Assignment 13: Merge Sort Analysis Due **in class** *Monday, 1/14*

January 13, 2019
CS: DS&A
Proof School

Write up Parts I and III and turn then in at the start of class!

Code up Part II and submit via Dropbox before class!

Part I.

Suppose that instead of the usual merge sort, we break up our list into two pieces roughly of size 1/3 and 2/3 the original before recursively sorting them. To be more precise, given numbers $a_1, \ldots, a_n$, suppose the two lists we form are $L_1 = a_1, \ldots, a_{\lceil n/3 \rceil}$ and $L_2 = a_{\lceil n/3 \rceil + 1}, \ldots, a_n$. (We then recursively sort them, and merge the two sorted lists together.)

a) What do you think the runtime is of this algorithm?

b) How rigorous an argument can you make for your answer? You may assume anything you like about n (e.g. that it's a power of 3).

Part II.

And now, for a refreshing change: code up regular merge sort! Your algorithm should take a list as input, return nothing, and change the list in place.

Try to do this with the following memory constraint: When given a list of n numbers, you should use only an additional n numbers' worth of memory as scratch work.

Part III.

In class, we gave a fully rigorous proof that the $T(n)$ function associated with merge sort is $\Theta(n \log n)$. It involved a lot of floors and ceilings. Here we outline an alternate approach that takes much less room, but is a little tricky.

Write $n = 2^k + l$ where $0 \leq l < 2^k$. This means

$$2^k \leq n < 2^{k+1}.$$

So assuming that T is increasing (i.e. $n_1 \leq n_2$ implies that $T(n_1) \leq T(n_2)$), we conclude that

$$T(2^k) \leq T(n) \leq T(2^{k+1}).$$

Using these bounds (and the actual formula we developed in class for T of a power of 2), try to carefully show that $T(n)$ is $O(n \log n)$, and that $T(n)$ is $\Omega(n \log n)$.

If you get stuck, we can talk about Problem 3 in class on Monday, and you can turn it in Thursday of next week instead.