Jackie Diep
CSC 413 Algorithms
Assignment #3
4/2/2021

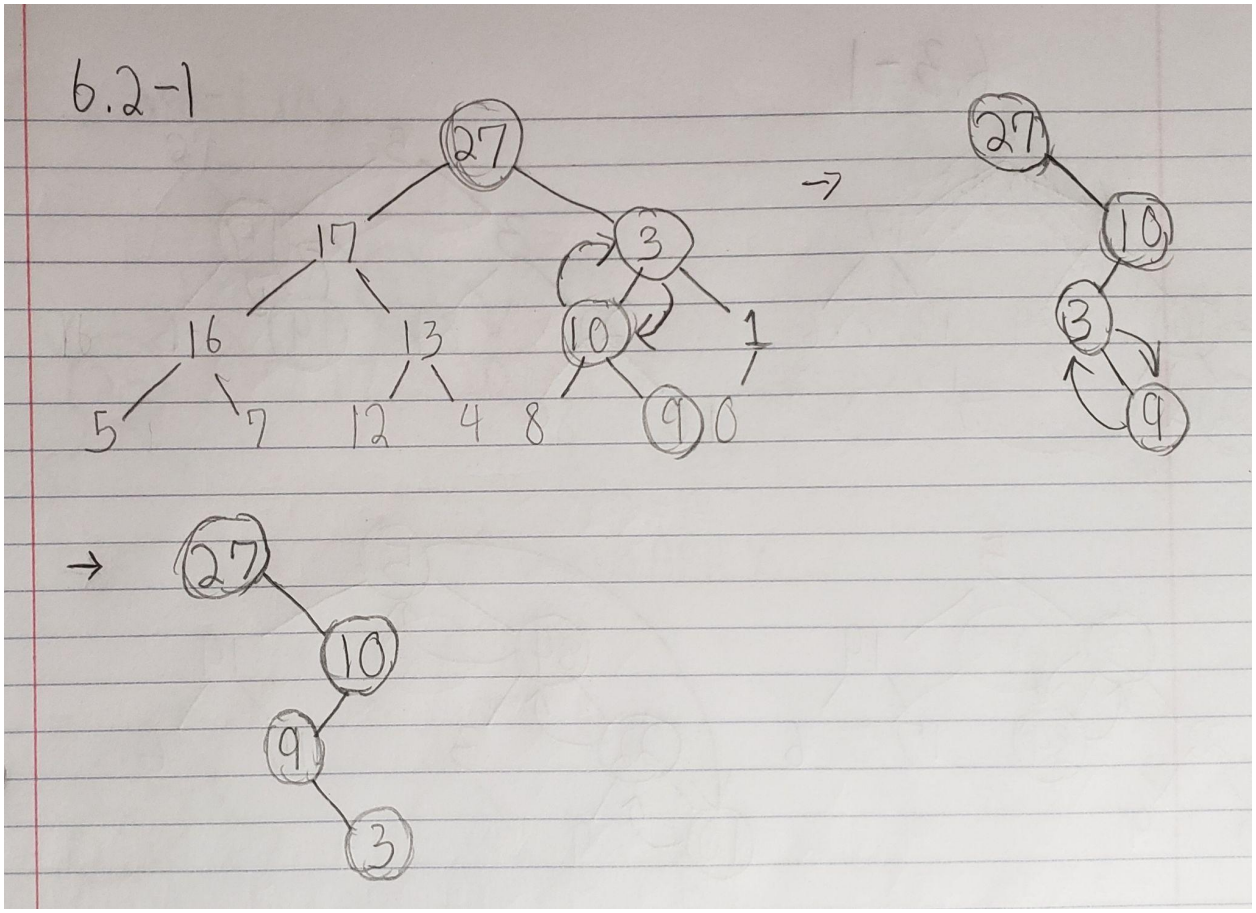
- **6.1-2 - Show that an n-element heap has height h = ⌊lgn⌋.**

Best case: lg(n + 1) - 1, when the entire tree is filled
Worst case: lg n, when the bottom level of the tree has only one element
For the tree to exist, it must be at least 2^h when h = 0, and $2^0 = 1$. Also, $lg(2^h) = h$.
So, $h \leq lg(n) < lg(h + 1) - 1$
s.t. the floor function of lg(n) = h
$h = \lfloor lgn \rfloor$

- **6.1-6 - Is the array with values <23, 17, 14, 6, 13, 10, 1, 5, 7, 12> a max-heap?**

No, because when expanded upon, 6 is a parent node of 7, yet 7 is greater than 6.

- **6.2-1 - Using Figure 6.2 as a model, illustrate the operation of MAX-HEAPIFY(A, 3) on the array A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>.**

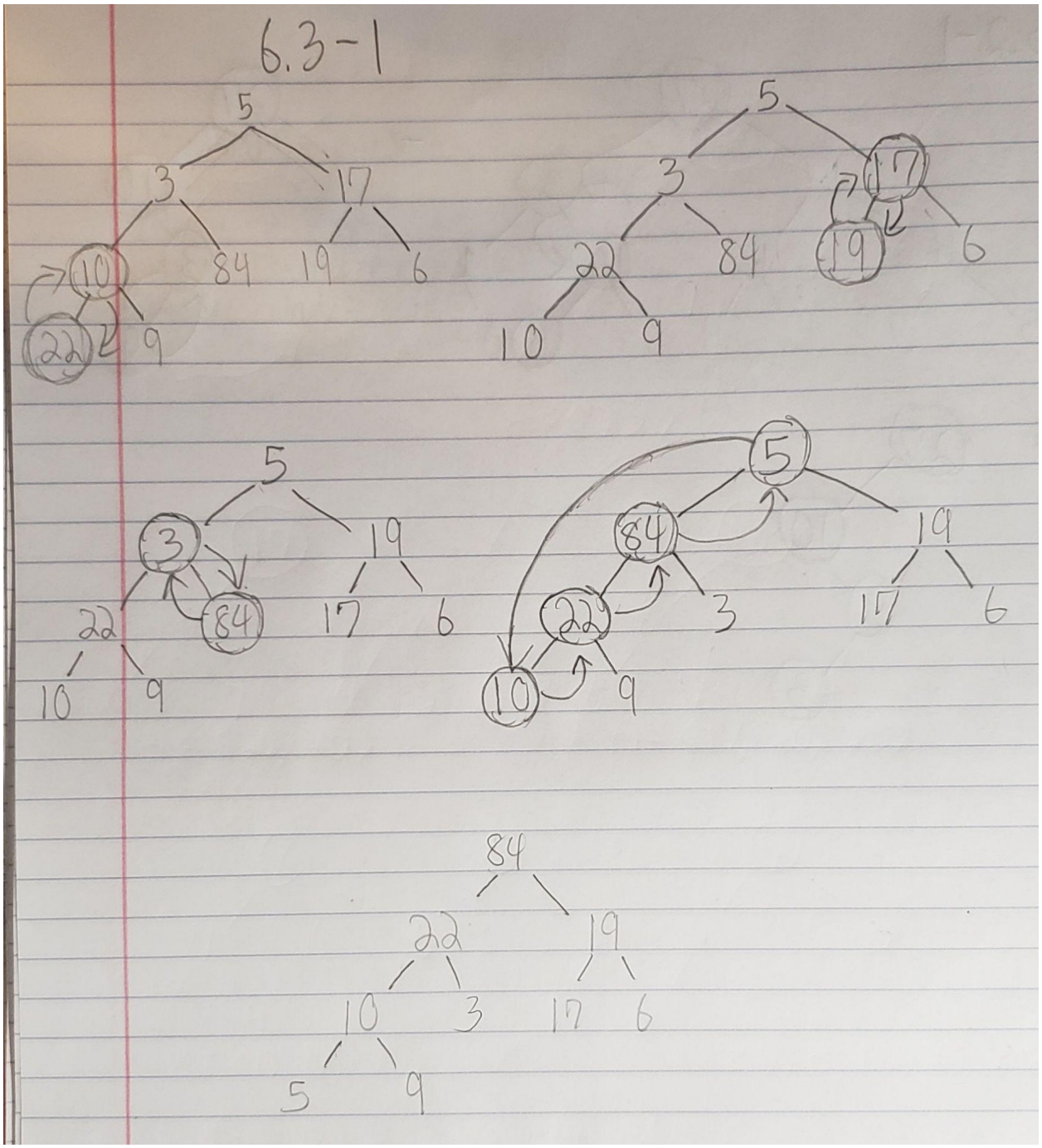


- **6.2-2 - Starting with the procedure MAX-HEAPIFY, write pseudocode for the procedure MIN-HEAPIFY(A, i), which performs the corresponding manipulation on a min-heap. How does the running time of MIN-HEAPIFY compare to that of MAX-HEAPIFY?**

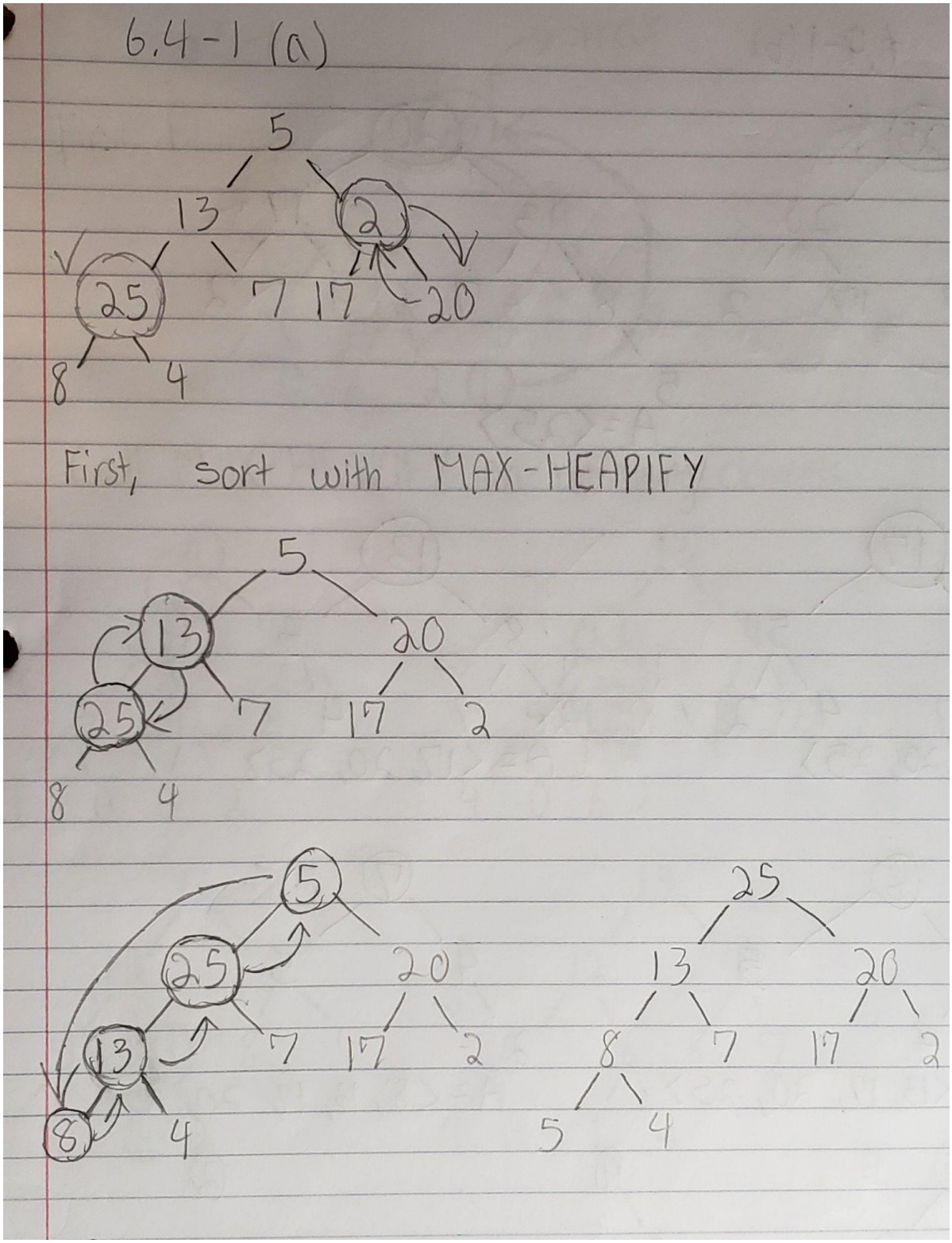
```
MIN-HEAPIFY(A, i)
{
l = LEFT(i)
r = RIGHT(i)
if (l ≤ A.heap-size and A[l] < A[i]
        smallest = l
else    smallest = i
if (r ≤ A.heap-size and A[r] < A[i]
        smallest = r
else    smallest = i
if smallest ≠ i
        swap(A[i], A[smallest])
        MIN-HEAPIFY(A, smallest)
}
```

The running time of MIN-HEAPIFY should be the same as MAX-HEAPIFY with this version.

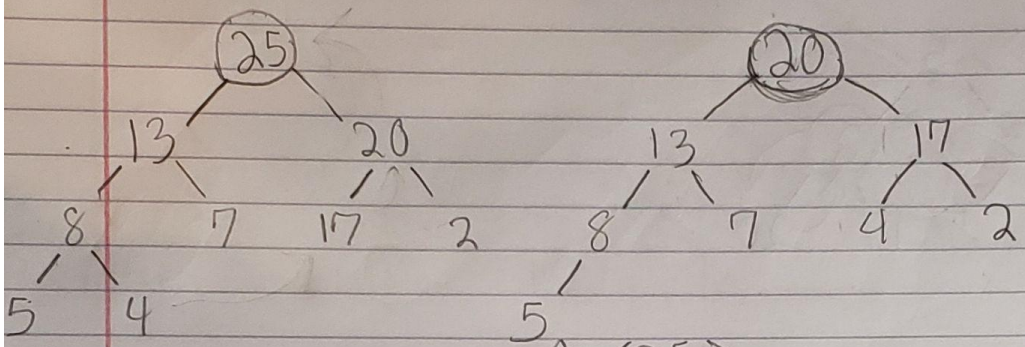
- **6.3-1 - Using Figure 6.3 as a model, illustrate the operation of BUILD-MAX-HEAP on the array A = <5, 3, 17, 10, 84, 19, 6, 22, 9>.**



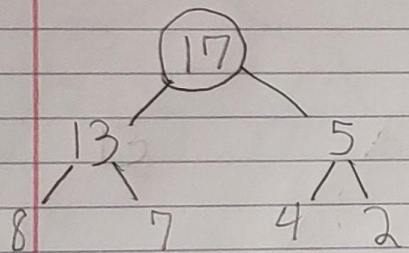
- **6.4-1 - Using Figure 6.4 as a model, illustrate the operation of HEAPSORT on the array A = <5, 13, 2, 25, 7, 17, 20, 8, 4>.**



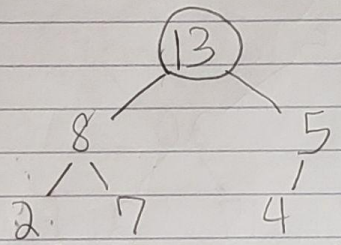
6.4-1(b)



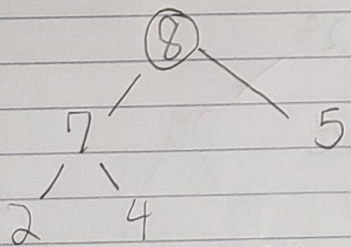
A = ⟨25⟩



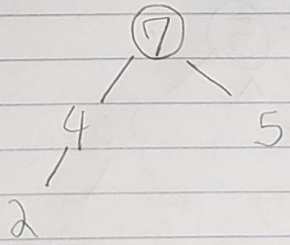
A = ⟨20, 25⟩



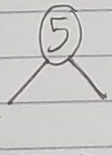
A = ⟨17, 20, 25⟩



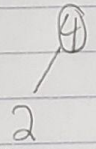
A = ⟨13, 17, 20, 25⟩



A = ⟨8, 13, 17, 20, 25⟩



A = ⟨7, 8, 13, 17, 20, 25⟩



A = ⟨5, 7, 8, 13, 17, 20, 25⟩

A = ⟨4, 5, 7, 8, 13, 17, 20

A = ⟨2, 4, 5, 7, 8, 13, 17, 20, 25⟩

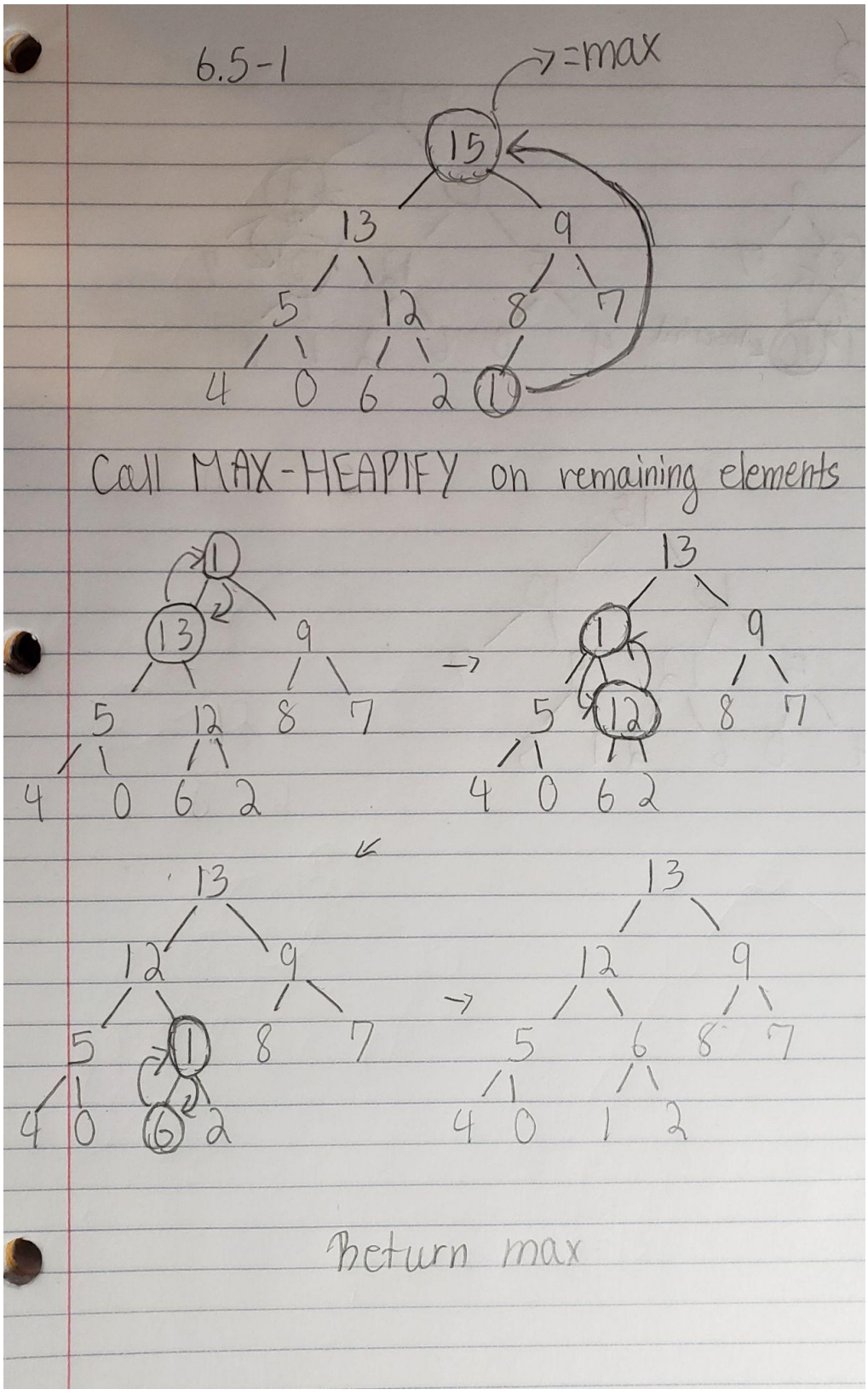
- **6.4-2 - Argue the correctness of HEAPSORT using the following loop invariant: At the start of each iteration of the for loop of lines 2–5, the subarray A[1..i] is a max-heap containing the i smallest elements of A[1..n], and the subarray A[i + 1..n] contains the n - i largest elements of A[1..n], sorted.**

Initialization: The loop invariant holds true because the subarray A[i + 1..n] is empty such that it holds the largest elements of n - i sorted because i = n.

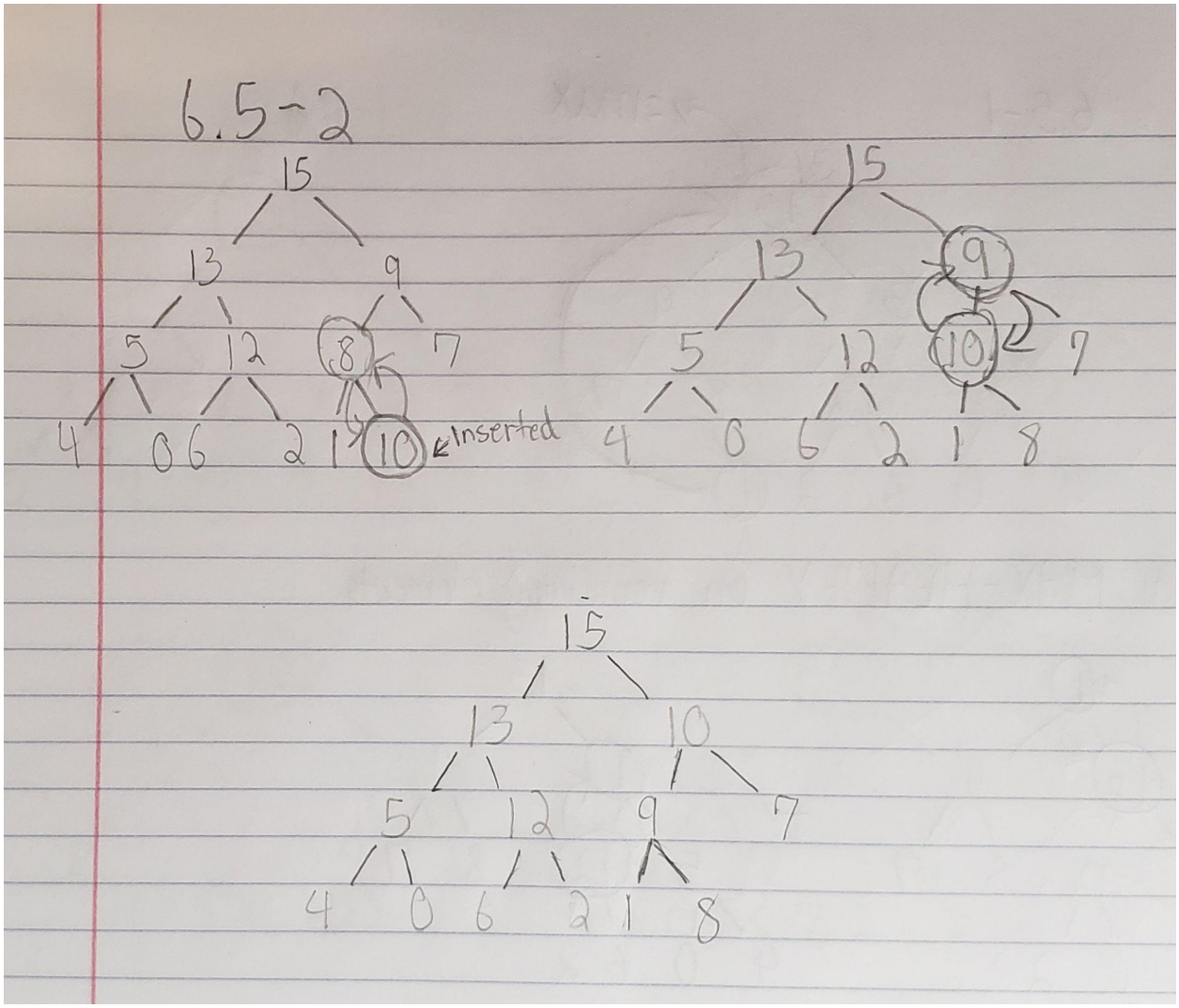
Maintenance: During iterations, the element of A[1] is always the largest element of the max-heap subarray A[1..i]. After each iteration, the element in A[1] is added to the subarray in A[i + 1..n] at position A[i], such that it is the smallest element in the sorted subarray consisting of the largest elements of n - i. Calling max-heap on the remaining elements of A[1..i - 1] prepares for the following iteration.

Termination: After the last iteration of the for loop, there is only one element remaining in A, the smallest element from the original subarray. This means that every element of A[2..n] is sorted in ascending order, and A[1] is guaranteed to be the smallest number of A. The final result of the algorithm in a sorted array A[1..n].

- **6.5-1 - Illustrate the operation of HEAP-EXTRACT-MAX on the heap A = <15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1>.**



- **6.5-2 - Illustrate the operation of MAX-HEAP-INSERT (A, 10) on the heap A = <15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1>.**



- **6.5-7 - Show how to implement a first-in, first-out queue with a priority queue. Show how to implement a stack with a priority queue. (Queues and stacks are defined in Section 10.1.)**

With a FIFO queue, we would implement a MIN priority queue with an incrementing counter assigned to each element. This way, elements that enter first have a lower counter value assigned to them, and the MIN priority queue would put them first.

With a stack, we would implement a MAX priority queue with an incrementing counter assigned to each element as stacks are LIFO. This way, elements that enter last have the highest counter value, and the MAX priority queue would put them first.