15-451 Algorithms, Fall 2014

Homework # 1 **Solutions**

(60 pts) 1. **Compare and Contrast.**

Dr. Demento is holding you hostage on the notorious Loch Madness, and will release you only if you can solve the following four problems. In all cases, S is a set of n arbitrary but distinct numbers.

(a) Give an algorithm to output the largest $n^{3/4}$ of the numbers in S in sorted order. Your algorithm must use $O(n)$ comparisons.

Solution: First use, say, the deterministic selection algorithm from Lecture #2 to find the k^{th} smallest element q, where $k = n - n^{3/4}$. This takes $O(n)$ comparisons. Take all the $n - k = n^{3/4}$ elements greater than q, and sort them using, say, MergeSort, and output the result. This takes at most $n^{3/4} \log_2 n^{3/4} = O(n^{3/4} \log n) = o(n)$ comparisons. A total of $O(n)$ comparisons.

(b) Give an algorithm to find both the maximum element *and* the minimum element in S. (You may assume that n is even.) The algorithm must use at most $\frac{3}{2}n - 2$ comparisions.

Solution: Pair the n elements arbitrarily and compare them. Let W be the set of "winners" of these $n/2$ comparisons, and L be the losers. Note that the maximum in S must lie in W, and the minimum must lie in L. Now use $|W| - 1$ comparisons to find the maximum of the set W, and $|L| - 1$ comparisons to find the minimum of the set L, and output these as the maximum and minimum respectively. The total number of comparisons is $n/2 + 2(n/2 - 1) = \frac{3}{2}n - 2$.

(c) Show that any deterministic comparison-based algorithm that correctly outputs the median on inputs of n distinct elements must use at least $n - 1$ comparisons. To be completely precise, the median is defined to be the $\lceil n/2 \rceil^{th}$ smallest element.

Solution: Suppose there an algrithm A and some input of n elements such that the algorithm A uses fewer than $n-1$ comparions on this input. Consider a graph on these n elements where we add an edge between two elements when they are compared by A. If A uses fewer than $n - 1$ comparisons, this graph is not connected, it contains at least two components. Say C and C' are two components, such that the median m lies in C.

Suppose at least one element e of C' is smaller than m. Then increase the values of all elements in C' by some large quantity so that e is now larger than m. Note that m is no longer the median, since the number of elements smaller than it has changed. But since we changed only the elements of C' and changed them all by the same amount, all comparisons made by A will give the same results on this new input, and hence A will still output m as the median, which is incorrect. This contradicts the claim that A computes the median correctly.

A similar argument holds when all elements of C' are larger than m, except that we'd create the new input by reducing the values of elements in C' by some large amount.

(d) You have three algorithms A, B, and C, which solve the same problem in different ways. On an input of size n:

- Algorithm A breaks it into 3 pieces of size $n/2$, recursively solves each piece, and then combines the solutions in time $n^{2.5}$.
- Algorithm B breaks it into 4 pieces of size $n/2$, recursively solves each piece, and then combines the solutions in time n^2.
- Algorithm C breaks it into 5 pieces of size $n/2$, recursively solves each piece, and then combines the solutions in time $n^{1.5}$.

Give the runtime of these algorithms in $\Theta()$ notation, and sort these runtimes from fastest to slowest. (Assume n is a power of 2 and $T(1) = 1$ for all three.)

Solution: The recurrences are

$$\begin{aligned} T_A(n) &= 3T_A(n/2) + n^{2.5} \\ T_B(n) &= 4T_B(n/2) + n^2 \\ T_C(n) &= 5T_C(n/2) + n^{1.5} \end{aligned}$$

with $T(1) = 1$ for all three. This solves to (using, e.g., the Master theorem) to

$$T_A(n) = \Theta(n^{2.5}), \quad T_B(n) = \Theta(n^2 \log n), \quad T_C(n) = \Theta(n^{\log_2 5}) = \Theta(n^{2.32..})$$

So the ordering is B, C, A.

(40 pts) 2. **Tight Upper/Lower Bounds (It Takes All Sorts of Sorts)**

Consider the following problem.

INPUT: an $n \times n$ matrix M containing n^2 distinct numbers, where the n numbers in each row of the matrix are in sorted order. (Such a matrix is called a *row-sorted* matrix.)

OUTPUT: a sorted list L of the n^2 numbers in the matrix M.

EXAMPLE: $n = 3$, so $n^2 = 9$. Say the 9 numbers in M are the digits $1, \ldots, 9$. Possible inputs (row-sorted matrices) include:

```
1 4 7         1 4 5        3 4 6        1 2 3
3 5 8    or   2 7 8   or   2 5 9   or   4 5 6   or  ...
2 6 9         3 6 9        1 7 8        7 8 9
```

The output for all these would be the sorted list $L = 1, 2, 3, 4, 5, 6, 7, 8, 9$.

It is clear that we can solve this problem using at most $2n^2 \lg n$ comparisons by forgetting about the row-sorted structure of the input matrix M, and sorting the n^2 numbers using, say, MergeSort. (Remember that $\lg n^2 = 2 \lg n$, where $\lg x == \log_2 x$.)

In this problem you will show how to do better and then give a lower bound. For simplicity, you can assume n is a power of 2.

(a) Show how to solve this problem using at most $n^2 \lg n$ comparisons.

(b) Show that any comparison-based algorithm to solve this problem must use at least $\frac{9}{10} n^2 \lg n - O(n^2)$ comparisons.
Note: the number $\frac{9}{10}$ is somewhat arbitary. A lower bound of $n^2 \lg n - O(n^2)$ comparisons also follows from the same ideas, but as long as you correctly prove a lower bound of $\frac{9}{10} n^2 \lg n - O(n^2)$ comparisons, you get all the points.

Some hints for part (b): Show that if you could solve this problem using fewer than that many comparisons, then you could use this to violate the $\lg(m!)$ lower bound for comparisons needed to sort m elements (which we prove in Lecture #2). You may want to use the fact that $m! > (m/e)^m$. Also, recall that you can merge two sorted arrays of size k using at most $2k-1$ comparisons.

Solution: For part (a), we will use the fact that merging two sorted lists of size K takes (2K-1) comparisons.

Now use merge-sort to combine the matrix rows in pairs (first pair the sorted rows and merge them, the merge the resulting lists in pairs, etc.), which takes $(n/2) \times (2n-1) + (n/4) \times (4n-1) + ... + 1 \times (2 * n^2/2 - 1) \leq n^2 \lg n - (n-1)$ comparisons.

For part (b), recall the lower bound for sorting n^2 numbers is

$$\ln(n^2!) \geq n^2 \lg n^2 - 1.4n^2 \geq 2n^2 \lg n - 1.4n^2, \tag{1}$$

using the "information theoretic bound" in Lecture #2.

So suppose there was an algorithm $\mathcal{A}$ that could get from any row-sorted matrix M to a sorted list L using at most $An^2 \lg n - B$ comparisons. Using $\mathcal{A}$ we show how to sort an arbitrary set of n^2 elements using

$$(A+1)n^2 \lg n - Bn^2 \tag{2}$$

comparisons. This must be at least the lower bound (1), which means that $\mathcal{A}$ requires at least $n^2 \geq 2n^2 \lg n - 1.4n^2$ comparisons.

Indeed, start with any set of n^2 elements. Break this into an n groups of n elements each, in an arbitrary way. This requires *no* comparisons, of course. Now sort each of these n groups, and place them in the n rows of a matrix M. Using MergeSort, for instance, this construction of M takes at most $n(n \lg n) = n^2 \lg n$ comparisons. Now using this supposed algorithm $\mathcal{A}$, we can get from M to a sorted list in a further $An^2 \lg n - Bn^2$ comparisons. A total of $(A+1)n^2 \lg n - Bn^2$ as claimed in (2).

Remark: Some of you (erroneously) claimed that you could show a lower bound of $n^2 \lg n$. But note that the solution to part (a) sketched above takes strictly fewer than that number of comparisons, so this lower bound must be clearly false. Whenever you prove a lower bound, one useful check is to make sure you cannot do better than the lower bound—if you can, something is clearly broken.