due Monday, 6 October 2014, at 5:00 PM

[Total points: 24]

1. [worth 6 points] *understanding Heapsort and algorithm lower bounds*

Give (and prove) a Θ bound for the *average* number of comparisons used by Heapsort on an array of n elements. As with the average case analysis of Quicksort, all permutations are equally likely.

Actually the assumption about permutations being equally likely is not relevant. You will end up showing that, as long as all keys are distinct initially, even the best case number of comparisons for heapsort is $\Omega(n \lg n)$ and the average case can obviously be no better than the best case. Oddly, converting the keys to 0's and 1's, which are far from distinct, makes the argument much easier.

The gist of your argument, if you want to pursue this the way I did it, is as follows. Argue that, in any configuration, not too many 0's can be close to the bottom, i.e., there must be enough 1's close to the bottom to force enough 0's to trickle most of the way down when they are swapped into the root position.

2 points for explaining why 0's and 1's are sufficient for the analysis, 4 for proving the lower bound, 1 for proving the upper bound and completing the argument. That makes a total of 7 points – there's a built-in extra credit point if you nail it.

2. [worth 8 points] *understanding Quicksort*

(a) Describe a modification of the PARTITION procedure, call it EQ-PARTITION that splits the array into three parts as follows: Return two indices r and t so that $p \le q \le t \le r$, and

- all elements of $A[q \ldots t]$ are equal,
- each element of $A[p \ldots q-1]$ is less than $A[q]$, and
- each element of $A[t+1 \ldots r$ is greater than $A[q]$.

Your procedure must do $\mathrm{O}(r-p)$ comparisons and swaps and be in-place, i.e., no additional space other than a few variables may be used.

(b) Assuming that EQ-PARTITION is used for partitioning in QUICKSORT and no recursive calls are done for elements with keys equal to the pivot, what is the worst-case number of comparisons required to sort an array whose elements have keys that are entirely 0's and 1's. Your answer should give the *exact* number of comparisons based on your solution to part (a) and describe what the array for the worst case looks like. Note that the algorithm does not "know" that the inputs are 0's and 1's.

(c) Give a Θ bound for the worst case number of comparisons when the keys are from the set $\{1, \ldots, k\}$. Again your bound should assume that EQ-PARTITION is used. Your bound should be in terms of both n and k. In this situation a Θ bound would be defined as:

> $f(n,k)$ is $\Theta(g(n,k))$ if there exist four positive constants c_1, c_2, n_0 and k_0 so that $c_1 g(n,k) \le f(n,k) \le c_2 g(n,k)$ for all $n \ge n_0$ and $k \ge k_0$.

For the lower bound, you should describe what a general worst-case example looks like.

3 points for part (a), 2 points for (b), 3 points for (c)

3. [worth 4 points] *understanding decision trees*

Draw the decision tree for *selection sort* on 3 elements. What is the worst case number of comparisons? What is the average case?

4. [worth 6 points] *understanding the linear-time selection algorithm*

Exercise **9.3-1** on page 223 (page 192 in 2/e).

3 points for each part