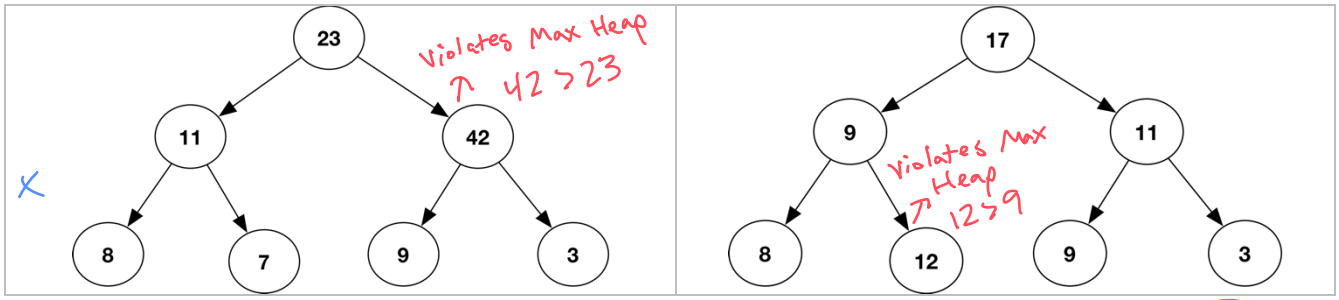
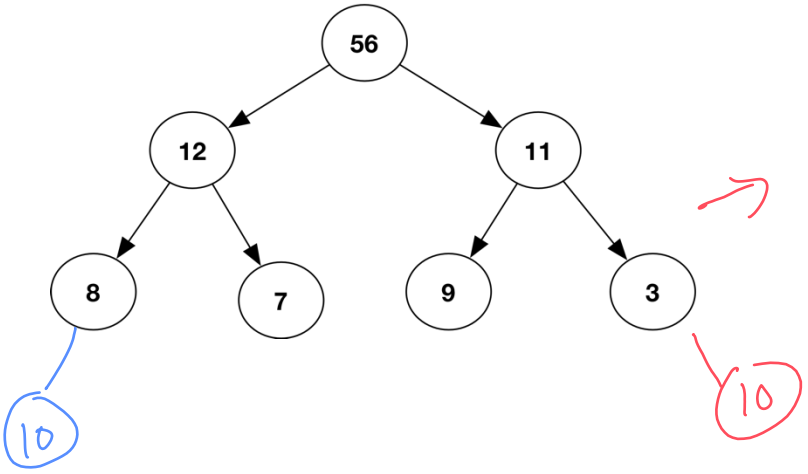
Problem Set 5 **CS3851** **Name** Hudson Arney

Submission instructions: Your answers should be submitted to Canvas as a PDF.

1. Validate that the given heaps satisfy the max heap property. If they violate the property, indicate where the property is violated and re-arrange the elements to restore the property.



2. Illustrate the steps of inserting the element 10 into the heap below.



3. Illustrate the steps of extracting maximum elements from the heap in question 2.

Take out root 56, 12 replaces as root

4. Partial sorting is the problem of finding the k smallest (or largest) elements in a collection of n items and sorting them. Partial sorting is preferable to sorting all elements in the collection because it can be accomplished more quickly. Remember that we have seen five sorting algorithms (insertion sort, merge sort, bubble sort, selection sort, and heapsort) so far in this class.

a) What is the lowest bound on the worst-case running time of the five sorting algorithms?

b) You can solve the partial sorting problem as follows: (1) construct a heap of the n items and (2) call the extract maximum (or minimum) operation k times. What is the worst-case running time?

c) A second algorithm from among the five algorithms we studied can be modified to perform partial sorting. Identify this algorithm.

d) Explain how to modify the algorithm from (c) to perform partial sorting.

e) Estimate the worst-case running time for partially sorting k items using the modifications you proposed in (d).

f) Which approach (heapsort, algorithm from (d)) is faster?

Build max heap : O(n)
Extracting Max element : O(1)
Re-heapify : O(logn)

a) Heap sort: O(nlogn)
Bubble sort: $O(n^2)$
Selection sort: $O(n^2)$
Merge sort: O(nlogn)
Insertion sort: $O(n^2)$

b) O(n + k*logn)
Initial construction of the heap takes O(n), and extracting the max takes O(logn), which gets called "k" times.

c) Merge Sort

d) Make sure to only merge up to k elements When you find one element that is smaller in than an element in another subarray increment the count of merges, and copy to the output. Repeat the recursive divide-and-conquer method.

n = input array size

e) Still O(k logn)

f) When comparing w/ Heapsort both have a worst case of O(klogn)

4c) Selection Sort, It does not have to sort the whole array before finding k smallest/largest

d) Iterate K times instead of n times. Rest of algorithm remains the same.

e) O(k*n)

f) Heapsort as O(klogn) is faster than O(k*n)