15-451 Algorithms, Fall 2004

Homework # 3  
due: Tuesday October 12, 2004

Please hand in each problem on a separate sheet and put your name and recitation (time or letter) at the top of each sheet. You will be handing each problem into a separate box, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done individually. Group work is only for the oral-presentation assignments.

Problems:

(35 pts) 1. Hashing. As discussed in class, the notion of universal hashing gives us guarantees that hold for arbitrary (i.e., worst-case) sets $S$, in expectation over our choice of hash function. In this problem, you will work out what some of these guarantees are.

(a) Describe an explicit universal hash function family from $U = \{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{0, 1\}$. Hint: you can do this with a set of 4 functions.

(b) Let $H$ be a universal family of hash functions from some universe $U$ into a table of size $m$. Let $S \subseteq U$ be some set we wish to hash. Prove that if we choose $h$ from $H$ at random, the expected number of pairs $(x, y)$ in $S$ that collide is $O(|S|^2/m)$.

(c) Prove that for some constant $c$, with probability at least $3/4$, no bin gets more than $1 + c|S|/\sqrt{m}$ elements. (So, if $|S| = m$, you are showing that with probability $3/4$ no bin gets more than $1 + c\sqrt{m}$ elements.) Hint: use part (b).

To solve this question, you should use “Markov’s inequality”. Markov’s inequality is a fancy name for a pretty obvious fact: if you have a non-negative random variable $X$ with expectation $E[X]$, then for any $k > 0$, $\Pr(X > kE[X]) \leq 1/k$. For instance, the chance that $X$ is more than 100 times its expectation is at most 1/100. You can see that this has to be true just from the definition of “expectation”.

(30 pts) 2. Treaps and amortized analysis. Suppose you have an array of $n$ keys that is already sorted, and you want to convert it into a treap (e.g., so that you can later do additional inserts). Here is a procedure for converting the array into a treap in linear time, no matter what the priorities are — we won’t be relying on the priorities being chosen randomly here. The procedure walks down the array, inserting the elements one at a time in a special way. Your job is to show that the amortized cost per insert for this procedure is $O(1)$.

First of all, in addition to keeping a pointer to the root node, we will also keep a pointer to the rightmost node of the treap. (The rightmost node is the one with the largest key so far). Also, every node will have a parent pointer in addition to left-child and right-child pointers.

Algorithm. Let $A$ be the input array, where the $i$th key and priority appear in $A[i].key$ and $A[i].prio$ respectively, and the keys are in sorted order. We will insert the elements one by one, into an initially empty treap $T$. 
We insert element $i$ into the treap $T$ made of elements $1 \cdots (i-1)$ as follows:

(a) if $A[i].prio$ is less than the priority of the root of $T$, then $i$ becomes the new root and $T$ is made into its left child;

(b) if $A[i].prio$ is greater than the priority of the rightmost node in the treap, then element $i$ is made into the right child of this node;

(c) if $A[root].prio < A[i].prio < A[right].prio$, then element $i$ is temporarily made the right child of the rightmost node, and the heap property of the treap is then restored by successive rotations of the newly inserted node. (Note: $A[right]$ is really the same thing as $A[i-1]$ since the keys are in sorted order.)

Cases (a) and (b) above are clearly constant-time. The problem is that case (c) could involve a lot of rotations. Your job is to show that nonetheless, the amortized time per operation is $O(1)$.

(35 pts) 3. lower bounds.

Consider the following problem.

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 $1, \ldots, 9$. Possible outputs include:

\[
\begin{array}{ccc}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9
\end{array}
\quad
\begin{array}{ccc}
1 & 4 & 5 \\
2 & 6 & 7 \\
3 & 8 & 9
\end{array}
\quad
\begin{array}{ccc}
1 & 3 & 4 \\
2 & 5 & 8 \\
6 & 7 & 9
\end{array}
\]

It is clear that we can solve this problem in time $O(n^2 \log n)$ by just sorting the input (remember that $\log n^2 = O(\log n)$) and then putting 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.

Some hints: show that if you could solve this problem using $o(n^2 \log n)$ comparisons (in fact, in less than $n^2 \log(n/e)$ comparisons), then you could use this to violate the $\lg(m!)$ lower bound for comparisons needed to sort $m$ elements. 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.

For simplicity, you can assume $n$ is a power of 2.