15-451 Assignment 01

Karan Sikka
ksikka@cmu.edu

September 1, 2014

1a.

Compute $n^3/4$ in constant time.
Use DeterministicSelect to select the $n^{3/4}$th largest number in $O(n)$ time.
Then filter out the elements greater than or equal to it in $O(n)$ time.
Now sort the $n^{3/4}$ numbers using mergesort in $O(n^{3/4}log(n^{3/4}))$ time.
The algorithm seems to be dominated by the latter expression, but it can be reduced to $O(n)$ as follows:

$O(n^{3/4}log(n^{3/4}))$
$\leq O(\frac{3}{4}n^{3/4}log(n))$

Notice that $O(n^{1/4}) \geq O(log(n))$ so we can make the following substitution:

$\leq O(\frac{3}{4}n^{3/4}n^{1/4})$
$\leq O(n)$

1b.

Pair up the elements, and for each pair, compare the elements. ($\frac{n}{2}$ comparisons)
Call the larger element a "winner" and the smaller element a "loser".
Among the $\frac{n}{2}$ winners, find the max by going one by one keeping track of the max so far. ($\frac{n}{2} - 1$ comparisons)
The minus one term is because you don't have anything to compare the first element to - you assume it as the max at first.
This is the max of all the elements.

Among the $\frac{n}{2}$ losers, find the min by going one by one keeping track of the min so far. ($\frac{n}{2} - 1$ comparisons)
This is the min of all the elements.

The sum of all the comparisons is $(\frac{n}{2}) + (\frac{n}{2} - 1) + (\frac{n}{2} - 1) = \frac{3n}{2} - 2$

1c.

In this proof, I will:

1. Show that if the graph of comparisons contains disconnected subgraphs,
then you could change the inputs so that the answers to the comparisons remain the same,
but the global ordering of the elements changes such that the median changes, making the algorithm incorrect.

2. Show that with only $n - 2$ comparisons, you must have multiple disconnected chains in the comparison graph.

Proof:

1. Construct a graph of the comparisons such that the vertices are the inputs and there is a directed edge between vertices if the vertices are compared by the algorithm.

Notice that if the graph is disconnected, then you can add a scalar to all the numbers in a subgraph not containing the median,
without altering the answers to the comparisons.

Furthermore, you can find a scalar to add to this subgraph which will cause the median to be in this subgraph instead of that subgraph. This is true because adding a scalar to elements can shift them left or right arbitrarily in the ordering.

2. A path of n vertices must have $n-1$ edges. So with $n-2$ comparisons (edges), no path can cover all vertices.
The graph must be disconnected. By subclaim 1, an algorithm which performs $n-2$ comparisons cannot be correct on all inputs.

1d.

Since all recurrences are divide-and-conquer style, we can use the Master theorem to solve all of them.

A. $a=3, b=2, c=1, k=2.5$
$r = a/b^k = 3/2^{2.5} \approx .5303 < 1$
Then the recurrence is in $\Theta(n^k) = \Theta(n^{2.5})$

B. $a=4, b=2, c=1, k=2$
$r = a/b^k = 4/2^2 = 1$
Then the recurrence is in $\Theta(n^k log(n)) = \Theta(n^2 log(n))$.

C. $a=5, b=2, c=1, k=1.5$
$r = a/b^k = 5/2^{1.5} \approx 1.7678 > 1$
Then the recurrence is in $\Theta(n^{log_b(a)}) = \Theta(n^{log_2(5)})$.

Ordering:
$B < C$ since $\Theta(n^2 log(n)) \leq \Theta(n^2 n^{1/2}) \leq \Theta(n^{2.5})$
$C < A$ since $log_2(5) \approx 2.3219 \leq 2.5$
Thus $B < C < A$.

2a

For the sake of this proof, we'll think of the matrix as n sorted arrays of n.
You can look at the problem as recursively merging the n sorted arrays pairwise.
It's obvious that this algorithm will be correct as each of the arrays is sorted and you can easily merge two sorted arrays to produce another sorted array containing the original elements.
First observe there will be $log(n)$ iterations as we're merging arrays pairwise.
At each step, the total number of comparisons is at most n^2 since each of the n elements need only be compared with any other element once in the sequential merging algorithm.
So the number of comparisons in the algorithm is at most $n^2 log(n)$.

2b

If the algorithm takes k comparisons, then there are 2^k possible sets of answers.
For each set of answer, there is exactly one abstract matrix for which that set of answers is correct (and no other set of answers is correct).
This logic yields the following equality:
2^k =(number of abstract matrices for which the algorithm is correct).
But the total number of abstract matrices is $\frac{(n^2)!}{nn!}$

So the minimum number of comparisons required to be correct on every input is k where
$k = log_2(\frac{(n^2)!}{nn!})$
$= log_2((n^2)!) - (log_2(n) + log_2(n!))$

We can find a lower bound first log term by looking at the factorial expansion (as we did in lecture) and realizing that

$$\frac{n^2}{2}^{\frac{n^2}{2}} < n^2!$$

$k \geq \frac{n^2}{2} log_2(\frac{n^2}{2}) - (log_2(n) + log_2(n!))$
$= \frac{n^2}{2}[log_2(n^2) - log_2(2)] - (log_2(n) + log_2(n!))$
$= n^2 log_2(n) - \frac{n^2}{2} - (log_2(n) + log_2(n!))$
$= n^2 log_2(n) - O(n^2)$

Note that the $log_2(n!)$ term is in $O(nlog(n))$ as was discussed in lecture.