UNIVERSITY OF ALBERTA
CMPUT 204 (Session A2)

Term Test 2, October 2004

Duration: 50 min.

Student Number: |_|_|_|_|_|_|_|_|_|

Last Name: _____

First Name: _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above.)

Aids allowed: NONE

1: _____/10

2: _____/15

3: _____/20

TOTAL: _____/45

Good Luck! PLEASE HAND IN

Question 1. [10 MARKS]

Consider array $A = [5, 2, 6, 1, 3, 7, 4]$ which has size 7.

Part (a) [5 MARKS]

Recall the procedure `Partition` in the `QuickSort` algorithm. Show what would be the result of calling `Partition`$(A, 1, 7)$.

Solution: A will be: $[2, 1, 3, 4, 6, 7, 5]$.

Part (b) [5 MARKS]

Show what would be the result of calling `Build-Max-Heap`(A) (with the original array A).

Solution: A will be: $[7, 3, 6, 1, 2, 5, 4]$.

Question 2. [15 MARKS]

Suppose that we are given d (non-empty) arrays $A_1, A_2, \ldots, A_d$ of elements such that each array A_i is already sorted and there is a total of n elements in the arrays (arrays may have different lengths). Design an $O(n \lg d)$ algorithm to merge all the arrays into one sorted array. Explain the data structure you use, the algorithm, and the running time. (**Hint:** Use priority queues)

Solution: We are going to merge these arrays into array $A'[1 \ldots n]$. If we do the merge in a naive way, at each step i to find the appropriate element to be placed in $A'[i]$, we have to compare (up to) d elements. This will take $O(d)$ if we use linear search and the overall time will be $O(nd)$.

To speedup the search at each step, we are going to use priority queues based on Max-Heaps. Suppose that for every $1 \leq i \leq d$, l_i is the length of array A_i. Consider the elements $A_1[l_1], A_2[l_2], \ldots, A_d[l_d]$ (i.e. the last elements) and build a Max-Heap H with these elements. This takes $O(d)$ time which is in $O(n)$ since $n \geq d$, because each array is non-empty. With each element in H we also store the index of the array from which that element is coming from. For example, element $A_5[l_5]$ will have index 5 (because it is coming from A_5).

The largest element among all the n elements is the largest of $A_1[l_1], \ldots, A_d[l_d]$ and since they are in a heap (H), it is enough to call `Heap-Extract-Max`(H) and place that element in $A'[n]$. If the index of this element was i (i.e. it belongs to array A_i) then we insert the second last element of A_i into H by calling `Insert`, and then call `Heap-Extract-Max`(H) again, and so on. For each element to be placed in the correct position in A' we call each of `Heap-Extract-Max` and `Heap-Insert` once and the heap size is always at most d. Thus each of these takes $O(\lg d)$ time. This is the time required to place one element into array A'. Since there are n elements in total, the running time is $O(n \lg d)$, plus the time required to build the heap at the first place which was $O(n)$. Thus the overall time will be $O(n \lg d)$.

Question 3. [20 MARKS]

In lectures we have seen the problem of Longest Common Subsequence (LCS), where given two sequences X and Y we wanted to find a LCS of X and Y (a subsequence of a sequence S is obtained by deleting some of the letters in S).

Given two sequences X and Y, we say Z is a common *supersequence* of them if both X and Y are subsequences of Z. For instance, *abbacd* is a supersequence of *aba* and *bbcd*.

Given $X = x_1x_2 \dots x_n$ and $Y = y_1y_2 \dots y_m$, the goal is to find a Shortest Common Supersequence (SCS) of X and Y using dynamic programming. To simplify the problem, assume that we want to find only the *length* of a SCS of X and Y.

Part (a) [6 MARKS]

Define an appropriate array/table (corresponding to the subproblems) and say how to extract the solution from the array.

Solution: For any $0 \leq i \leq n$ and $0 \leq j \leq m$ let X_i denote the sequence $x_1, \dots, x_i$ and Y_j denote the sequence $y_1, \dots, y_j$, respectively. Note that $X_n = X$ and $Y_m = Y$. We define $S[i, j]$ $(0 \leq i \leq n,\ 0 \leq j \leq m)$ to be the length of SCS of X_i and Y_j. By this definition, the length of SCS of X and Y will be $S[n, m]$.

Part (b) [14 MARKS]

Give a recurrence (including the base cases) to compute the entries of your array. Briefly explain how you derive this recurrence.

Solution: Clearly the SCS of the empty string and any string T is T itself. Therefore for any value of $0 \leq i \leq n$ and $0 \leq j \leq m$: $S[i, 0] = i$ and $S[0, j] = j$.

If $i, j \geq 1$ then to compute $S[i, j]$ we consider the last elements of X_i and Y_j (i.e. x_i and y_j):

- If $x_i = y_j$ then they can be paired with the same element of an SCS of X_i and Y_j. In other words, adding x_i to the SCS of X_{i-1} and Y_{j-1} will form an SCS of X_i and Y_j which has length $S[i-1, j-1] + 1$.
- If $x_i \neq y_j$ then either we have to add x_i to the SCS of X_{i-1} and Y_j, or we have to add y_j to the SCS of X_i and Y_{j-1}. Taking the minimum among these two cases gives the answer.

So we get the following recerrence:
$S[i, 0] = i$ and $S[0, j] = j$, for any $0 \leq i \leq n$, $0 \leq j \leq m$. If $i, j > 0$ then:

$$S[i, j] = \min \begin{cases} S[i-1, j] + 1, \\ S[i, j-1] + 1, \\ S[i-1, j-1] + 1, & \text{if } x_i = y_j \end{cases}$$

Total Marks = 45