Problem 1. Consider an array of size nine with the numbers in the following order $40, 20, 80, 60, 30, 90, 10, 70, 50$.

(a) Form the heap using the algorithm described in class. Show the heap as a tree. Show the heap as an array. Exactly how many comparisons did heap creation use?

(b) Start with the heap created in Part (a). Show the *array* after each element sifts down *during the Finish phase*. How many comparisons does each sift use? What is the total number of comparisons *after heap creation*?

Problem 2. Assume that your machine has a built-in data structure for **one** priority queue, which can be either a min-queue or a max-queue. It has the following operations:

- Insert an element, which is either simply a value, or a pair (value,name).

```
insert(Q,value)
insert(Q,(value,name))
```

- Delete the smallest (largest) value in a min-queue (max-queue), and optionally assign it to a variable.

```
delete(Q)
value ← delete(Q)
(value,name) ← delete(Q)
```

- Peek at the smallest (largest) value in a min-queue (max-queue),

```
peek(Q)
```

- Determine how many elements are in the queue.

```
size(Q)
```

The insert and delete operations count as exactly one "comparison" (as do standard comparisons). The peek and size operations do not count as comparisons (unless the peek value is actually used in a comparison).

This machine can sort a list of size n with exactly $2n$ comparisons: Insert all of the elements into a max-queue and then remove them (one at a time). What if your priority queue is restricted to having size $s < n$?[1] If an item is inserted into a full min-queue (max-queue), the queue ends up with the s smallest (largest) values. Now how fast can you sort? Think about this before starting the problems below.

For the following problems, you may describe your algorithm in high level English or in pseudo code, but the description must be clear. Your analyses should be a function of the list size, n, and the queue size, s. Just get the exact high order term. Show your work.

(a) i. Design an "efficient" sorting algorithm based on a quadratic sorting algorithm (bubblesort, insertion sort, or selection sort).

ii. Analyze the number of comparisons. You may assume that n is "nice".

(b) i. Design an efficient sorting algorithm based on mergesort. Minimize the number of comparisons.

ii. Analyze the number of comparisons. You may assume that n is "nice".

[1] If s is small enough such a machine is actually quite buildable.

Problem 3. Consider the recurrence, from Problem 4b of Homework 4, for the number of comparisons in our recursive version of Modified Selection Sort. [You may want to wait until after Monday night to get the exact recurrence.]

(a) Use the tree method to solve the recurrence exactly assuming n is even.

(b) Use the tree method to solve the recurrence exactly assuming n is odd.