Problem 1. Consider a tree size nine with the numbers in the following order (left-to-right, top-to-bottom):

$$30, 40, 70, 60, 90, 20, 90, 50, 10.$$

We would like to run Heap Sort on the tree.

(a) Phase 1: Form the heap. Show the heap as a *tree*. Exactly how many comparisons did heap creation use? Make sure to form the heap bottom up as done in class. (Even if you happen recall Floyd's version of Heapify (i.e. Sift), do NOT use it.)

(b) Phase 2: Start with the heap created in Part (a). Show the heap as a *tree* after each Heapify (i.e. Sift) *after heap creation*. How many comparisons does each Heapify (i.e. Sift) do? What is the total number of comparisons *excluding heap creation*?

Problem 2. The tree for Heap Sort is normally stored in an array left-to-right, top-to-bottom. So the root is in $A[1]$, the left child of the root is in $A[2]$, the right child of the root is in $A[3]$, the left child of node 2 is in $A[4]$, the right child of node 2 is in $A[5]$, etc. In general, the left child of node i is in $A[2i]$, the right child is in $A[2i+1]$, and the parent is in $A[\lfloor \frac{i}{2} \rfloor]$.

We would like to take into account *memory hierarchies* when analyzing the Heapify (i.e. Sift) operation from Heap Sort. We will assume that $f(i)$ grows logarithmically (which is not an unreasonable assumption). To keep things simple, we will only count the time for comparisons. To compare two elements in locations i and j takes time $\lg(i) + \lg(j)$. For obvious reasons, we will not use location 0.

Assume we have a tree, stored as an array, with exactly $n = 2^k - 1$ nodes, where the left and right children of the root are each heaps. We would like to create a heap out of the whole tree.

How much *comparison* time does the Heapify (i.e. Sift) operation use assuming the root element goes all the way down to a leaf. Just get the exact high order term. To keep things simple, assume the element always goes down to the left. Do NOT modify the algorithm. E.g., do not be clever by keeping the element being Heapified (i.e. Sifted) near the root of the tree. You can assume that $\lg(a+1) \sim \lg(a)$.

Problem 3. **Challenge problem, will not be graded.** Analyze the time for the full Heap Sort.

Problem 4. **Challenge problem, will not be graded.** Modify the Heapify (i.e. Sift) operation to make it efficient. You should take into account the time to move an element, not just the time to compare two elements. Analyze the time.