Algorithms Qualifying Exam

Fall 2000

9-12am, Thursday August 24
Closed book exam

Problem 1

Show that the recurrence $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + b\log n & \text{if } n \geq 2 \end{cases}$

where b is a positive constant has solution $T(n) = O(n)$.

Problem 2

In this problem you can assume that $\Omega(n \log n)$ is a lower bound—in the (algebraic) decision tree model—for the following problems; Sorting, Element Distinctness, the $A \cap B = \emptyset$ problem, the $A = B$ problem, the $A \subset B$ problem.

Prove that $\Omega(n \log n)$ is a lower bound for the following problem:
Given reals $x_1, x_2, ..., x_n$, decide if the following holds for all $i \neq j$: $(|j-i| \bmod 2 = 0) \Rightarrow (x_i \neq x_j)$.

Problem 3

In this problem we consider the O-1 KNAPSACK PROBLEM: Given n items, with item i being worth $v[i]$ dollars and having weight $w[i]$ pounds, fill a knapsack of capacity m pounds with the maximal possible value.

The algorithm Knapsack(i,j) below returns the maximal value obtainable when filling a knapsack of capacity j using items among items 1 through i (Knapsack(n,m) solves our problem). The algorithm works by recursively computing the best solution obtainable *with* the last item and the best solution obtainable *without* the last item, and returning the best of them.

```
Knapsack(i,j)

  IF w[i] <= j THEN
    with = v[i] + Knapsack(i-1, j-w[i])
  ELSE
    with = 0
  FI
  without = Knapsack(i-1,j)
  RETURN max{with, without}

END
```

a) Show that the running time T of Knapsack(n, m) is exponential in n or m.

b) Describe and analyze an improved algorithm for computing the value of the optimal solution.

Problem 4

The *maximum partial sum* problem is defined as follows. Given an array $A[1..n]$ of integers, find values of i and j with $1 \leq i \leq j \leq n$ such that

$$\sum_{k=i}^{j} A[k]$$

is maximized. Devise an algorithm for solving the maximum partial sum problem in time $O(n \log n)$.

Problem 5

Consider a list data structure with the following three operations:

- `insert`(x, y): Insert a new record y after the record x in the list.
- `delete`(x): Delete a record x from the list.
- `order`(x, y): Return true if and only if record x is before record y in the list.

We implement the structure using a *simple linked list* of records as follows (initially the structure contains only a header and tailer record):

- Initially, assign an ID (identification number) 0 to the header record and an ID 1 to the tailer record.
- To `insert` a record between consecutive records with IDs α and β, assign the new record the ID $\lfloor \frac{\alpha+\beta}{2} \rfloor$. If the new ID is not unique, then assign new IDs to each record in the list according to the scheme that the ith record gets ID $2^n i$, where n is the length of the list.
- To `delete` a record, search for it from the head of the list and remove it without changing any IDs.
- To determine the `order` of two records, compare their IDs.

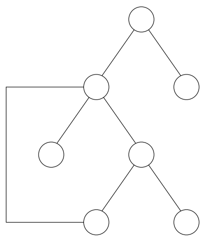
We assume that when performing `insert`(x, y) or `order`(x, y) we are given pointers to elements x and y.

a) What is the worst-case running time of each of the operations?

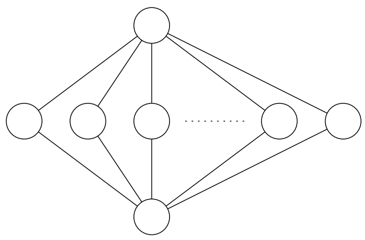
b) Prove tight bounds on the amortized running time of each of the operations.

Problem 6

Consider an undirected weighted graph which is formed by taking a binary tree and adding an edge from exactly one of the leaves to another node in the tree. We call such a graph a *loop-tree*. An example of a loop-tree could be the following:



Consider also a so-called *pole-graph* which is an undirected weighted graph consisting of two poles connected through a layer of nodes as follows:



Let n be the number of vertices in a loop- or pole-graph and assume that the graphs are given in the normal edge-list representation without any extra information. In particular, the representation does not contain information about which vertex is the root (in the case of a loop-graph) or which vertices are the poles (in the case of a pole-graph).

a) How long time would it take Prim's or Kruskal's algorithms to find the minimal spanning tree of a loop- or pole-graph?

b) Describe and analyze a more efficient algorithm for finding the minimal spanning tree of a loop-graph. Remember to prove that the algorithm is correct.

c) Describe and analyze a more efficient algorithm for finding the minimal spanning tree of a pole-graph. Remember to prove that the algorithm is correct.