15-451/651 Algorithms, Spring 2019

Homework #1 **Due: January 30, 2019**

---

0. **(Exercises: Recurrences and Probability.)**

Solve each recurrence below in Θ notation. As always, prove your answer. For all of these problems $T(1) = 1$.

(a) Solve $T(n) = 3\,T(\lfloor n/2 \rfloor) + n$.

Solution: $T(n) = \Theta(n^{\log_2 3})$. Master theorem.

(b) Now solve $T(n) = 3\,T(\lfloor n/2 \rfloor) + n \lg n$.

Solution: $T(n) = \Theta(n^{\log_2 3})$. For this we can also use the Master theorem. We'll use it to solve two recurrences:

$$T'(n) = 3\,T'(n/2) + n$$

and

$$T''(n) = 3\,T''(n/2) + n^{3/2}$$

By the master theorem, the solution to both of these is $\Theta(n^{\log_2 3})$. Also we have:

$$T'(n) < T(n) < T''(n)$$

From which the result follows

(c) Finally, solve $T(n) = n^{2/3}\,T(\lfloor n^{1/3} \rfloor) + n$.
(E.g., we might get this from a divide-and-conquer procedure that uses linear time to break the problem into $n^{2/3}$ pieces of size $n^{1/3}$ each. Hint: write out the recursion tree.)

Solution: Each level of the recursion tree has a total cost of n, so we only need to figure out the number of levels. If we write n as 2^k we see that as we go down the tree the problem sizes are $2^k, 2^{k/3}, 2^{k/9}, 2^{k/27}, \ldots, 2^{k/(3^{\lceil \log_3 k \rceil})}$. So, the depth of the tree is $\Theta(\log k) = \Theta(\log \log n)$. So, the recurrence solves to $\Theta(n \log \log n)$.

(25 pts) 1. **(A Box of Sorts (or a Sort of Box)?)** Consider the following problem.

INPUT: n^2 distinct numbers in some arbitrary order.

OUTPUT: an $n \times n$ matrix containing the input numbers, and having all rows and all columns sorted in increasing order.

EXAMPLE: $n = 3$, so $n^2 = 9$. Say the 9 numbers are the digits $1, \ldots, 9$. Possible outputs include:

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

It is clear that we can solve this problem in time $2n^2 \lg n$ by just sorting the input (remember that $\lg(n^2) = 2 \lg n$) and then outputting the first n elements as the first row, the next n elements as the second row, and so on. Your job in this problem is to prove a matching $\Omega(n^2 \log n)$ *lower bound* in the comparison-based model of computation.[1]

For simplicity, you can assume n is a power of 2. Recall $\lg x = \log_2 x$.

Some hints: Show that if you could solve this problem using $o(n^2 \log n)$ comparisons (in fact, in less than $n^2 \lg(n/e)$ 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 n using at most $2n - 1$ comparisons.

Solution: We show that if we are able to solve the "Matrix sorting" problem using less than $n^2 \lg(n/e)$ comparisons, we can sort n^2 elements in less than $\lg((n^2)!)$ comparisons, violating our sorting lower bound from lecture.

Here is to do the sorting. Begin by putting the n^2 elements into an n-by-n matrix and running our fast matrix-sorting algorithm. Then repeatedly merge the rows of the matrix as follows. First, pair up the rows, and merge rows $2i - 1$ and $2i$ $(i = 1, ..., \frac{n}{2})$, thereby obtaining a new, $\frac{n}{2} \times 2n$ matrix (which will obviously have its rows sorted). Now pair up the rows of this matrix, merging rows $2i - 1$ and $2i$ $(i = 1, ..., \frac{n}{4})$, and obtaining a new matrix of size $\frac{n}{4} \times 4n$ (which, again, will have its rows sorted). Repeat the procedure until we merge the two rows of a $2 \times \frac{n^2}{2}$ matrix and get the sorted sequence of n^2 numbers. (We are using the fact that n is a power of 2, so in every step we can pair up the rows.)

The total number of comparisons this algorithm uses for merging is at most:

$$\frac{n}{2}(2n-1) + \frac{n}{4}(4n-1) + \frac{n}{8}(8n-1) + ... + 2(\frac{n^2}{2} - 1) < n^2 \lg n.$$

Adding the above to the $n^2 \lg(n/e)$ comparisons we supposedly used for the matrix-sorting problem, we get a total of less than $n^2 \lg(n^2/e) = \lg((n^2/e)^{n^2}) < \lg((n^2)!)$ comparisons, violating our sorting lower bound.

(25 pts) 2. **Sorting with Strange Primitives**

In both of the parts below an array A initially contains a permutation of $1, \ldots n$. The goal in each case is to give an algorithm to sort the array by applying a linear number of the given primitives.

[1] By an "$\Omega(n^2 \log n)$ *lower bound*", we mean a lower bound of $cn^2 \log n$ for some constant $c > 0$ that is independent of n.

(a) A *prefix reversal* of A just reverses some prefix of the array. For example if $A = [1, 5, 3, 2, 4]$ this can be turned into $[3, 5, 1, 2, 4]$ with one prefix reversal (of the first 3 elements of A). Give an algorithm for sorting any initial permutation in at most $c_1 n$ prefix reversals. What is c_1 for your algorithm? (Try to make it as small as you can.)

(b) A *block swap* takes a pair of neighboring blocks of A of the same size and exchanges them. For example if $A = [8, 1, 3, 2, 4, 6, 5, 7]$ in one block swap we could get $[4, 6, 5, 7, 8, 1, 3, 2]$, or $[8, 3, 1, 2, 4, 6, 5, 7]$, or $[2, 4, 6, 8, 1, 3, 5, 7]$, but not $[7, 1, 3, 2, 4, 6, 5, 8]$. Give an algorithm for sorting any initial permutation in at most $c_2 n$ block swaps. What is c_2 for your algorithm? (Try to make it as small as you can.)

(25 pts) 3. **More Upper/Lower bounds (Stuck Here in the Middle with You.)**

Consider the problem of finding both the maximum element and the minimum element of an arbitrary set of n distinct numbers, where n is even.

It turns out that $\frac{3}{2}n - 2$ comparisons are required in the worst case to do this. That is, $\frac{3}{2}n - 2$ is a lower bound on the number of comparisions needed. There's also an algorithm that achieves this bound. That is $\frac{3}{2}n - 2$ is an upper bound on the number of comparisons needed. Since these bounds are equal, they are optimal.

(a) Prove the following theorem:

Theorem: Consider a deterministic comparison based-algorithm $\mathcal{A}$ which does the following: Given a set S of n numbers as input, $\mathcal{A}$ returns the largest and the smallest element of S. Prove that there is an input on which $\mathcal{A}$ must perform at least $\frac{3}{2}n - 2$ comparisons.

Hints:

Call an element *top* if it has been involved in at least one comparison and it has never lost a comparison.
Call an element *bottom* if it has been involved in at least one comparison and never won a comparison.
Call an element *free* if it has been involved in zero comparisons.
Call an element *middle* if it has won at least one comparion and lost at least one comparison.
Every element falls into exactly one of these categories, and this classification of elements evolves over time. Initially all elements are free. You know that at the end of a run of any correct algorithm, there are _____free ones, _____middle ones, _____top ones and _____bottom ones.
Define a potential function $\Phi()$ that is a linear function in the number of each type of element. You, the adversary, have to decide the outcome of each comparison to give to algorithm $\mathcal{A}$ when it asks you to compare two elements. By making the "right" choice of the outcome of a comparison, you should ensure that the potential decreases by at most 1 for each comparison. Using this fact, along with the initial value and final value of the potential, prove that algorithm $\mathcal{A}$ must do at least $\frac{3}{2}n-2$ comparisons.

Solution: Using F, T, B, M for the count of free, top, bottom, and middle elements, let $\Phi(s_i) = \frac{3}{2}F + T + B$. I next show that for each comparison, the adversary can guarantee the potential drops by at most -1:

- (free, free): always results in one top and one bottom, so the drop in potential is $-2\frac{3}{2} + 2(1) = -1$.
- (free, bottom) and (free,top) and (free,mid): the adversary can answer free>bottom and so only the free changes to a top. Similarly the adversary can answer free<top and so only the free changes to a bottom. In both cases the potential drop by $-\frac{3}{2} + 1 = -\frac{1}{2}$. The case of (free,mid) can be arbitrarily answered and results in one of the prior two cases.
- (top, top), (bottom, bottom): always results in one element becoming the middle and the other element staying in the current category, so the drop in potential is -1.
- (top,bottom): the adversary can answer top>bottom, and so the potential doesn't change.
- For all other operations, answer arbitrarily in a way that is consistent with previous operations. In all remaining cases of (top,mid) and (bottom,mid) and (mid,mid), the potential drops by at most -1.

Thus the adversary guarantees that potential drops by at most -1. The starting state of all frees has a potential of $\frac{3}{2}n$ and the ending state of 1 top, 1 bottom, and the rest middle has a potential of 2, so the adversary can guarantee that an algorithm will take at least $\frac{3}{2}n - 2$ steps. Of course, since the number of comparisons must be an integer, we get a lower bound of $\lceil \frac{3}{2}n - 2 \rceil$.

What remains is to show that the adversary is consistent. For sake of a contradiction, suppose it wasn't consistent. Let $a_k < a_1$ be the first inconsistent answer given, so the adversary has already answered $a_1 < a_2 < \cdots < a_k$. Consider the state before this comparison. In this state, a_1 is either a bottom or middle, since it has already lost a comparison. Also, a_k is either a top or a middle, since it has already won a comparison. At least one of these two must be middle, because if neither was middle the adversary would have answered $a_1 < a_k$. But if one of these was middle, then the adversary would have answered in a manner consistent with the previous answers and said $a_1 < a_k$, which is a contradiction. So the adversary is consistent.

(b) The proof above leads to a optimal deterministic algorithm (in terms of the number of comparisions). Describe one such algorithm.

Solution: Here is one such algorithm: At any time, pick any move that causes the potential $\frac{3}{2}F + T + B$ to decrease by 1. As long as there are two tops or two bottoms or two frees, we can do this. If you have no such move,

- you are either done with one item each in the top and bottom, and the rest in the middle, the potential is now 2. Since this potential starts at $\frac{3}{2}n$ and ends at 2, any such algorithm would take $\frac{3}{2}n - 2$ comparisons. This matches the lower bound.
- Or you have one item each in top, bottom, and free, and the rest in the middle (with current potential $\frac{3}{2} + 1 + 1$). This will happen only when n is odd. You've

done $\frac{3}{2}(n-1)-2$ comparisons, equal to the drop in potential thus far. Now you need two more comparisons, giving a total of $\frac{3}{2}(n-1)$ comparisons. However, for n being odd, $\frac{3}{2}(n-1) = \frac{3}{2}n - \frac{3}{2} = \lceil \frac{3}{2}n - 2 \rceil$, which matches the lower bound.