Problem Set 2

Labs: Feb 22-24, 2012
Due: Mar 1, 2012 10:30am

All problem sets must be handed in electronically by 10:30am on the due date.

Be sure to review the Electronic Submission document on the course website before handing in. **We will not grade submissions that do not meet these regulations.**
Run `cs157_handin ps2` in the directory where your file is stored to hand in.

Please ensure that your solutions are complete, clear, and concise. Points will be deducted for overly complex solutions.

Problem 1

The *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix [1].
- For $k > 0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

Throughout this problem assume that the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

a. What is H_0? What is H_1? What is H_2?

b. Compute $H_0 \cdot (z)$, $H_1 \cdot \begin{bmatrix} x \\ y \end{bmatrix}$ and $H_2 \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$, where a, b, c, d, x, y and z are numbers.

c. Assume you have a black-box that computes $H_{k-1} \cdot v$ for any column vector v (of the appropriate size). Show how to use two calls to this black box, plus $O(2^k)$ additional work, to compute $H_k \cdot u$ for any column vector u (of the appropriate size). Briefly prove the correctness of your approach. As an indication, we expect that the proof will typically be only a few lines long.

d. Design a recursive algorithm that takes as input a non-negative integer k and a column vector v of appropriate size and computes $H_k \cdot v$.

e. Analyze (and prove) your algorithm's runtime. Express the runtime in terms of the size of the input vector $n = 2^k$. As an indication, we expect that the proof will typically be only a few lines long.

Problem 2

The goal of the exercise is to design an algorithm, based on comparisons, that takes as input a positive integer k and an array $A[0...n-1]$ containing $n = 2^m$ elements for an integer m, and outputs whether some element of A is present at least k times. The algorithm should take $O(n \log(n/k))$ time. For example, if $k = 2$, the question is whether A has any duplicate elements. Unlike in problem 3, the only test that we can do is comparing $A[i]$ to $A[j]$. This test has 3 possible outcomes: $A[i] < A[j]$, $A[i] = A[j]$, or $A[i] > A[j]$.

a. What is the desired running time if $k = 2$? Give an algorithm for $k = 2$ matching this running time. (No proof required)

b. What is the desired running time if $k = n/4$? Give an algorithm for $k = n/4$ matching this running time. (No proof required)

c. Give an algorithm for the general case, where k can be any integer between 1 and n, inclusive.

d. Briefly prove correctness. As an indication, we expect that the proof will typically be about $\frac{1}{2}$ of a page, that is, roughly 200 words. A proof longer than 400 words is likely to be wordy and redundant; a proof shorter than 100 words might be imprecise.

e. Briefly prove a running time of $O(n \log(n/k))$. Similarly, we expect that the proof will typically be about $\frac{1}{2}$ of a page.

Problem 3

An array $A[0...n-1]$ is said to have a majority element if more than half of its entries are the same. Given an array A, the task is to design a linear time algorithm to tell whether the array has a majority element, and, if so, to find that element. In other words, we must decide if some element of A is present at least $\frac{n}{2}$ times and return that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (For example, the array elements might be GIF files.) This means that this problem is different than problem 2, since the only test we can do is querying if $A[i] = A[j]$.

a. Consider the following general approach:

   - Pair up the elements of A arbitrarily to get $n/2$ pairs
   - Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them

   This new array has a majority element if A does.

   Apply this approach to the following arrays. You may need to modify the approach for some of the arrays.

   (1) $A = [X, X, X, X, X, Y]$
   (2) $A = [X, Y, X, Y, X, Y]$
   (3) $A = [X, Y, X, Y, Y]$
   (4) $A = [Y, Y, X]$

b. Design an algorithm that takes as input an array A of size n, and outputs whether a majority element exists and, if so, outputs the majority element. The algorithm should take $O(n)$ time.

c. Prove correctness. As an indication, we expect that the proof will typically be about $\frac{3}{4}$ of a page long.

d. Prove a running time of $O(n)$. As an indication, we expect that the proof will typically be only a few lines long.

Problem 4

a. Let $A = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8]$ and $B = [b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8]$ be two lists of numbers sorted in increasing order, and assume that $a_i \neq b_j$ for all i, j. Assume that $a_4 < b_4$. What is the maximum possible rank of a_4 in $A \cup B$, and what is the minimum possible rank of b_4 in $A \cup B$? We define the rank of an element in a list as the index of that element when the list is sorted, using one-based indexing. This means, for example, that the rank of a_4 in the list A is 4.

b. Design an algorithm that takes as input two lists A and B of numbers sorted in increasing order such that $a_i \neq b_j$ for all i, j, and an integer k between 1 and $|A| + |B|$, and gives as output the kth smallest element of $A \cup B$. The algorithm should take $O(\log |A| + \log |B|)$ time. You may assume the input is provided in any basic data structure (e.g. linked list, array, queue), but you must state what data structure you chose in the algorithm.

c. Prove correctness. If your algorithm is iterative, you may state an invariant and prove it briefly by induction. If your algorithm is recursive, you may give specifications of your method and prove briefly by induction that the specifications are maintained. Although the length of your proof will vary based on your style and presentation, note that we expect that the proof will typically be about $\frac{2}{3}$ of a page.

d. Prove a running time of $O(\log |A| + \log |B|)$. As an indication, we expect that the proof will typically be about $\frac{1}{2}$ of a page long.