The University of British Columbia

Department of Computer Science

Computer Science 420—Advanced Algorithm Design and Analysis

Homework Assignment 2

Max-finding and dictionaries

Due: 2015 January 22

Please review the class policy on collaboration before starting this assignment. You are free to discuss problems in groups of size at most three. However, your actual homework submission must be prepared on your own. At the top of the first page of every homework submission you must provide a statement about external resources used in the preparation of your submission. This must clearly acknowledge all resources (including books, websites and discussions with fellow students) that you have used. Submissions missing this statement will not be graded.

1) In class, it was noted (*but not proved*) that if we are given a set $S = \{x_1, \ldots, x_n\}$ drawn from an input space $\mathcal{U} = \{0, 1, \ldots, m-1\}$, then we can determine the maximum element in S using $O(n + lg\, m)$ *unary predicate* evaluations (specifically, tests of the form "is $x_i \geq j$?"). (This was contrasted with the brute-force $O(n\, lg\, m)$-time algorithm that starts by determining the exact value of each of the inputs using binary search.)

   Describe a randomized algorithm, that runs in expected time $O(n + lg\, n \cdot lg\, m)$, and determines the maximum element in S, using only unary predicates (of the form described above). [Hint: recall the solution to the *incremental hiring problem*, described in class.]

2) [A maximum-guessing game] Suppose that the n numbers in some (unknown) set S of distinct numbers are presented to you in random order (i.e each ordering is equally likely). After you see each number, you are asked "Is this the maximum element in S?" If you say "no" then you continue to the next number. If you say "yes" then you win if the number truly is the maximum number in S, otherwise you lose. (Note: that you have only one opportunity to say "yes".)

   a) Describe a strategy to follow that will result in a probability of winning that is at least $1/4$.

   b) Give an analysis of the probability of winning, by following your strategy (i.e. prove that the strategy that you propose in part (a) has the desired property).

3) [See also the text (CLRS), Exercise 11.1-4] In class we sketched a scheme for doing *lazy initialization* of direct access tables for representing (dynamic) sets S drawn from the universe $\mathcal{U} = \{0, 1, \ldots, m-1\}$. We used two arrays $A[0 : m-1]$ and $B[0 : |S|-1]$ and maintain the *invariant*: $i \in S$ if and only if, there exists j, $0 \leq j \leq |S|-1$, such that ($A[i] = j$ and $B[j] = i$). Assuming that initially the set S is empty and the array A has arbitrary values, give pseudocode for each of the operations *member*, *insert* and *delete* that demonstate that they can all be executed in $O(1)$ time.

4) Suppose that we are given $D[1 : n]$, an array of real-valued keys sorted in increasing order, and we wish to use D as a *dictionary*, i.e. for each x_j in a sequence $x_1, x_2, \ldots, x_q$ of queries we want to return the location $\ell(j) \in \{1, \ldots, n\}$ of an element in $D[1 : n]$ that minimizes $|D[\ell(j)] - x_j|$ (in which case we say that key $D[\ell(j)]$ has been *accessed*.)

   It is not uncommon in certain applications that queries to a dictionary exhibit a kind of *locality of reference*: successive queries are frequently made to the same, or close to the same, key. In such situations, it is desirable to exploit this property to reduce the cost of successive searches. For $1 < j \leq q$, denote by Δ_j the value $|\ell(j) - \ell(j-1)|$.

a) Design an algorithm for searching array $D[1:n]$ that exploits locality of reference. Specifically, show that query x_j in the sequence $x_1, x_2, \ldots, x_q$ can be answered at a cost proportional to $lg(2+\Delta_j)$, for all $j > 1$. [Hint: start by designing an algorithm–a kind of blend of linear and binary search– whose cost, when the algorithm returns index i, is proportional to $lg(1+i)$.]

b) Let $\mathcal{A}$ be any comparison-based algorithm for searching $D[1:n]$. Prove that for every k, $1 \leq k < lg\, n$, there exists an index $i_k \in [\ell(j-1) - 2^k,\ \ell(j-1) + 2^k]$ such that $\mathcal{A}$ requires at least k comparisons to search for x_j, knowing $\ell(j-1)$, when it turns out that $\ell(j) = i_k$. [Hint: recall the fact that every binary tree has at most 2^d nodes at depth d, for every $d \geq 0$.]