15-451 Algorithms, Fall 2012

Homework # 2 **due: Mon-Wed, September 24-26, 2012**

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Saturday, September 22 at 11:59pm** your group should sign up for a 1-hour time slot on the signup sheet on the course web page.
- 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.

Problems:

1. [median of two sorted arrays] Let A and B be two sorted arrays of n elements each. We can easily find the median element in A — it is just the element in the middle — and similarly we can easily find the median element in B. (Let us define the median of $2k$ elements as the element that is greater than $k-1$ elements and less than k elements.) However, suppose we want to find the median element overall — i.e., the nth smallest in the *union* of A and B. How quickly can we do that? You may assume there are no duplicate elements.

   Your job is to give tight upper and lower bounds for this problem. Specifically, for some function $f(n)$,

   (a) Give an algorithm whose running time (measured in terms of number of comparisons) is $O(f(n))$, and

   (b) Give a lower bound showing that any comparison-based algorithm must make $\Omega(f(n))$ comparisons in the worst case.

   (c) Now, get rid of the O and Ω to make your bounds *exactly* tight in terms of the number of comparisons needed for this problem.

   Some hints: You may wish to try small cases. For the lower bound, you should think of the output of the algorithm as being the location of the desired element (e.g, "$A[17]$") rather than the element itself. How many different possible outputs are there?

2. [Universal 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 (in addition to solving a few short-answer questions).

   (a) Let $H = \{h_1, \ldots, h_k\}$ be a universal family of hash functions from some universe U ($|U| \geq 2$) into $\{0, 1\}$. Could it be that some function $h_i \in H$ maps all of U to 0? Explain.

(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 a set of m elements 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 $\leq \frac{m-1}{2}$.

(c) Let H be a universal family of hash functions from some universe U into a table of size m. Let $S \subseteq U$ be a set of m elements we wish to hash. Prove that with probability at least 3/4, no bin gets more than $1 + 2\sqrt{m}$ 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 $\mathbf{E}[X]$, then for any $k > 0$, $\mathbf{Pr}(X > k\mathbf{E}[X]) \leq 1/k$. For instance, the chance that X is more that 100 times its expectation is at most 1/100. You can see that this has to be true just from the definition of "expectation".

(d) We say that H is l *universal over range* m if, for every fixed sequence of l distinct keys $\langle x_1, x_2, \ldots, x_l \rangle$ and for any h chosen at random from H, the sequence $\langle h(x_1), h(x_2), \ldots, h(x_l) \rangle$ is equally likely to be any of the m^l sequences of length l with elements drawn from $\{0, 1, \ldots, m-1\}$. It's easy to see that if H is 2-universal then it is universal.

Consider strings of length n, $s = s_1, s_2, \ldots, s_n$ from an alphabet of size k. (Each character is a number in the range $[0, k-1]$.)

An interesting universal family $\mathcal{G}$ can be obtained by generating a 2-dimensional table T of b-bit random numbers where $b = \lg(m)$. The first index of $T_{i,j}$ is in the range $[1, n]$ and the second index is in the range $[0, k-1]$. Here the hash function $g_T()$ is defined as follows:

$$g_T(s) = \bigoplus_{i=1}^{n} T_{i,s_i}$$

where "$\bigoplus$" represents the xor function of integers represented in binary.

Prove or disprove: $\mathcal{G}$ is 3-universal over range $m = 2^b$.

3. [amortized analysis] Suppose we have a binary counter such that the cost to increment or decrement the counter is equal to the number of bits that need to be flipped. We saw in class that if the counter begins at 0, and we perform n increments, the amortized cost per increment is just $O(1)$. Equivalently, the total cost to perform all n increments is $O(n)$.

Suppose that we want to be able to both increment *and* decrement the counter.

(a) Show that even without making the counter go negative, it is possible for a sequence of n operations starting from 0, allowing both increments and decrements, to cost as much as $\Omega(\log n)$ amortized per operation (i.e., $\Omega(n \log n)$ total cost).

(b) To reduce the cost observed in part (a) we'll consider the following *redundant ternary number system*. A number is represented by a sequence of *trits*, each of which is 0, +1, or −1. The value of the number represented by $t_{k-1}, \ldots, t_0$ (where each $t_i, 0 \leq i \leq k-1$ is a trit) is defined to be

$$\sum_{i=0}^{k-1} t_i 2^i.$$

For example, $\boxed{1}\boxed{0}\boxed{\text{-}1}$ is a representation for $2^2 - 2^0 = 3$.

The process of incrementing a ternary number is analogous to that operation on binary numbers. You add 1 to the low order trit. If the result is 2, then it is changed to 0, and a carry is propagated to the next trit. This process is repeated until no carry results. Decrementing a number is similar. You subtract 1 from the low order trit. If it becomes -2 then it is replaced by 0, and a borrow is propagated. Note that the same number may have multiple representations (e.g., $\boxed{1}\boxed{0}\boxed{1} = \boxed{1}\boxed{1}\boxed{\text{-}1}$). That's why this is called a *redundant* ternary number system.

The cost of an increment or a decrement is the number of trits that change in the process. Starting from 0, a sequence of n increments and decrements is done. Give a clear, coherent proof that with this representation, the amortized cost per operation is $O(1)$ (i.e., the total cost for the n operations is $O(n)$). Hint: think about a "bank account" or "potential function" argument.