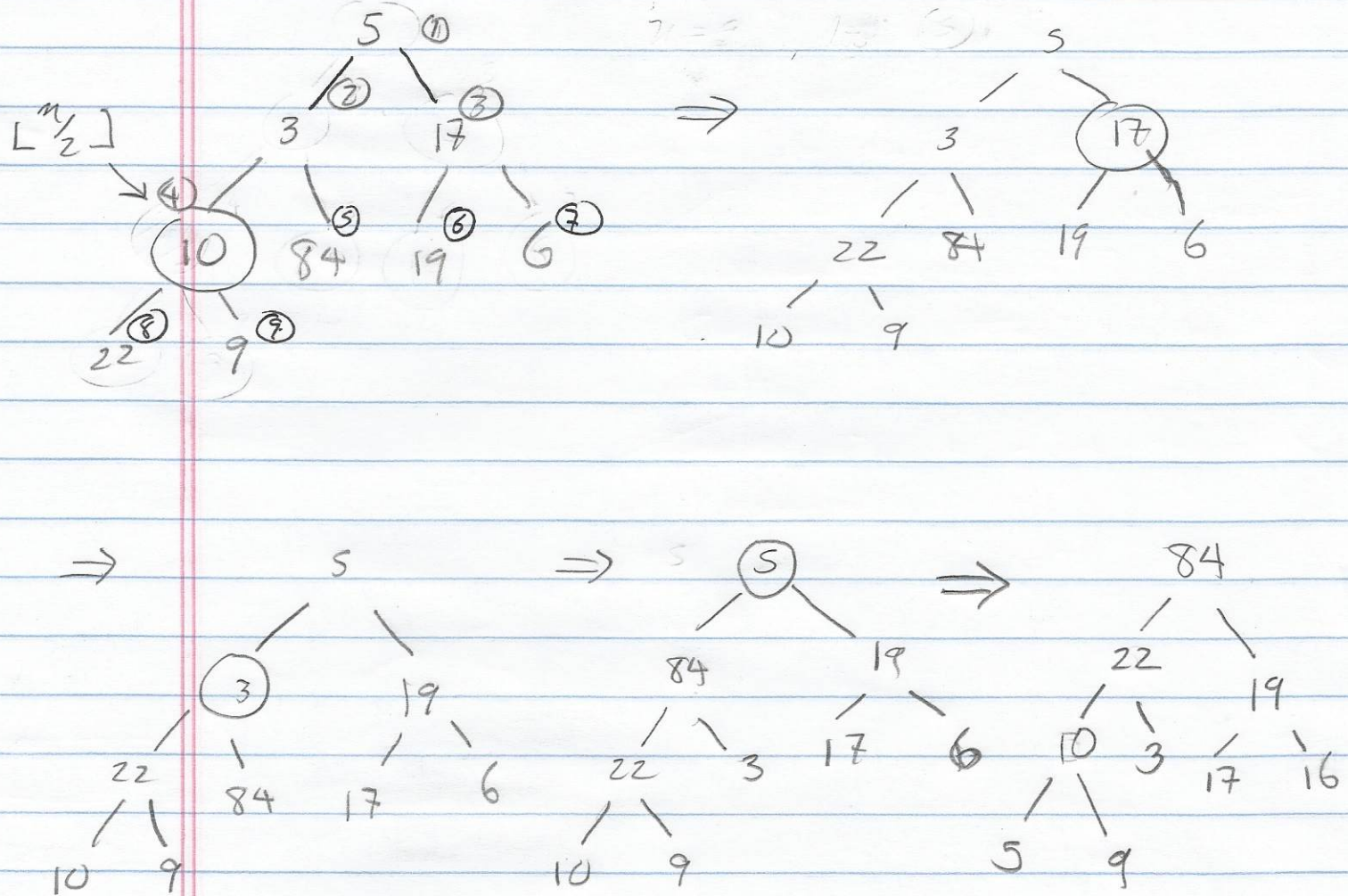
COMP 3270 HW#3 SOLUTION

6.3-1) $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$



6-3.2) Because otherwise largest element will not end up at the root

6.5-8) Implement HEAP-DELETE(A, i) in $O(\lg n)$ time for n-element max heap

```
HEAP-DELETE(A, i)
    // replace A[i] with -∞
    A[i] = -∞
    MAX-HEAPIFY(A, i)
    remove A[A.length]
    A.length--
```

7.1-3) Running time of PARTITION is $\Theta(n)$ if $p - r + 1 = n$

lines 1 & 2 take constant time: $\Theta(1)$
loop in lines 3-6 executes $r - 1 - p$ times
or about n times
each iteration takes $\Theta(1)$ time
lines 7-8 take constant time

hence total time is $\Theta(1) + \Theta(n) \cdot \Theta(1)$
$= \Theta(n)$

7.4-2) Best-case running time of quicksort is $\Omega(n \lg n)$

We know the running time for QUICKSORT is

$$T(n) = \Theta(n) + T(k) + T(n-k-1)$$

The best case will occur when balanced

$$T(n) = 2T(n/2) + \Theta(n)$$

Master theorem yields

$$T(n) = \Theta(n \log n)$$

Hence $T(n) = \Omega(n \log n)$ by defn of Θ

8.1-1) Smallest possible depth of a leaf for a comparison sort would be $n-1$ – that is how many elements it would have to compare to a single element to determine how many are greater/less than that element

8.3-2) Stable sorts

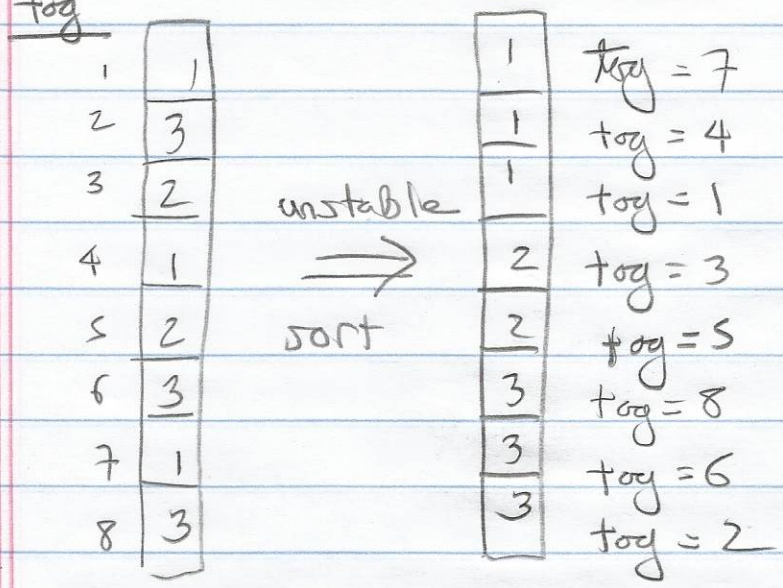
insertion – preserves order

merge – preserves order

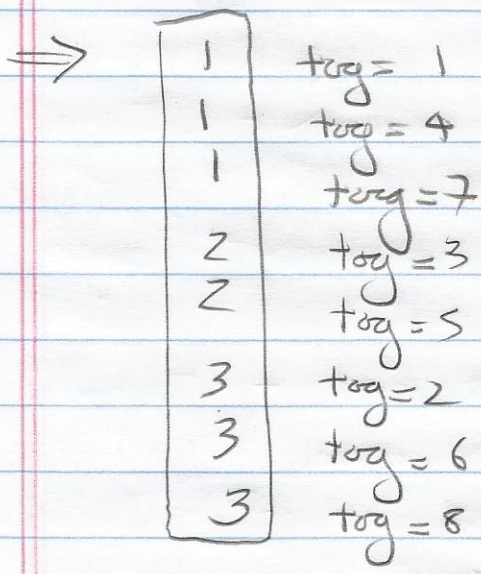
heapsort – order from original array destroyed to preserve heap property

quicksort – no – duplicate order is disturbed by the partition

To restore stability, you could associate with each array element a tag telling its original index in the input array, for example, see the following diagram



Then, sort each group of duplicates using the unstable sort



Worst case would be when the whole list is duplicated gives 2x the space and 2x the time

Another method would be to use the tag to resolve ties in comparisons, also 2x space / 2x time

7.1 a)

1	2	3	4	5	6	7	8	9	10	11	12
13	19	9	5	12	8	7	4	11	2	6	21

H-PARTITION (A, 1, 12)

$x = 13$

$i = 0$

$j = 13$

After 1st iteration

$i = 1$

$j = 11$

1	2	3	4	5	6	7	8	9	10	11	12
6	19	9	5	12	8	7	4	11	2	13	21

↑ i ↑ j

After 2nd iteration

$i = 2$

$j = 10$

1	2	3	4	5	6	7	8	9	10	11	12
6	2	9	5	12	8	7	4	11	19	13	21

↑ i ↑ j

After 3rd iteration

$i = 10$, $j = 9$

1	2	3	4	5	6	7	8	9	10	11	12
6	2	9	5	12	8	7	4	11	19	13	21

$i \geq j$ so return 9

↑ j ↑ i