The University of British Columbia

Department of Computer Science

Computer Science 420—Advanced Algorithm Design and Analysis

Homework Assignment 3
More extrema-finding and dictionaries

Due: 2015 January 29

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) Suppose that we are given 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 $(X[i], Y[i])$). We say that a point (x, y) is *dominated* by point (p, q), if $x < p$ and $y \leq q$, or $x = p$ and $y < q$. The un-dominated points can be viewed as the *two-dimensional extrema (maxima)* within S.

   The *un-dominated points problem* takes as input the two arrays $X[1:n]$ and $Y[1:n]$ and outputs the sequence consisting of all of the un-dominated points in the input set, ordered by x-coordinate. (Note: neither X nor Y is assumed to be sorted.)

   A natural approach to the *un-dominated points problem* that should suggest itself is to use divide-and conquer: (i) split the point set in half by x-coordinates, (ii) recursively find the un-dominated points in each of the subsets, and (iii) combine the solutions to the subproblems to form a solution to the original.

   a) Let $T(n)$ to denote the worst-case cost (using this approach) of solving a problem instance with n points, and let $M(n_1, n_2)$ the cost of combining the solutions of subproblems of size n_1 and n_2, as in step (iii). Give a recurrence relation for $T(n)$.

   b) Assuming that the solution to any subproblem is presented as a list of points sorted by x-coordinates, describe how step (iii) can be implemented in a way that $M(n_1, n_2)$ is $O(n_1 + n_2)$. What is the solution to the recurrence in part (a), in this case? Explain.

   c) Argue that the un-dominated points problem is at least as hard as sorting, in the sense that any algorithm that solves the un-dominated points problem could be used, with just $O(n)$ overhead, to solve an arbitrary instance of the sorting problem (of size n). Given this, what, if anything, can you conclude about the worst-case cost of solving the un-dominated points problem?

2) Let S be a set formed by drawing n numbers *uniformly at random* from the interval $[0, 1)$, and suppose that the elements of S are stored in sorted (increasing) order in the array $D[1:n]$. If we are given a query value q and we want to determine the element $D[\ell_q]$ that is closest in value to q, we can take advantage of our knowledge of the key distribution to do better than straightforward binary search in array D.

   a) Let X_i be the indicator random variable that take the value 1 just when the i-th number drawn for set S has a value less than or equal to q. What is the expected value of X_i? What is the expected value of the total number of elements of S with value less than or equal to q?

   b) Using the results from part (a), express, as a function of n and q the location e_q of D at which we *expect* to find the element closest to the query q. Explain.

c) If we expect to find the element closest to q in $D[e_q]$ and the element closest to q is actually located in $D[l_q]$, it is not too hard to show (*you do not have to do it*) that $\Pr\{|l_q - e_q| \geq s\sqrt{n}\} \leq 1/(4s^2)$, for any $s > 0$. Using this fact, show that the expected value of $|l_q - e_q|/\sqrt{n}$ is less than 2.

d) Use the result from part (c) to analyse the expected cost of determining a subarray $D[low : high]$ of size $\sqrt{n}$ containing $D[l_q]$ using the pseudocode below:

```
if $x > D[e_x]$
  then
        $low \leftarrow e_x$; $high \leftarrow e_x + \sqrt{n} - 1$
        while $(high \leq n)$ & $(x > D[high])$
              $low \leftarrow high$
              $high \leftarrow low + \sqrt{n} - 1$
  else
        $high \leftarrow e_x$; $low \leftarrow e_x - \sqrt{n} + 1$
        while $(low \geq 1)$ & $(x \leq D[low])$
              $high \leftarrow low$
              $low \leftarrow high - \sqrt{n} + 1$
```

e) If we apply this idea recursively to search for the element closest to q in the subarray $D[low : high]$, we get a recurrence: $T(n) = \Theta(1) + T(\sqrt{n})$, for $T(n)$, the expected cost of finding the element closest to q in an array of size n. What is the asymptotic solution of this recurrence (assuming, of course, that $T(n)$ is constant, for $n < 4$)?

3) Suppose that $A[1 : n]$ is an array of real numbers, *sorted in increasing order*. It is claimed that the following procedure determines if there exists a pair of (not necessarily distinct) elements of A that sum to zero.

```
$i \leftarrow 1$
$j \leftarrow n$
while $i \leq j$
      if $A[i] + A[j] > 0$
        then $j \leftarrow j - 1$
        else if $A[i] + A[j] < 0$
              then $i \leftarrow i + 1$
              else return TRUE
return FALSE
```

a) Analyse the worst-case cost of this procedure, as a function of n.

b) Prove that the procedure is correct. Specifically, show that the procedure returns TRUE if and only if there is a pair of indices i and j, where $i \leq j$ and $A[i] + A[j] = 0$.

c) Describe how you would generalize the procedure so that it runs in the same time and determines for some specified target value t (not necessarily 0) if there is a pair of numbers in A that sum to t. *Think about (but do not submit): How would you further modify the procedure so that it determines if there is a pair of distinct elements of A that sum to t?*

d) Describe how you would use the modified procedure from part (c) to construct a new procedure triple-sum-zero that efficiently determines if there is a *triple* of (not necessarily distinct) numbers in A that sum to zero. What is the worst-case cost of your procedure triple-sum-zero?