The University of British Columbia

Department of Computer Science

Computer Science 420—Advanced Algorithm Design and Analysis

Homework Assignment 4

Due: 2015 February 12

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 clearly acknowledge all sources (including books, websites and discussions with fellow students) that you have used in the preparation of your submission.

As always, if you are asked to describe an algorithm you should include (i) sufficient detail to make it structure and behaviour unambiguous, (ii) an argument concerning its correctness; and (ii) an argument concerning its analysis.

1) Recall the *un-dominated points problem* from Assignment 3: **Input**: a set S of n points in the plane, whose x and y co-ordinates are specified by two arrays $X[1:n]$ and $Y[1:n]$ of real numbers (i.e. the i-th point is $(X[i], Y[i])$). **Output**: The sequence consisting of all of the un-dominated points in the input set, ordered by x-coordinate. (Recall that point (x, y) is *dominated* by point (p, q), if $x < p$ and $y \leq q$, or $x = p$ and $y < q$.)

   In the midterm, Question 3, you showed that if the points in S are chosen uniformly at random from the unit square $[0,1]^2$ (i.e. all coordinates are independent random numbers in $[0,1]$) then the expected number of un-dominated points in S is $O(lg\,|S|)$. This raises the question as to whether the $\Theta(n\,lg\,n)$-time divide-and-conquer algorithm that you developed in Assignment 3 could be made to run more efficiently when the number of un-dominated points turns out to be significantly smaller than n. We would like to express the cost of solving the un-dominated points problem on both parameters: n, the number of input points, and u, the number of output (un-dominated) points.

   a) Describe an iterative algorithm that finds un-dominated points, in sequence, at a cost of $O(n)$ time per un-dominated point.

   The algorithm in part (a) solves the un-dominated points problem in $O(n \cdot u)$ time in total. We can improve this by modifying our earlier divide-and-conquer algorithm as follows: (i) split the point set S in half by x-coordinates (as before) forming sets S_L (the points with smaller x-coordinates and S_R (the points with larger x-coordinates), (ii) recursively find the un-dominated points in S_R, (iii) remove from S_L all points dominated by points in S_R, (iv) recursively find the un-dominated points among the points remaining in S_L, and (v) combine the solutions to the subproblems to form a solution to the original.

   b) Denote by u_R (respectively, u_L) the number of un-dominated points returned by the recursive call on set S_R (respectively, S_L). Show that $u_R + u_L = u$.

   c) Give a recurrence for $T(n, u)$ the worst-case cost of finding the u un-dominated points in an input set of size n, using the algorithm described above.

   d) Prove that $T(n, u)$ is $O(n\,lg\,u)$.

2) Suppose we are given a set of keys (in alphabetical order) $\{x_1, \ldots, x_n\}$, and we know that, for $1 \leq i < n$, the access frequency of key x_i is greater than the *total* access frequency of the keys $x_{i+1}, \ldots, x_n$.

   a) Describe the optimal binary search tree T for this set of keys. Prove that your answer is correct.

b) It will be apparent that for a fixed fraction of the nodes the access cost is $\Omega(n)$. Nevertheless, you should prove that the *expected* access cost of a successful search (expected number of nodes explored) in T is at most 2.

c) Describe how to modify the tree T in part (a), to form a new binary search tree T' on the same set of keys, so that (i) for $1 \le i \le n$, the depth of x_i in the modified tree T' is at most one more than the depth of x_i in T, and (ii) the maximum depth of any node in T' is at most $1 + lg\, n$. What can you say about the worst case and expected access cost in T'?

3) We saw in class that the move-to-front heuristic, for adaptive list-structured dictionaries, can be used in data compression. The basic idea is that we maintain a list L of all of the characters in a given string σ of text, ordered according to the move-to-front strategy (i.e. as we encounter the next character in σ we move it to the front of L). We encode each successive character by the (integer) position it occupies in L (before it is moved). To make this encoding efficient, we need to encode *small* integers (the position of frequently occurring characters) with small codes. For this we can do something reminiscent of our scheme for exploiting locality of search in sorted arrays....

Suppose that a is a positive integer. The following algorithm outputs a binary sequence that we will interpret as the *encoding* of a.

```
phase 1...
    $r \leftarrow 1$
    **while** $a \ge 2r$ **do**              invariant: $a \le r$
        **output** 1
        $r \leftarrow r + r$
    **output** 0
                                    assertion: $r = 2^{\lfloor lg\, a \rfloor}$
phase 2...
    $low \leftarrow r$; $high \leftarrow 2r$; $gap \leftarrow r$
    **while** $gap > 1$ **do**              invariant: $low \le a < high = low + gap$
        $mid \leftarrow low + gap/2$
        **if** $a < mid$
            **then do**
                **output** 0
                $high \leftarrow mid$
            **else do**
                **output** 1
                $low \leftarrow mid$
        $gap \leftarrow gap/2$
```

a) Prove that the algorithm provides a proper encoding (i.e. no two distinct numbers are given the same encoding).

b) Describe in pseudocode an efficient *decoding* procedure (taking the encoding of a and returning a).

c) Describe in pseudocode, with sufficiently many comments/invariants to make the correctness clear, the enhanced encoding scheme proposed in the BONUS part of the midterm question:\ [BONUS] Since the first phase of the algorithm outputs a sequence of $\lfloor lg\, a \rfloor$ 1's, followed by a zero, we could view this as the unary encoding of the number $\lfloor lg\, a \rfloor$. If we applied the same idea used in this algorithm to create a more compact encoding of $\lfloor lg\, a \rfloor$, what would the total length of the encoding of a become?