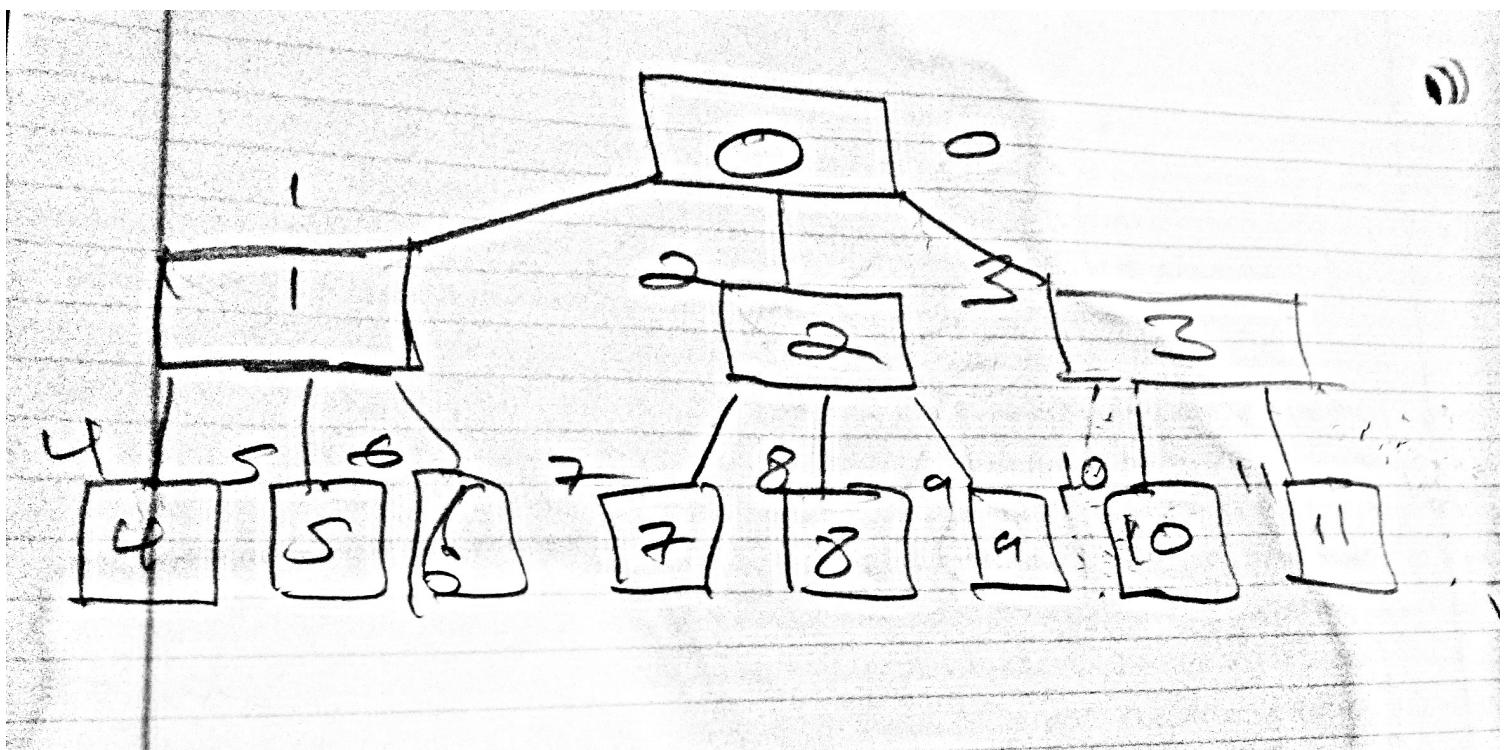
T = -22 | -3 | -1 | 0 | 1 | 10 | 1 | 17 | 2 | 87

resulting array

-22	-3	-1	0	1	10	1	17	2	87
-----	----	----	---	---	----	---	----	---	----

- 9.** Suppose that instead of binary heaps, we wanted to work with ternary heaps. Suggest an appropriate indexing scheme so that a complete tree will yield a contiguous sequence.

$$\begin{aligned}left_child &= 3 \cdot i + 1 \\middle_child &= 3 \cdot i + 2 \\right_child &= 3 \cdot i + 3 \\parent &= \lfloor \frac{i - 1}{3} \rfloor\end{aligned}$$



10. Use induction to prove that $1 + 2 + 4 + \dots + 2^h = 2^{h+1} - 1$.

Proof. We prove by induction

Let $P(h) = 1 + 2 + 4 + \dots + 2^h = 2^{h+1} - 1$

Base case: $h = 0$

$$1 = 1 = 2^{0+1} - 1.$$

Thus $P(0)$ holds.

Inductive step: Let $P(k)$ be true we show that $P(k + 1)$ is true, that is $1 + 2 + 4 + \dots + 2^k + 2^{k+1} = 2^{(k+1)+1} - 1$

$$\begin{aligned} 1 + 2 + 4 + \dots + 2^k + 2^{k+1} &= \sum_{i=0}^{k+1} 2^i \\ &= \sum_{i=0}^k 2^i + 2^{k+1} \\ &= 2^{k+1} - 1 + 2^{k+1} \\ &= 2 \cdot 2^{k+1} - 1 \\ &= 2^{k+1+1} - 1 \\ &= 2^{(k+1)+1} - 1 \end{aligned}$$

Thus $P(k + 1)$ holds. By the principle of mathematical induction, $P(h)$ holds for all integers $h \geq 0$. \square

11. What is the minimum and maximum number of leaves in a binary heap that has height h . Explain.

When $h = 0$ then the minimum equal the maximum namely 1. Suppose that $h \geq 1$. The minimum case is where there is only one left child in the level h . The rest of the leaves are on the $h-1$ level where the leftmost node is the only parent. Thus there are a total of $2^{h-1} - 1 + 1 = 2^{h-1}$ minimum number of leafs for height h . In a perfect tree where all levels are filled we have $\sum_{i=0}^h 2^i = 2^{h+1} - 1$ total nodes where the leafs are the last summation term and thus contribute 2^h nodes. Thus the minimum and maximum number of leaves in a binary heap that has height h is 2^{h-1} and 2^h leaves for $h \geq 1$.

12. Prove that a binary heap with n elements has height $\lfloor \log_2(n) \rfloor$.

$$\begin{aligned} 2^h &\leq n && \leq 2^{h+1} - 1 \\ 2^h &\leq n && < 2^{h+1} \\ \log_2(2^h) &\leq \log_2(n) && < \log_2(2^{h+1}) \\ h &\leq \log_2(n) && < h + 1 \end{aligned}$$

which by definition of the floor $h = \lfloor \log_2(n) \rfloor$. Hence we conclude a binary heap with n elements has height $\lfloor \log_2(n) \rfloor$.

13. Prove that a binary heap with n nodes has exactly $\lceil \frac{n}{2} \rceil$ leaves.

Version 1:

The n th element of the heap is the last node. If the n th element is a left child then it can be expressed as $n = 2 \cdot i$ where i is the index value of the parent node. solving for i we obtain $\frac{n}{2} = \lfloor \frac{n}{2} \rfloor = i$. If the n th element is a right child then it can be expressed as $n = 2 \cdot i + 1$. Solving for i we obtain $\frac{n-1}{2} = \lfloor \frac{n}{2} \rfloor = i$. We conclude the parent index of the last node can be expressed as $\lfloor \frac{n}{2} \rfloor$. By definition the parent is also the last internal node and thus the last internal node has index $\lfloor \frac{n}{2} \rfloor$. n can be expressed as the sum of $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$ which when solved for $\lceil \frac{n}{2} \rceil$ gives

$$\begin{aligned} n &= \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil \\ n - \lfloor \frac{n}{2} \rfloor &= \lceil \frac{n}{2} \rceil. \end{aligned}$$

Hence the number of elements n subtracted by the last internal node index tells us the amount of leaves in the heap. We conclude that binary heap with n nodes has exactly $\lceil \frac{n}{2} \rceil$ leaves.

Version 2:

We prove by induction. Let $P(n)$ be the statement that a binary heap with n nodes has exactly $\lceil \frac{n}{2} \rceil$ leaves. $P(1)$ holds because $\lceil \frac{1}{2} \rceil = 1$ leaf, which is namely the root. We assume $P(k)$ holds for some integer k and show that it must hold for the $k+1$ integer. Case 1: the $\lfloor \frac{k}{2} \rfloor$ internal node has one child, then the next node will be its right child.

$$\begin{aligned} \lceil \frac{k}{2} \rceil + 1 &= \lceil \frac{k+2}{2} \rceil \\ &= \frac{k+1}{2} \quad (\text{since } k+2 \text{ is odd}) \\ &= \lceil \frac{k+1}{2} \rceil \end{aligned}$$

which proves the 1st case.

Case 2: The $\lfloor \frac{k}{2} \rfloor$ internal node has two children, then the next node will be the left child of the $\lfloor \frac{k+1}{2} \rfloor$. Prior to the insertion, by the inductive hypothesis the binary heap has $\lceil \frac{k}{2} \rceil$ leaves. Adding the $k+1$ node will add one leaf but will make the parent of the node no longer a leaf so the tree still has $\lceil \frac{k}{2} \rceil$ leaves. Since k is odd this can be expressed as $\frac{k+1}{2}$ which is equivalent to $\lceil \frac{k+1}{2} \rceil$ which proves the 2nd case.

Thus $P(k+1)$ holds for both cases. By the principle of mathematical induction, $P(n)$ holds for all integers $n \geq 1$.

14. Show that there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes of height h in any n -element heap.

We prove by induction. Let $P(h)$ be the statement that there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes of height h in any n -element heap. $P(0)$ holds since $\lceil \frac{n}{2^1} \rceil$ are the number of leaves in the heap which by definition have height 0. We assume $P(k)$ holds for some integer k and show that it must hold for the $k + 1$ integer. We have that there at most $\lceil \frac{n}{2^{k+1}} \rceil$ nodes of height k in any n -element heap. We make a new heap H' which has all the elements of the original heap H except the leaves, hence it has $\lfloor \frac{n}{2} \rfloor$ total elements. The nodes of height k in this heap will be of height $k + 1$ in the original heap. $\lceil \frac{\lfloor \frac{n}{2} \rfloor}{2^{k+1}} \rceil \leq \lceil \frac{\frac{n}{2}}{2^{k+1}} \rceil = \lceil \frac{n}{2^{(k+1)+1}} \rceil$. Hence $P(k + 1)$ holds. By the principle of mathematical induction, $P(h)$ holds for all integers $h \geq 0$.