15-451 Algorithms, Spring 2009

Homework # 2 **due: Tue-Thu, February 10-12, 2009**

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. Please sign up for a 1-hour time slot for your group *in recitation* on Wednesday, Feb 3. From Wednesday afternoon until Sunday, Feb 8 at 11:59pm, there will be a signup sheet posted on the door of Doherty Hall, 4301a. You can use this to sign up, although the later you do this, the less likely you are to get a good time slot.
- Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. The other group members may assist the presenter.
- You are not required to hand anything in at your presentation, but you may if you choose.

1. **Problem 1: median of two sorted arrays (Pranjal Awasthi)**

   Consider a new definition of the median: We will define the median of an array of $2k$ elements as the ordered pair (a_i, a_j) such that a_i is greater than $k-1$ elements and less than k elements and a_j is greater than k elements and less than $k-1$ elements. For example in the array $[2, 3, 4, 5, 6, 7, 9, 11]$, the median would be $(5, 6)$. The median of an array containing $2k-1$ elements is defined as the element which is greater than $k-1$ elements and less than $k-1$ elements.

   Let $A = [a_1, \ldots, a_n]$ and $B = [b_1, \ldots, b_n]$ be two sorted arrays of n elements each. Assume that n is a power of 2 and that no two elements are the same. We can easily, i.e. without making any comparisons, find the location of the median in A — it is just $(A[\frac{n}{2}], A[\frac{n}{2}+1])$ — and similarly we can, without making any comparisons, find the location of the median in B. But, what if we want to find the location of the median overall — i.e., the location of the n^{th} smallest and the $(n+1)^{th}$ smallest elements in the union of A and B. For example if the two arrays are $A = [2, 3, 4, 6]$ and $B = [5, 7, 9, 11]$, then the location of the overall median would be $(B[1], A[4])$(Assume that the arrays start from 1).

   Specifically, assuming that every comparison costs one dollar, given A and B how cheaply can we find the location of the pair of elements which form the median in the union of the two arrays?

   (a) Give an algorithm to solve the problem. Prove that your algorithm works and give a function $f(n)$ such that using your algorithm one will pay at most $f(n)$ dollars.

   (b) Give a function $g(n)$ and prove that using any algorithm, one must pay at least $g(n)$ dollars in the worst case.

   The points you get on this problem will depend on how close $f(n)$ and $g(n)$ are to each other.

   Some hints: You may wish to try small cases. For the lower bound, notice that the output of the algorithm is the location of the desired median (e.g, "$(A[17], B[24])$"). How many different possible outputs are there?

Extra Credit: Can you come up with a better algorithm if one is only interested in knowing the unordered median pair. For example, what if both $(A[3], B[1])$ and $(B[1], A[3])$ are valid answers?

2. **Problem 2: tight upper/lower bounds (Maxim Makatchev)**

   Consider the following problem (let's call it Matrix Sorting).

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

   OUTPUT: an $n \times n$ matrix of the inputs having all rows and columns sorted in increasing order.

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

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

   In this problem you are going to prove tight upper and lower bounds in comparison model of computation. Namely, you will show that you can solve this problem using $O(n^2 \log n)$ comparisons and also that you need at least $\Omega(n^2 \log n)$ comparisons to solve it.

   (a) Show how to solve Matrix Sorting in time $O(n^2 \log n)$.

   (b) Show that you can merge two sorted arrays of size n using at most $2n-1$ comparisons.

   (c) Show that if you could solve Matrix Sorting using less than $n^2 \lg(n/e)$ comparisons, then you could sort an array of m elements using fewer than $\lg(m!)$ comparisons. You may want to use the fact that $m! > (m/e)^m$. For simplicity, you can also assume that n is a power of 2. As before, lg stands for $\log_2$.

   (d) Explain why your result in part (c) implies that there is no solution to the Matrix Sorting problem that uses less than $c \cdot n^2 \lg(n)$, for any constant c, such that $0 < c < 1$.

   (e) Explain why your result in part (d) implies that the lower bound on the number of comparisons in the Matrix Sorting problem is $\Omega(n^2 \log n)$.

3. **Problem 3: Amortized Analysis (Daniel Nuffer)**

   Imagine a computer, with a queue of n processors (CPUs). Each processor may have any number of threads running on it.

   Initially, out computer has $n - 1$ threads in total, with the first $n - 1$ processors in the queue each have one thread. We never add more threads to our computer, so there are always $n - 1$ in total. However, occasionally a processor must be reset, in which case it goes to the end of the queue and its m threads are reassigned, one apiece, to each of the m processors at the front of the queue. The cost of resetting a processor with m threads is exactly m.

   The following table gives an example of what could happen. In the initial state, the

first 4 processors in the queue have a thread. If the processor at the top of the queue resets, it gets moved to the back, and its thread gets reassigned to the new first processor in the queue, which now has 2 threads. In the final step, the first two processors in the queue have 2 threads each. Then the second processor in the queue resets and moves to the back of the queue. The result is that the new top processor in the queue now has 3 threads, whilst the other thread from the resetting processor gets reassigned to the third processor, which is now in second place.

Queue Index	1	2	3	4	5
Initial State	1	1	1	1	0
After resetting index 1	2	1	1	0	0
After resetting index 1	2	2	0	0	0
After resetting index 2	3	1	0	0	0

(a) Find and prove an upper and lower on the worst case cost of a single reset. (Please be exact - no big-O notation.)

(b) Find and prove an upper and lower bound on the worst case amortized cost per reset in big-O notation.