**This homework will not have an extension. You must do it by Friday, March 31.**

NP-completeness homework, Part 5, will be due Friday, April 7.

1. Consider an array of size eight with the numbers $30, 80, 50, 40, 70, 20, 10, 60$. Assume you execute quicksort using the version of partition from CLRS. Note that in this algorithm an element might exchange with itself (which counts as one exchange).
   (a) Show the array after the first partition. How many comparisons and exchanges are used?
   (b) Show the left side after the next partition. How many comparisons are used? How many exchanges?
   (c) Show the right side after the next partition on that side. How many comparisons are used? How many exchanges?

2. We would like to take into account *memory hierarchies* when analyzing algorithms. Assume that your machine takes time $f(i)$ to access memory location $i$, for some nondecreasing function $f(i)$. Most of the time, when doing algorithmic analysis, we assume $f(i)$ is a constant. In reality it is a slow growing function. For this problem, to keep things simple, we will assume that $f(i)$ grows linearly, so $f(i) = i\mu$ for some constant $\mu$. This is pretty drastic. We will also assume that there are a constant number of *special memory* locations that take no time to access. Thus to move a value from location $i$ to location $j$ takes $(i + j)\mu$ time; to exchange the values in location $i$ and $j$ takes $2(i + j)\mu$ time; and to move a value from location $i$ into special memory (or vice versa) takes $i\mu$ time.

   We would like to consider implementing Quicksort under these conditions. Assume that the array to be sorted is in locations 1 to $n$. *We will only charge for moving and comparing values from the array.* To compare two values in locations $i$ and $j$ takes $(i + j)\mu$ time. A great place to put the pivot is in special memory. (You cannot put everything into special memory because it only has constant size.) Single variables can be assumed to be stored in special memory.

   EXAMPLE: Assume we Quicksort the values 30,10,20 stored in location 1,2,3, respectively. Then setting the pivot $X$ to 20 takes $3\mu$, comparing the pivot to 30 takes $1\mu$, comparing the pivot to 10 takes $2\mu$, exchanging 30 and 10 takes $2(1 + 2)\mu$, and exchanging the pivot 20 (in location 3) and 30 takes $2(3 + 2)\mu$. The list is now sorted; there is no charge for the rest of the algorithm (since the remaining two lists each have size 1).

   In order to make Quicksort more efficient, our idea is to put the part of the array currently being sorted into the beginning of the array. So, if we are sorting the values from index $p$ to index $r$, we copy them to locations 1 to $r - p + 1$, and then sort them. This risks overwriting parts of the original array. To avoid this, as parts of the array are sorted, they are put into locations $n + 1$ to $2n$, where the final sorted list will reside at the end. For consistency, assume that we always use the last value as the pivot (as done in class).

For the following analyses, we want the exact high order term. You can be a little careless with floors and ceilings.

(a) Write the pseudo-code for this algorithm.

(b) It is easier to calculate the cost for moving all $n$ values into their final sorted positions (in locations $n+1$ to $2n$), rather than saving that calculation for each individual value to do inside a recurrence. What is the cost to move all $n$ values into their final sorted positions? Justify.

(c) We will analyze this algorithm, assuming we always partition on the largest value.

(i) How much time does partition take in the worst case, for an array of size $n$? Justify.

(ii) Analyze how much time the algorithm takes, using a recurrence. Use your result from Part (i). Show your work.

(d) Assume we always partition on the median value. Assume partition takes time $(n^2 + O(n))\mu$. Analyze how much time the algorithm takes, using a recurrence. Show your work.

(e) Suppose that the pivot is the $q$th smallest element: assume that partitioning takes time $nq\mu$ on average, and moving the larger list to the early locations takes time $(2n^2 - nq)\mu$ on average. Analyze the average time the algorithm takes, using a recurrence. Show your work.

(f) **Challenge Problem. Will not be graded.** Implement an efficient, in place version of Quicksort. An algorithm is *in place* if it only uses a constant amount of extra memory.