CSCI3160: Midterm Exam

NOTE 1: Write all your solutions in the answer book.
NOTE 2: For a problem that demands an algorithm of $f(n)$ time, you will still get a full mark if your algorithm runs in $f(n)$ *expected* time.
NOTE 3: You do not need to describe any algorithm that has already been discussed in the lectures or tutorials. For example, if you want to use the k-selection algorithm to find the median in an array of size n, just say "find the median using the k-selection algorithm in $O(n)$ time".

Problem 1 (20%). Consider applying the algorithm discussed in the class to calculate the edit distance between strings $s =$ "honda" and $t =$ "pony". Recall that the algorithm fills in a matrix. Show the values for all the cells in the matrix.

Solution.

	p	o	n	y
h	1	2	3	4
o	2	1	2	3
n	3	2	1	2
d	4	3	2	2
a	5	4	3	3

Problem 2 (20%). Assuming $m \geq n$, give an algorithm to multiply an $m \times n$ matrix A with an $n \times m$ matrix B in $O(m^2 \cdot n^{0.81})$ time. You can assume that m is a multiple of n.

Solution. Cut A and B each into m/n sub-matrices of dimensions $n \times n$. The product AB can be obtained by multiplying each sub-matrix of A with each sub-matrix of B using Strassen's algorithm in $O(n^{2.81})$ time. The total running time is $O((\frac{m}{n})^2 \cdot n^{2.81}) = O(m^2 \cdot n^{0.81})$.

Problem 3 (20%) Let A be an array of n integers. Consider the following recursive function which is defined for any i, j satisfying $1 \leq i \leq j \leq n$:

$$f(i,j) = \begin{cases} 0 & \text{if } i = j \\ A[i] \cdot A[j] + \min_{k=i+1}^{j-1}\{f(i,k) + f(k,j)\} & \text{if } i \neq j \end{cases}$$

Design an algorithm to calculate $f(1, n)$ in $O(n^3)$ time.

Solution. First set $f(i,i) = 0$ for all $i \in [1, n]$. In general, after calculating all $f(i,j)$ with $j - i = s$ (for some integer $s \geq 0$), calculate $f(i,j)$ for all i, j satisfying $j - i = s + 1$. In this way, each $f(i,j)$ can be obtained in $O(n)$ time. Since there are $O(n^2)$ values to compute, the total running time is $O(n^3)$.

Problem 4 (20%). Let S be a set of n integers where n is a power of 2. We want to design an algorithm to output the i-th smallest integer in S for $i = 2^0, 2^1, 2^2, ..., 2^{\log_2 n}$ (namely, $1 + \log_2 n$ integers to output in total). For example, suppose that the input array is $(8, 10, 2, 4, 12, 16, 14, 6)$; we should output 2, 4, 8, and 16. Attempt the following tasks:

(a) (5%) Prove: Suppose that, for some $i \geq 2$, we have already collected the i smallest integers in S into some array A (which is not necessarily sorted). We can obtain in $O(i)$ time the $i/2$ smallest integers in S.

(b) (2%) Prove: $1 + 2 + 4 + 8 + ... + n/2 + n = O(n)$.

(c) (13%) Design an algorithm to find the $1 + \log_2 n$ integers in $O(n)$ time.

Solution. (a) Use k-selection to find the $(i/2)$-th smallest integer x in A. Then collect all the integers in A that are at most x.

(b) Solution obvious and omitted.

(c) Define S_i as the set of i smallest integers in S. After obtaining S_i, we can find the $(i/2)$-th smallest integer in $O(i)$ time. Using (a), $S_{i/2}$ can also be obtained in $O(i)$ time. The algorithm then runs recursively from $i = n$ (and ends at $i = 2$).

Problem 5 (20%). Let $\mathcal{I}$ be a set of n intervals, each of which is in the domain $[0, U]$ for some very large $U \gg n$. It is guaranteed that the union of all the intervals in $\mathcal{I}$ equals $[0, U]$ (i.e., every value in $[0, U]$ is covered by at least one interval in $\mathcal{I}$). We want to pick the smallest number of intervals in $\mathcal{I}$ whose union equals $[0, U]$.

For example, suppose that $\mathcal{I} = \{[10, 15], [0, 35], [20, 50], [55, 60], [5, 30], [0, 25], [40, 60], [45, 50], [25, 45]\}$ and $U = 60$. We need to pick at least 3 intervals, e.g., $\{[0, 35], [20, 50], [40, 60]\}$. Another optimal solution is $\{[0, 25], [25, 45], [40, 60]\}$.

Attempt the following tasks:

(a) (5%) Suppose that I is the longest interval in $\mathcal{I}$ that starts from 0 (e.g., $I = [0, 35]$ in the above example). Prove: I must appear in an optimal solution.

(b) (15%) Describe an algorithm to find an optimal solution. Your algorithm should finish in polynomial time, e.g., $O(n^{100})$.

Solution. (a) Take any optimal solution. Identify the interval I' therein that covers 0. Replace I' with I, which still yields a solution of the same size.

(b) Find the longest interval I covering 0. Suppose that $I = [0, x]$. Discard all the intervals in S that are contained in I. For each remaining interval $[a, b] \in S$, if $x \in [a, b]$, trim the interval into $[x, b]$. Then recursively to pick the smallest number of intervals in S to cover $[x, U]$. Return those intervals together with I.