CORE EXAMINATION : SOLUTIONS
Department of Computer Science
New York University
January 16, 2004

Basic Algorithms

Question 1

Suppose you have a data structure and corresponding algorithms that implement a "lookup table" of objects. Assume that objects are ordered by some abstract comparison operator "<", and that it is only through this operator that objects are accessed by the algorithms. Suppose that the data structure supports the following operations:

- Create an empty table T — *assume that this operation performs no comparisons.*
- Create a table T containing a single object x — *assume that this operation performs no comparisons.*
- Insert an object x into a table T.
- Search for an object x in a table T.
- Merge two tables T_1, T_2, creating a new table containing all the objects in both tables T_1 and T_2 — *assume that this operation performs* $O(n_1 + n_2)$ *comparisons of objects, where* n_i *is the number of objects in table* T_i, $i = 1, 2$.
- Output all the objects in a table T in sorted order — *assume that this operation performs no comparisons.*

1. Design and analyze an algorithm for the following problem. Your algorithm should take as input tables $T_1, \ldots, T_m$, and should output a table T containing all the objects in tables $T_1, \ldots, T_m$. Your algorithm should use $O(n \log m)$ comparisons of objects, where $n = n_1 + \cdots + n_m$ and n_i is the number of objects stored in table T_i for $i = 1, \ldots, m$. Be sure to argue that your algorithm is correct and that it runs in the stated time bound.
2. Professor Smith has announced an algorithm to solve the problem in part (1) using $O(n \log \log m)$ comparisons. Can Professor Smith's claim possibly be true? Why or why not?

Solution to part (1). This is just a simple application of divide and conquer. Split the tables into two subsets, $T_1, \ldots, T_k$ and $T_{k+1}, \ldots, T_m$, where $k = \lfloor m/2 \rfloor$, recursively merge $T_1, \ldots, T_k$ into T' and $T_{k+1}, \ldots, T_m$ into T'', and then merge T' and T'' into T using the built-in "merge" operation. The depth of the recursion tree is $O(\log m)$. The number of comparisons done at each level of the recursion tree is $O(n)$.

Solution to part (2). This is impossible, as it would violate the $\Omega(n \log n)$ lower bound for comparison-based sorting. Indeed, using Prof. Smith's algorithm, we could sort $x_1, \ldots, x_n$ using $O(n \log \log n)$ comparisons as follows:

(i) create n tables $T_1, \ldots, T_n$, where T_i contains x_i, using the built-in "create" operation;

(ii) apply the Smith merging algorithm to merge $T_1, \ldots, T_n$ into a single table T;

(iii) output the contents of T in sorted order, using the built-in "output" operation.

Steps (i) and (iii) use no comparisons, while step (ii) uses just $O(n \log \log n)$ comparisons.

Question 2

Design and analyze an efficient algorithm for the following problem. The input consists of two strings, $s[1 \ldots m]$ and $t[1 \ldots m]$, of letters 'a', ..., 'z'. Also included in the input are two "cost vectors," $C[1 \ldots m]$ and $D[1 \ldots n]$, of non-negative numbers. The goal of your algorithm is to make the two string s and t equal by deleting some letters from both strings, while minimizing the *deletion cost*, which is defined as follows: if your algorithm deletes letters from s at positions $i_1, \ldots, i_p$ and letters from t at positions $j_1, \ldots, j_q$, then the deletion cost is defined to be

$$C[i_1] + \cdots C[i_p] + D[j_1] + \cdots + D[j_q].$$

In words, the cost of deleting any single letter depends on its position with the string s or t, as determined by the cost vectors C and D, and the total deletion cost is defined to be the sum of the costs of deleting individual letters.

To simplify matters, it is sufficient if your algorithm just returns the *cost* of the minimal cost deletion sequence that makes s and t equal, and need not return the deletion sequence itself. Make sure you argue the correctness of your algorithm, and state and prove a bound on its running time in terms of m and n.

Hint: use dynamic programming, defining the following variables. If s has length m and t has length n, define $E(i,j)$, for $i = 0,\ldots,m$ and $j = 0,\ldots,n$, to be the minimal cost of making $s[1\ldots i]$ equal to $t[1\ldots j]$ by deleting letters. Write down equations defining $E(i,j)$ in terms of $E(i-1,j)$, $E(i-1,j-1)$, $E(i,j-1)$, $C[i]$, and $D[j]$, considering the following cases:

- $i = 0$ and $j = 0$;
- $i > 0$ and $j = 0$;
- $i = 0$ and $j > 0$;
- $i > 0$, $j > 0$, and $s[i] = t[j]$;
- $i > 0$, $j > 0$, and $s[i] \neq t[j]$.

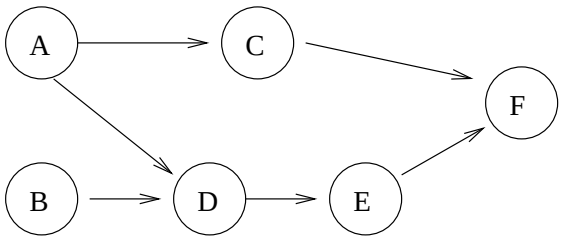
Solution. For $i = 0,\ldots,m$ and $j = 0,\ldots,n$, we have

$$E(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0; \\ E(i-1,j) + C[i] & \text{if } i > 0 \text{ and } j = 0; \\ E(i,j-1) + D[j] & \text{if } i = 0 \text{ and } j > 0; \\ \min\{E(i-1,j-1), & \\ \quad E(i-1,j) + C[i], E(i,j-1) + D[j]\} & \text{if } i > 0,\ j > 0, \text{ and } s[i] = t[j]; \\ \min\{E(i-1,j) + C[i], E(i,j-1) + D[j]\} & \text{if } i > 0,\ j > 0, \text{ and } s[i] \neq t[j]. \end{cases}$$

In the fourth case, we have a choice between keeping both $s[i]$ and $t[j]$, or deleting one of them — in an optimal solution will never delete both $s[i]$ and $t[j]$ when they are equal; however, since the deletion cost is position dependent, it may make sense to delete one of them. As for the running time, we have $O(mn)$ variables, and each them takes time $O(1)$ to evaluate, and so the total running time is $O(mn)$.

Question 3

Certain partial orderings can be represented using an *interval representation*. In an interval representation, each element A of the partial order is associated with a pair of integers $[A_l, A_h]$. The rule is that $A < B$ in the partial ordering if and only if $A_h < B_l$. For example, the partial ordering illustrated below can be represented using the intervals A → [0,2]; B → [0,4]; C → [3,8]; D → [5,6]; E → [7,8]; F → [9,10]



A. (3 points) Prove that there does *not* exist any interval representation for the following partial ordering:

```
A -------> B

C -------> D
```

B. (2 points) Design a $\Theta(N \log N)$ algorithm for computing a topological sort from an interval representation of a partial ordering.

C. (5 points) Design a $\Theta(N^2)$ algorithm for the following problem: The input is a partial ordering on N objects, expressed as an $N \times N$ Boolean matrix M, where $M[u, v] =$ T if u is constrained to precede v. The desired output is an interval representation for the partial ordering, if one exists. (Hint: Use a graph whose nodes are the proposed bounds of the intervals, A_l and A_h.)

For example, the input corresponding to the figure above would be the matrix

	A	B	C	D	E	F
A:	F	F	T	T	T	T
B:	F	F	F	T	T	T
C:	F	F	F	F	F	T
D:	F	F	F	F	T	T
E:	F	F	F	F	F	T
F:	F	F	F	F	F	F

Solution:

A. Since A precedes B, we must have $A_h < B_l$. Since C precedes D, we must have $C_h < D_l$. But then, if $A_h \leq C_h$, then $A_h < D_l$, so A would be constrained to precede D, and if $A_h > C_h$, then $C_h < B_l$, so C would be constrained to precede B.

B. For each element X, pick a value X.val between X_l and X_h. Sort the elements in increasing order of X.val. If U is constrained to precede V, then U.val must be less than V.val.

C. Construct a graph G whose nodes are the lower and upper bounds of each elements, and whose arcs are defined as follows:

```
{ for each pair of elements U,V,
    if (M[U,V]) then construct an arc from U.upper to V.lower;
    else if M[V,U] then construct an arc from V.upper to U.lower;
    else {  /* U and V are unordered */
```

```
construct an arc from U.lower to V.upper;
construct an arc from V.lower to U.upper;
   }
```

Then do a topological sort on G, and label each node with the index in the topological sort.