Introduction to Algorithms February 24, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser Handout 7

Problem Set 3

Reading: Chapters §8.1-8.3, 31.1-31.5, 31.7-31.8

There are **four** problems. Each problem is to be done on a **separate sheet** (or sheets) of paper. Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

You will often be called upon to "give an algorithm" to solve a certain problem. Giving an algorithm entails:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. A proof (or argument) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

It is also suggested that you include at least one worked example or diagram to show more precisely how your algorithm works. Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions. If you cannot solve a problem, give a brief summary of any partial results.

Problem 3-1. Min and Max

Suppose we wish to find both the minimum and maximum values in an array of n distinct elements. To individually find either the minimum or maximum, there is clearly a $n - 1$ lower bound on comparisons, since we must compare every element at least once. We can save a comparison by first scanning for the minimum, removing it, then scanning for the maximum. This takes a total of $2n - 3$ comparisons.

Based on this observation, prove the following statement **true** or **false**:

"It takes at least $2n - c$ comparisons, for some constant c, to find both the minimum and the maximum in an array of n distinct elements."

Problem 3-2. Monotone Priority Queues

A "monotone priority queue" (MPQ) is a data structure that supports the following operations:

- MIN(Q) - Returns the minimum element in Q. The minimum of a new, empty MPQ is initially $-\infty$. Otherwise, the minimum of an empty MPQ is the last element deleted.
- INSERT(Q, x) - Inserts x into Q given that $x \geq$MIN(Q). If $x <$MIN(Q), then the MPQ is not modified.

- DELETE-MIN(Q) - If Q is empty, returns the minimum. Otherwise, removes and returns the minimum from Q. If the queue is empty after the operation, the last deleted value remains the minimum. In other words, the minimum value is monotonically increasing and does not reset when the MPQ is empty.

For this problem, assume that x is an integer in the range $[0, k]$ for some fixed integer value k.

(a) Implement a monotone priority queue that takes $O(m \log m)$-time to perform m operations starting with an empty data structure.

(b) Give an implementation of a monotone priority queue that takes $O(m + k)$ time to perform m total operations. Hint: Use an idea from COUNTING-SORT.

(c) 6.046 student Ben K. Bitdiddle has invented a MPQ that operates on any *totally ordered* set, rather than just integers in the range $[0, k]$. A total ordering defines a $\leq$ relation for all pairs of elelments in a set.

Ben claims that his MPQ can perform m operations in $O(m \log \log m)$-time. Ben's classmate Alyssa P. Hacker quickly dismisses his claim as impossible. Explain who is correct and prove why.

Problem 3-3. Operations on Elliptic Curves

Throughout this problem we will be discussing operations on elliptic curve points. You do not need to know any specifics about elliptic curves or their operations. The basic operand will be points denoted by bold capital letters, for example **P**.

You are given a subroutine ADD which can add points, for example ADD(**P**,**Q**) = **P** + **Q**. You may assume that ADD runs in $O(n)$-time, where **P** has an n-bit representation.

Multiplying an elliptic curve point **P** by a scalar s is equivalent to adding **P** to itself $(s - 1)$ times. That is, 2**P** = **P** + **P**, 3**P** = **P** + **P** + **P**, etc. By this definition, (s**P** + t**P**) = $(s + t)$**P** = (t**P** + s**P**).

(a) Suppose you are given a k-bit integer s and a point **P** as inputs. Give a $\Theta(n2^k)$-time scalar multiplication algorithm that computes s**P** using only calls to ADD.

(b) Give an $\Theta(nk)$-time scalar multiplication algorithm.

(c) Ben K. Bitdiddle, notices that his solution to part (b) always makes between k and $2k$ calls to ADD. He thinks he can improve on this and writes a point doubling procedure DOUBLE that runs in $O(1)$ time. The output of DOUBLE(**P**) is 2**P**.

Rewrite your solution to part (b) using DOUBLE. What are the new upper and lower bounds on the runtime? What is the expected number of calls to ADD if s is chosen uniformly at random from $\{0, 1\}^k$?

(d) Ben gets the idea to pre-compute the values **P**, 2**P**, 3**P**, ..., $(2^d - 1)$**P** and store them in an array A such that $A[i] = i$**P**. Suppose you naively fill in the array A in $O(n2^d)$-time by repeated point addition. Give an $O(\frac{nk}{d})$-time scalar multiplication algorithm. You may assume that d divides $(k - 1)$ and may use both DOUBLE and ADD in your code.

(e) Give a value of d such that the algorithm in part (d) runs in $o(nk)$. Include both the time it takes to fill A and compute $s\mathbf{P}$, i.e. $O(n2^d + \frac{nk}{d})$.

Problem 3-4. Man on the Moon

Alyssa (A) wishes to determine whether her n-bit string a is the same as Ben's (B) n-bit string b. Unfortunately, Ben lives on the moon and communication costs are very high. Ben devises a scheme to determine with high probability whether or not $a = b$, while minimizing communication. Let a_i and b_i denote the ith bits of a and b's respective representations:

1. A picks a prime p such that $n^2 < p < 2n^2$.
2. A defines a n-degree polynomial over $\mathbb{Z}_p$, denoted $a(x) = (\sum_{i=0}^{n} a_i x^i) \bmod p$.
3. A picks a random $x \in \mathbb{Z}_p$ and computes $a(x)$, and sends $a(x)$, x, and p to B.
4. B defines a n-degree polynomial over $\mathbb{Z}_p$, denoted $b(x) = (\sum_{i=0}^{n} b_i x^i) \bmod p$.
5. B computes $b(x)$ and accepts if $a(x) = b(x)$.

(a) Given that a n-degree polynomial can have at most n roots, if $a \neq b$ what is the maximum probability that Ben accepts?

(b) Give an explicit upper bound (in terms of n) on the number of bits transmitted in this scheme. Do not give an asymptotic upper bound, but rather an actual function, e.g. $5n^2$ instead of $O(n^2)$.

Alyssa suggests a second scheme:

1. Repeat k times:
   (a) A picks a prime p uniformly at random from the range $[1, W]$.
   (b) A sends p and $(a \bmod p)$ to B.
   (c) B rejects if $(b \bmod p) \neq (a \bmod p)$.
2. B accepts if $(b \bmod p) = (a \bmod p)$ for all steps.

(c) Assume that there are L primes less than W, that is $(p_1 < p_2 < \ldots < p_L < W)$, and that $(\Pi_{i=1}^{L} p_i) = P_L > 2^n$. If $a \neq b$, give an upper bound on the probability, in terms of k and L, that $(a \bmod p) = (b \bmod p)$ for all k rounds. You will need to use the Chinese Remainder Theorem.

(d) Using the the Prime number theorem in CLRS Section 31.8, upper bound the probability of failing all k rounds of the protocol in terms of k and W?