Homework - 5 Design & Analysis of Algorithms

Pranav Umakant Pujar 1001965075

6.1-1) Minimum number of elements in a heap of height $h = \lceil 2^h \rceil$

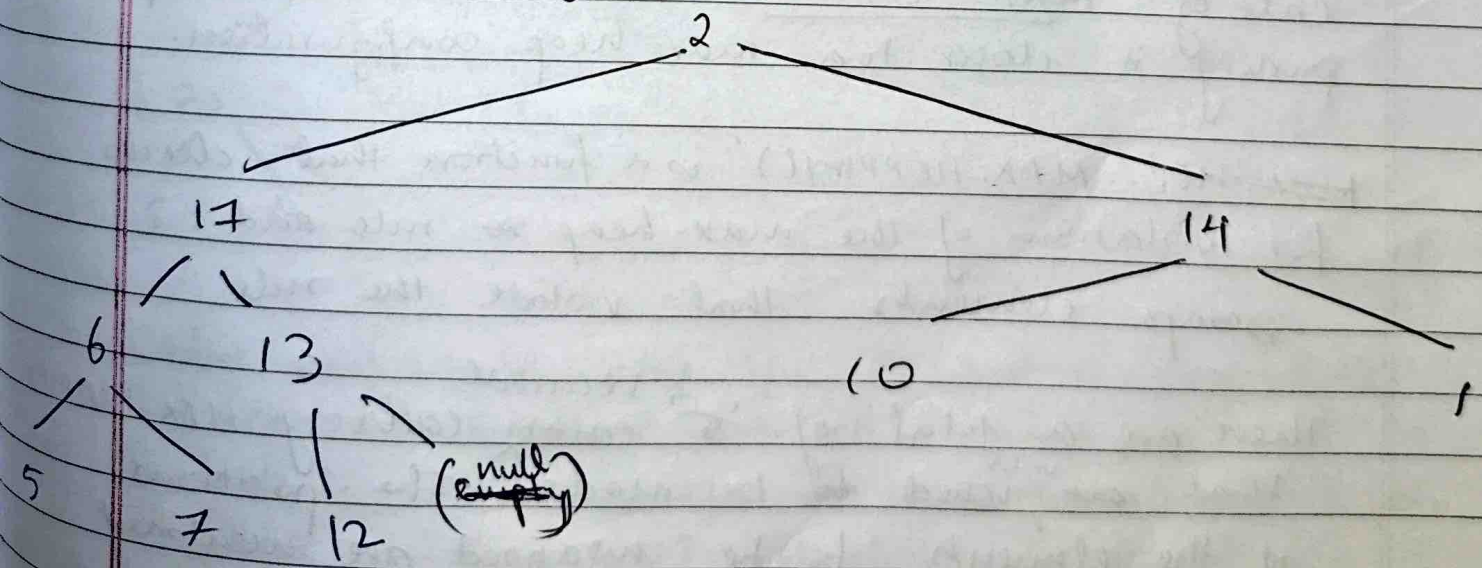
Max. number of " " of height $h = \lceil 2^{h+1} - 1 \rceil$

6.1-4) In a max heap, the smallest element can only reside in one (or many) of the ~~be~~ leaf nodes.

6.1-5) Yes, an array that is sorted in order is indeed a min-heap (if sorted in ascending order).

This is because the array representation of a min-heap, ~~where~~ where parent elements are lesser than or equal to their children, will look like an array in ascending order.

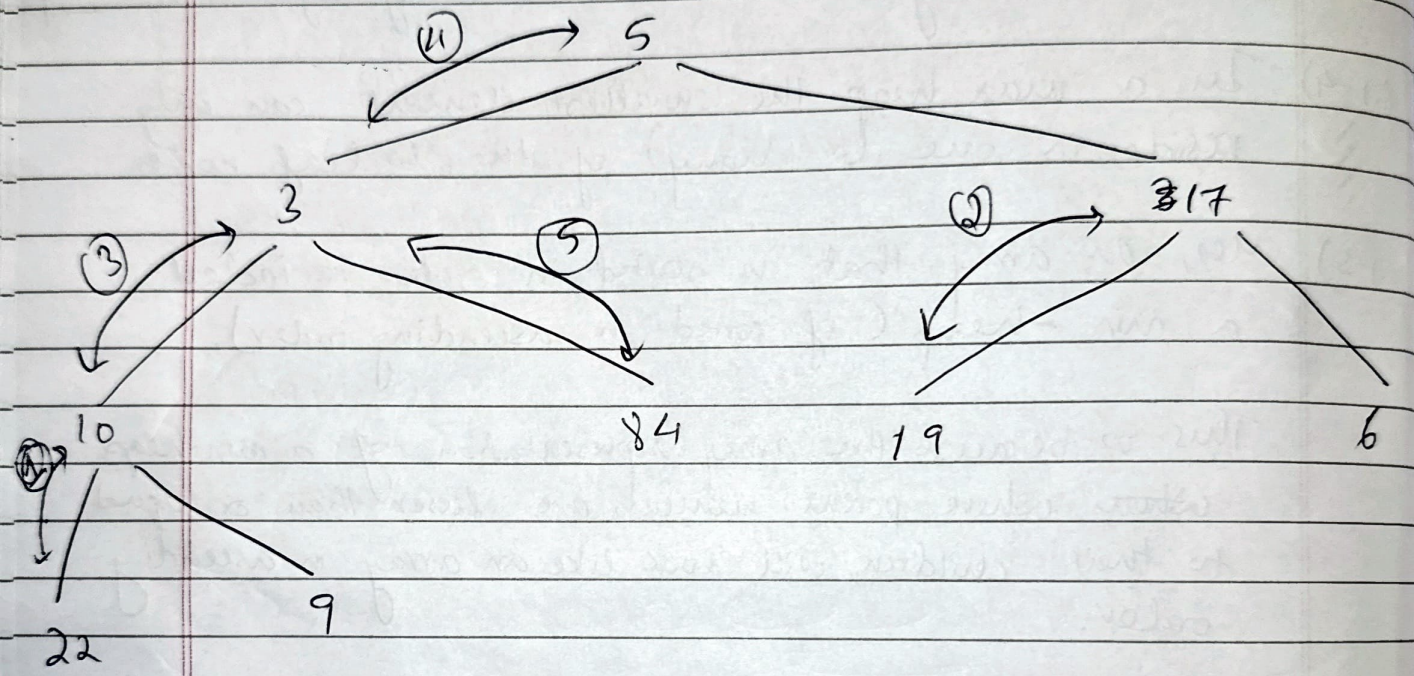
6.1-6) No, this array is not a max heap.



In the above ~~tree~~ heap representation of the given array, we can see that the ~~max heap~~ max-heap property

is not maintained for node with value '6'. It has a ~~child~~ child node with value '7', which is greater than 6.

6.3-1) Initial heap provided:-



I have listed out the order in which every call of 'MAX-HEAPIFY()' restructures the heap, pushing it closer to a max-heap configuration.

~~MAX-HEP~~ 'MAX-HEAPIFY()' is a function that checks for violations of the max-heap ~~so~~ rule and ~~swaps~~ swaps elements that violate the rule.

There are a total of '5' recursive calls of MAX-HEAPIFY that ~~are~~ will be made. The positions of the elements to be swapped are mentioned for each call in the above diagram. Each call number is circled. Eg, in call number 5,

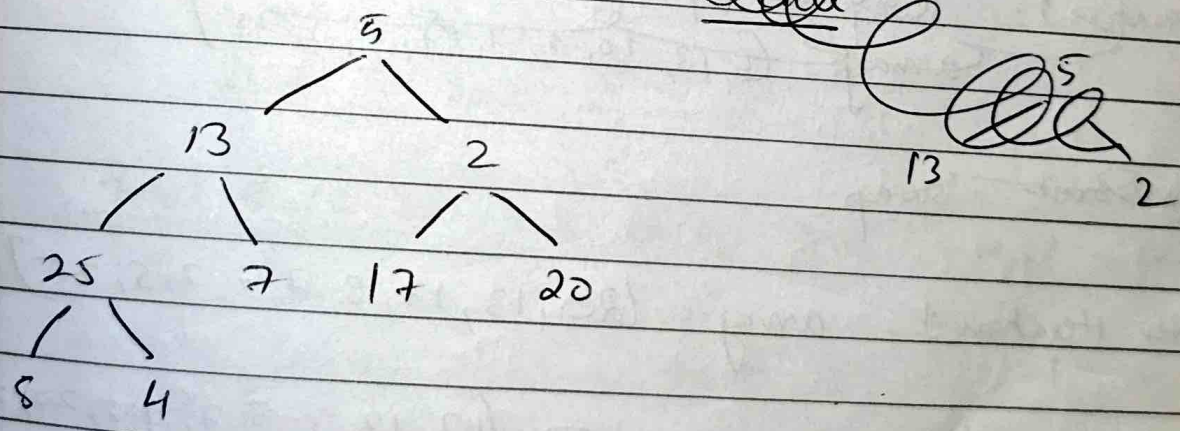
element at index 1 and index 4 (0-indexed) will be swapped, not the literal numbers 3 and 84, as there will be different numbers there by that time (5 and 84).

6.4-1) ~~I will delete~~

The final heap will look like this ~~in~~ in array form: ~~[84, 22, 19, 10, 5, 17, 6, 3, 9]~~
[84, 22, 19, 10, 5, 17, 6, 3, 9]

6.4-1) I will show the steps in the heap-sorting process. Every step after the initial ~~1st~~ stage will show how the heap looks after 4 extra calls of ~~heapsort~~ 'MAX-HEAPIFY(1)' to sort the heap in ascending order.

initial



~~after~~ call 1: No swap b/w 25 and children {8, 4}
call 2: Swap 2 with child node 20
call 3: No swap b/w '2' and nonexistent children
call 4: Swap 13 with child node 25
call 5: No swap b/w 13 and its children
call 6: Swap '5' with its child '25'

call 7: Swap node '5' with its child '13'.

call 8: Swap node '5' with its child '8'.

Now, the max Heap will look like this:-

array = [25, 13, 20, 8, 7, 17, 2, 5, 4]

It is still not fully sorted, as ~~this is the~~ only the first iteration ~~in the~~ of the recursive max-heapify() function.

Now we have to extract the elements from the heap and place them in their respective ~~buckets (to correct positions)~~ positions.

~~iteration 1: Swap root w/ last element~~
~~array: [4, 13, 20, 8, 7, 17, 2, 5, 25]~~

~~iteration 2: swap~~

after iteration 1: array = [20, 13, 17, 8, 7, 4, 2, 5, 25]

after iteration 2: array = [17, 13, 5, 8, 7, 4, 2, 20, 25]

after iteration 3: array = [13, 8, 5, 2, 7, 4, 17, 20, 25]

after iteration 4: array = [8, 4, 5, 2, 7, 13, 17, 20, 25]

after iteration 5: array = [7, 4, 5, 2, 8, 13, 17, 20, 25]

after iteration 6.. array = [5, 4, 2, 7, 8, 13, 17, 20, 25]

after iteration 7: array = [4, 2, 5, 7, 8, 13, 17, 20, 25]

final sorted array = [2, 4, 5, 7, 8, 13, 17, 20, 25]

1) Prove height of heap of n-element = $\Theta(\log n)$

~~Question~~ We need to derive the formula for number of nodes per level.

~~It~~ in a heap:-

level 0 has 1 node (root)
level 1 has 2 nodes
level 2 has 4 nodes
level 3 has 8 nodes

level 'i' has '2^i' nodes

Total # of nodes in a binary tree (heap) = $1+2+3+\dots+2^h$

$$= \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \approx 2^{h+1}$$

We have ~~proven~~ derived that [total # of nodes $\approx 2^{h+1}$]

where 'h' is height of heap.

now, $n \approx 2^{h+1}$ = total # of nodes in heap

$\Rightarrow \log_2(n) \approx h+1$

$h \approx \log_2(n) - 1$

$\Rightarrow h \approx \log_2(n)$

$\rightarrow \boxed{h = \Theta(\log(n))}$

Hence proved, height of heap 'n' element heap is $\Theta(\log(n))$

2) 2.a) A stable sorting algorithm maintains the relative order of elements that are equal in value. Heapsort does not guarantee this. It is possible for equal values in heapsort to change their relative ordering, due to the swapping that occurs multiple times

2.b) Yes, Bubble Sort is a stable sorting algorithm. This is because the swapping occurs only if an element in the left is greater than the element in the right, no swapping occurs if the two elements are equal. This preserves their relative order.

2.c) In situations where multiple sorting criteria are involved, using sorting algorithms that are not stable can pose a problem. We need stable sorting algorithms in this case. Sometimes the relative ordering of elements with same value holds meaning due to the other attributes those elements might have. For correctness, we must try to always use stable sorting algorithms.