

Problem 1. We want to sort the following list, $[70, 50, 110, 60, 90, 40, 100, 80]$, of numbers using heap sort algorithm that we covered in class. It will involve two steps, first, building a max heap, second, sorting, by extracting the root and removing any max-heap violations. Apply this algorithm to the given array and answer the following questions.

1. Show the max-heap as a tree. Show the array corresponding to this max-heap. Exactly how many comparisons did it take to build the max-heap?
2. Starting with the max-heap built in the previous step, show the array after each max-heapify. How many comparisons does each max-heapify use? What is the total number of comparisons for just this part, excluding the number of comparisons in Part(1).

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 $A[\lfloor i/2 \rfloor]$.

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 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 $lg(a + 1) \approx lg(a)$.

Problem 3. Assume that each word of your machine has 64 bits. Assume that you can multiply two n -word numbers in time $3n^2$ with a standard algorithm. Assume that you can multiply two n -word numbers in time $8n^{lg3}$ with a “fancy” algorithm. For each part *briefly justify* and *show your work*.

- (a) Approximately, how large does n have to be for the fancy algorithm to be better?
- (b) How many bits is that?
- (c) How many decimal digits is that?