

CS 373U: Combinatorial Algorithms, Spring 2004

Homework 3

Due Friday, March 12, 2004 at noon

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

-
- For each numbered problem, if you use more than one page, staple all those pages together. **Please do *not* staple your entire homework together.** This will allow us to more easily distribute the problems to the graders. Remember to print the name and NetID of every member of your group, as well as the assignment and problem numbers, on every page you submit. You do not need to turn in this cover page.
 - This homework is challenging. You might want to start early.
-

#	1	2	3	4	5	6*	Total
Score							
Grader							

1. Let S be a set of n points in the plane. A point p in S is called *Pareto-optimal* if no other point in S is both above and to the right of p .
 - (a) Describe and analyze a deterministic algorithm that computes the Pareto-optimal points in S in $O(n \log n)$ time.
 - (b) Suppose each point in S is chosen independently and uniformly at random from the unit square $[0, 1] \times [0, 1]$. What is the *exact* expected number of Pareto-optimal points in S ?

2. Suppose we have an oracle $\text{RANDOM}(k)$ that returns an integer chosen independently and uniformly at random from the set $\{1, \dots, k\}$, where k is the input parameter; RANDOM is our only source of random bits. We wish to write an efficient function $\text{RANDOMPERMUTATION}(n)$ that returns a permutation of the integers $\langle 1, \dots, n \rangle$ chosen uniformly at random.
 - (a) Consider the following implementation of RANDOMPERMUTATION .

```

RANDOMPERMUTATION(n):
  for i = 1 to n
    π[i] ← NULL
  for i = 1 to n
    j ← RANDOM(n)
    while (π[j] != NULL)
      j ← RANDOM(n)
    π[j] ← i
  return π

```

Prove that this algorithm is correct. Analyze its expected runtime.

- (b) Consider the following partial implementation of RANDOMPERMUTATION .

```

RANDOMPERMUTATION(n):
  for i = 1 to n
    A[i] ← RANDOM(n)
  π ← SOMEFUNCTION(A)
  return π

```

Prove that if the subroutine SOMEFUNCTION is deterministic, then this algorithm cannot be correct. [Hint: There is a one-line proof.]

- (c) Consider a correct implementation of $\text{RANDOMPERMUTATION}(n)$ with the following property: whenever it calls $\text{RANDOM}(k)$, the argument k is at most m . Prove that this algorithm *always* calls RANDOM at least $\Omega(\frac{n \log n}{\log m})$ times.
- (d) Describe and analyze an implementation of RANDOMPERMUTATION that runs in expected worst-case time $O(n)$.

3. A *meldable priority queue* stores a set of keys from some totally-ordered universe (such as the integers) and supports the following operations:

- MAKEQUEUE: Return a new priority queue containing the empty set.
- FINDMIN(Q): Return the smallest element of Q (if any).
- DELETEMIN(Q): Remove the smallest element in Q (if any).
- INSERT(Q, x): Insert element x into Q , if it is not already there.
- DECREASEKEY(Q, x, y): Replace an element $x \in Q$ with a smaller key y . (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q containing x .
- DELETE(Q, x): Delete the element $x \in Q$. The input is a pointer directly to the node in Q containing x .
- MELD(Q_1, Q_2): Return a new priority queue containing all the elements of Q_1 and Q_2 ; this operation destroys Q_1 and Q_2 .

A simple way to implement such a data structure is to use a heap-ordered binary tree, where each node stores a key, along with pointers to its parent and two children. MELD can be implemented using the following randomized algorithm:

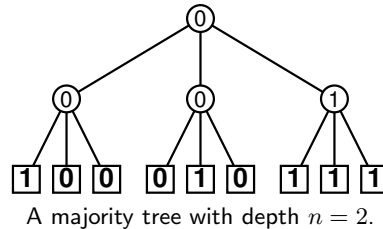
```

MELD( $Q_1, Q_2$ ):
  if  $Q_1$  is empty return  $Q_2$ 
  if  $Q_2$  is empty return  $Q_1$ 
  if  $key(Q_1) > key(Q_2)$ 
    swap  $Q_1 \leftrightarrow Q_2$ 
  with probability 1/2
     $left(Q_1) \leftarrow \text{MELD}(left(Q_1), Q_2)$ 
  else
     $right(Q_1) \leftarrow \text{MELD}(right(Q_1), Q_2)$ 
  return  $Q_1$ 

```

- (a) Prove that for *any* heap-ordered binary trees Q_1 and Q_2 (not just those constructed by the operations listed above), the expected running time of MELD(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. [Hint: How long is a random root-to-leaf path in an n -node binary tree if each left/right choice is made with equal probability?]
- (b) **[Extra credit]** Prove that MELD(Q_1, Q_2) runs in $O(\log n)$ time with high probability.
- (c) Show that each of the other meldable priority queue operations can be implemented with at most one call to MELD and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ time with high probability.)

4. A *majority tree* is a complete binary tree with depth n , where every leaf is labeled either 0 or 1. The *value* of a leaf is its label; the *value* of any internal node is the majority of the values of its three children. Consider the problem of computing the value of the root of a majority tree, given the sequence of 3^n leaf labels as input. For example, if $n = 2$ and the leaves are labeled 1, 0, 0, 0, 1, 0, 1, 1, 1, the root has value 0.



- (a) Prove that *any* deterministic algorithm that computes the value of the root of a majority tree *must* examine every leaf. [Hint: Consider the special case $n = 1$. Recurse.]
- (b) Describe and analyze a randomized algorithm that computes the value of the root in worst-case expected time $O(c^n)$ for some constant $c < 3$. [Hint: Consider the special case $n = 1$. Recurse.]
5. Suppose n lights labeled $0, \dots, n - 1$ are placed clockwise around a circle. Initially, each light is set to the off position. Consider the following random process.

LIGHTTHECIRCLE(n):
 $k \leftarrow 0$
 turn on light 0
 while at least one light is off
 with probability $1/2$
 $k \leftarrow (k + 1) \bmod n$
 else
 $k \leftarrow (k - 1) \bmod n$
 if light k is off, turn it on

Let $p(i, n)$ be the probability that light i is the last to be turned on by LIGHTTHECIRCLE($n, 0$). For example, $p(0, 2) = 0$ and $p(1, 2) = 1$. Find an exact closed-form expression for $p(i, n)$ in terms of n and i . Prove your answer is correct.

6. [Extra Credit] Let G be a *bipartite* graph on n vertices. Each vertex v has an associated set $C(v)$ of $\lg 2n$ colors with which v is compatible. We wish to find a coloring of the vertices in G so that every vertex v is assigned a color from its set $C(v)$ and no edge has the same color at both ends. Describe and analyze a randomized algorithm that computes such a coloring in expected worst-case time $O(n \log^2 n)$. [Hint: For any events A and B , $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$.]