CISC 3220 Homework Chapter 6

Rachel Friedman

April 19, 2020

Exercises 6.1

Question 6.1-1

What are the minimum and maximum numbers of elements in a heap of height h?

A heap is a balanced binary tree where all levels except the last one is completely full. Therefore, the minimum amount of elements is: 2^h and the maximum amount is: $2^{h+1} - 1$.

Question 6.1-2

Show that an n element heap has height $\lfloor \lg n \rfloor$.

From the solution above, we know that the minimum amount of elements in a tree is 2^h and the maximum amount of elements is $2^{h+1} - 1$. Obviously, the minimum amount is less than or equal to the amount of elements which is in turn less than or equal to the maximum amount of elements. Thus:

$$
\begin{aligned}
2^h \leq n \leq 2^{h+1} - 1 & \\
h \leq \log n \leq h + 1 & \qquad \text{take log on all sides} \\
h = \lfloor \lg n \rfloor & \qquad \text{since } h \text{ is an integer}
\end{aligned}
$$

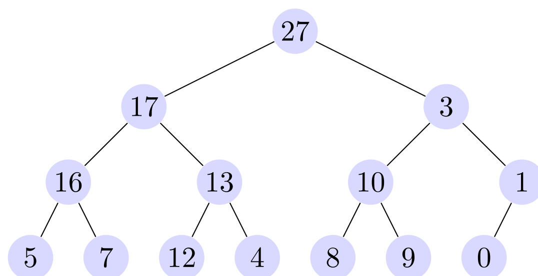
Question 6.1-6

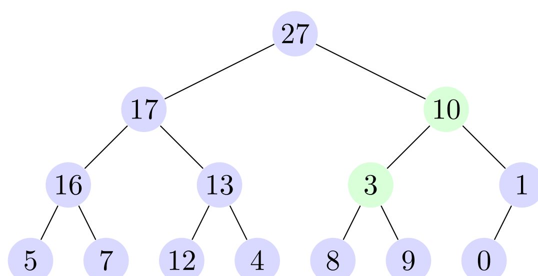
The given array is not a max heap since the subtree with 6 as its root and 7 as its left child violates the max-heap property.

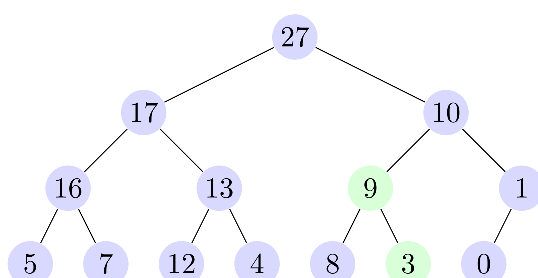
Exercises 6.2

Question 6.2-1

Illustrate **MAX-HEAPIFY(A, 3)** on the array $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$. (Note to self: top down. Running time: $\mathcal{O}(\lg n)$)







Question 6.2-3

What is the effect of calling MAX-HEAPIFY(A, i) when the element $A[i]$ is larger than its children?

The heap is unchanged.

Exercises 6.3

Question 6.3-1

Illustrate **BUILD-MAX-HEAP** on the array $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.
(Note to self: bottom up. Running time is $\mathcal{O}(n)$.)

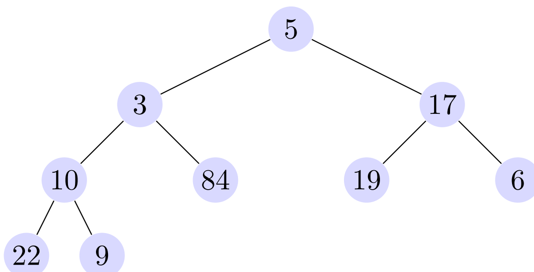


Figure 1: (a)

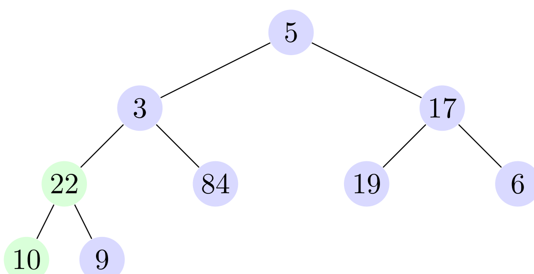


Figure 2: (b)

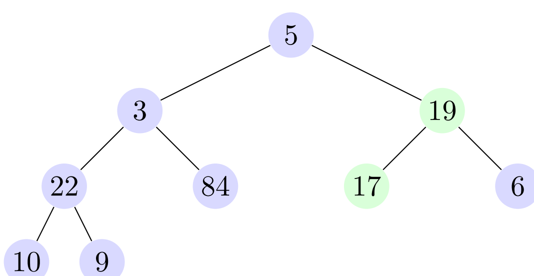


Figure 3: (c)

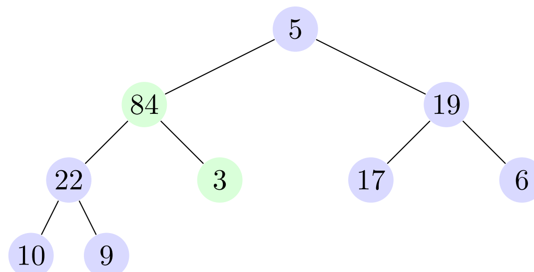


Figure 4: (d)

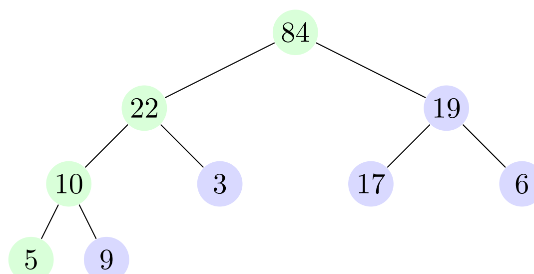


Figure 5: (e)

The intuition behind $T(n) = \mathcal{O}(n)$ is that half of the nodes have only leaves as subtrees, so Heapify takes less time on those nodes. As you go higher up in the tree, the subtrees get larger, so heapify takes longer to run, but there are less of those nodes. This is a geometric series which converges to a constant. Drop the constant for asymptotic analysis and all that's left is n. Mathematically, here's what that looks like:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h)$$

Exercises 6.4

Question 6.4-1

Illustrate **HEAPSORT** on the array $\langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.
(Running time: $\mathcal{O}(n \lg n)$)

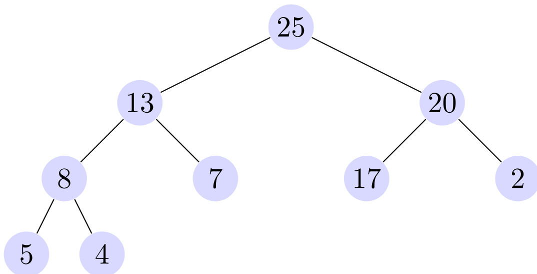


Figure 6: (a) After BUILD MAX-HEAP

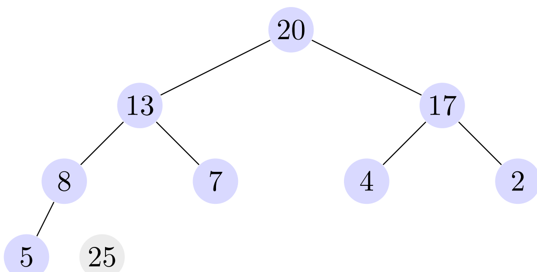


Figure 7: (b)

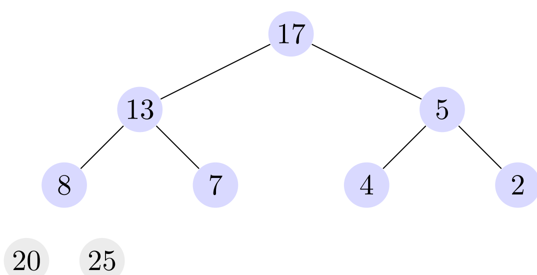


Figure 8: (c)

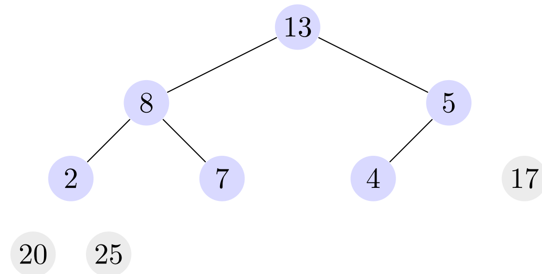


Figure 9: (d)

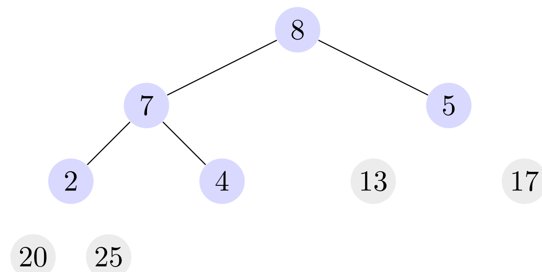


Figure 10: (e)

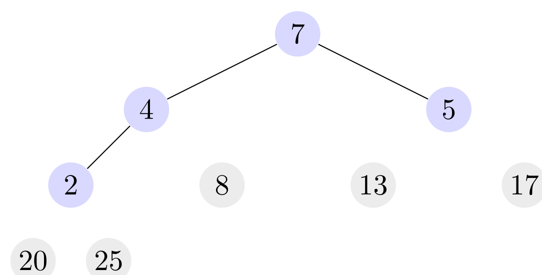


Figure 11: (f)

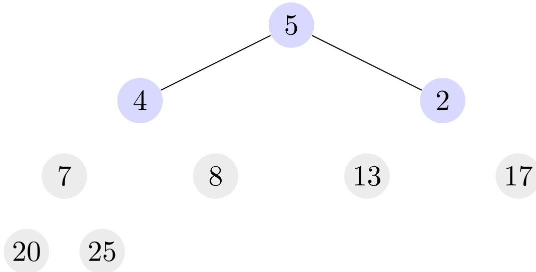


Figure 12: (g)

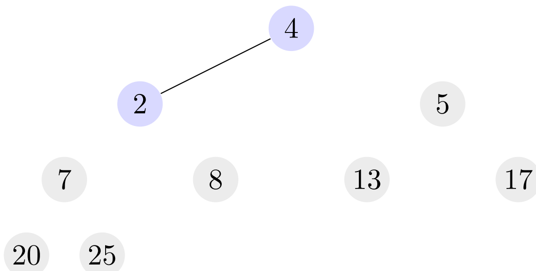


Figure 13: (h)

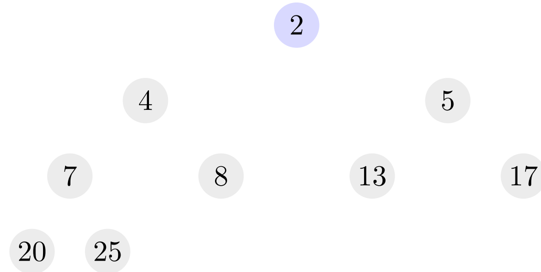


Figure 14: (i)

2	4	5	7	8	13	17	20	25

Figure 15: (j)

Question 6.4-3

What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

Even though it's sorted, the first line of HEAPSORT calls BUILD-MAX-HEAP and we know the running time for that is $\mathcal{O}(n)$. It then calls MAX-HEAPIFY for all the nodes, and the running time for MAX-HEAPIFY is $\mathcal{O}(\lg n)$. So the total running time for Heapsort, regardless of the order of the array, is $\mathcal{O}(n \lg n)$.