CS 466/666, Fall 2007 due Friday November 9
A. Lubiw

ASSIGNMENT 3

Do not copy from others, and acknowledge your sources.

1. [10 marks] The RSA cryptosystem requires large primes chosen at random. Give a randomized algorithm for generating a random t bit prime number. The highest order bit (the one in the t^{th} position) should be a 1. Analyze the expected run time of your algorithm. You will need the Prime Number Theorem, which states that $\pi(k)$, the number of distinct primes less than k, is asymptotically $k/\ln k$. (You don't need to prove the Prime Number Theorem.)

2. [10 marks] This problem is about implementing a counter using very limited space. Normally, an n-bit register can count from 0 to $2^n - 1$. Here we use randomness to count higher, with the trade-off that the count may not be exact.

   We will implement two operations, Increment and Count. Let r be the value stored in the register. Let c be number of calls to Increment (i.e. the true value the counter should have).

   (a) Consider the following implementation

   ```
   Increment
     toss a coin
     if Heads then r <-- r+1

   Count
     return 2r
   ```

   i. ~~Prove that the expected value of $2r$ is c.~~ Prove that the expected value returned by Count is c.

   (b) To count even further, we can use exponentiation instead. ~~Let Count return 2^{r-1}.~~ Let Count return $2^r - 1$.

   i. Describe how to implement Increment so that it increases r with probability 2^{-r}. (Hint: r coin tosses and no arithmetic.)

   ii. ~~Prove that the expected value of 2^{r-1} is c.~~ Prove that the expected value returned by Count is c.

   Hints: For both (a.i) and (b.ii), you will need $p_c(i)$, the probability that the register holds value i after c calls to Increment. For (a.i) you can write $p_c(i)$ explicitly, and plug in to the formula for expected value, obtaining a a sum that you may wish to look up or go back to your generating function notes for. However, for (b.ii) I suggest writing a recurrence relation for $p_c(i)$ in terms of $p_{c-1}(i)$ and $p_{c-1}(i-1)$, and then proving by induction that the expected value is as required. This same approach can be used for (a.i).

3. [20 marks] The convex hull of a set of n points in the plane is the smallest convex set containing the points. Informally, it is the polygon you get by putting an elastic band around the points. You can see examples and read about deterministic $O(n \log n)$ time convex hull algorithms in CLRS. This question is about a randomized incremental algorithm with expected run time

$O(n \log n)$. This algorithm isn't faster than the deterministic ones, but the point is that it generalizes easily to 3D with the same run time, which the deterministic ones don't.

The randomized incremental algorithm picks a random order $p_1, \ldots, p_n$ of the points, and adds the points in that order. Let C_i be the convex hull of $p_1, \ldots, p_i$. We will store C_i as a double linked list so that we can traverse it clockwise or counterclockwise. Initialize by computing the triangle C_3. Let q be a point interior to C_3. For simplicity's sake we will assume that no 3 points among $q, p_1, \ldots, p_n$ lie on a line.

```
(0)  for i = 4 . . n
(1)    if p_i is inside C_{i-1} then C_i <-- C_{i-1}
(2)    else
(3)      let e be the edge of C_{i-1} crossed by the line segment q, p_i
(4)      scan clockwise and counterclockwise around C_{i-1} from e to find
           the edges of C_i incident with p_i
(5)      construct C_i by adding these two edges and deleting the enclosed
           path of C_{i-1}
```

(a) [4 marks] Fill in the details of step (4) and (5), and argue that the worst case time spent by the algorithm not counting step (3) and the test in step (1) is $O(n)$. Hint: prove that the total number of edges added during the course of the algorithm is at most $2n$.

We implement step (3) by storing (and updating) the required information. In particular, for each point p_j we store the edge $e(p_j)$ where the line segment $p_j q$ crosses the current convex hull. If p_j is inside the current convex hull then $e(p_j)$ is null. With this information, both step (3) and the test in step (1) take constant time. To facilitate updating we also store for each edge e of the current convex hull the set $P(e)$ of all points p_j with $e(p_j) = e$.

(b) [2 marks] Show how to initialize the mappings $e()$ and $P()$ for C_3 in $O(n)$ time.

(c) [4 marks] Show how to update the mappings $e()$ and $P()$ when we go from C_{i-1} to C_i. Hint: for each edge that disappears from C_{i-1}, go through $P(e)$.

(d) [3 marks] Find an example set of points (for general n) and two orderings of these points, such that one ordering causes the algorithm to take $O(n)$ time, and the other ordering causes the algorithm to take $O(n^2)$ time.

(e) [7 marks] Do a backwards analysis to prove that the expected run time of the algorithm is $O(n \log n)$. Hint: We need to account for the work done in iteration i to update $e()$ and $P()$. Show that this is proportional to the number of points $p_j, j > i$, whose $e()$ value changes. Consider iteration i going backwards from C_i to C_{i-1}. A random point among $p_1, \ldots, p_i$ is removed. The $e()$ value of a specific point p_j changes iff line segment $p_j q$ crosses an edge of C_i that is removed as we step backwards to C_{i-1}. Compute the probability of this.