CPS 130 Final
Spring 1999
9:00am-12:00p.m, Wednesday May 5th
Closed book exam
Do any 7 questions of your choice

Problem 1 (15 points)
Indicate whether the following statements are true or false. Also mention in one sentence the reason behind your answer.
1. By reducing the problem of sorting to the problem of building a heap one can prove that building an $n$-element heap takes time $\Omega(n \log n)$ in the worst case.
2. The $(\log n)^{th}$ smallest number of $n$ unsorted numbers can be determined in $O(n)$ worst-case time.
3. The problem of determining an optimal order for multiplying a chain of matrices can be solved by a greedy algorithm, since it displays the optimal substructure and overlapping subproblem properties.
4. Kruskal's algorithm for finding the minimum spanning tree of a weighted undirected graph is an example of a dynamic programming algorithm.
5. The topological sort of an arbitrary directed graph $G = (V, E)$ can be computed in linear time.
6. In $O(V + E)$ time, a matching in a bipartite graph $G = (V, E)$, can be tested to determine whether it is maximum.

1 Sorting and Selection

Problem 2 (15 points)
Given a sequence of $n$ integers with many duplicates, such that the number of distinct integers in the sequence is $O(\log n)$.

a) Design a sorting algorithm to sort such a sequence using at most $O(n \log \log n)$ comparisons.
   hint: Use an auxiliary data structure that has $O(\log n)$ access time

b) Explain why the lower bound of $\Omega(n \log n)$ comparisons for sorting $n$ numbers does not hold for this case.

Problem 3 (15 points)
Let $A[1...n]$ be a sorted array of distinct integers. Design an algorithm that finds an index $i$ such that $A[i] = i$ if such an $i$ exists. Running time should be $O(\log n)$. 


Problem 4 (15 points)

What is the least number comparisons needed, in the worst case, to find the third largest element from a set of \( n \) elements.

*hint: find the largest and the second largest also*

Problem 5 (15 points)

Assume we have a worst-case linear-time subroutine for finding the median that can be treated like a "black-box". Give a simple, linear-time algorithm that solves the selection problem for an arbitrary order statistic.

2 Trees

Problem 6 (15 points)

Consider two balanced search trees \( T_1 \) and \( T_2 \) representing two sets of integers \( S_1 \) and \( S_2 \) with \( n_1 \) and \( n_2 \) elements, respectively. Consider the problem of determining if \( S_1 \subseteq S_2 \) using \( T_1 \) and \( T_2 \).

- Describe an \( O(n_1 \log n_2) \) time algorithm for determining if \( S_1 \subseteq S_2 \) using \( O(1) \) extra space.
- Describe an \( O(n_1 + n_2) \) time algorithm for determining if \( S_1 \subseteq S_2 \) using \( O(n_1 + n_2) \) extra space.
- Describe an \( O(n_1 + n_2) \) time algorithm for determining if \( S_1 \subseteq S_2 \) using \( O(\log n_1 + \log n_2) \) extra space.

3 Graphs

Problem 7 (15 points)

For the graph in the figure, write down a:

- Depth first traversal
- Breadth first traversal
Problem 8 (20 points)
Consider a special type of weighted directed graphs called flat graphs:
- A flat graph has two special vertices $s$ and $t$.
- The smallest flat graph consists of $s$, $t$, and a single edge:

\[ s \xrightarrow{w} t \]

- Given two flat graphs $G_1$ and $G_2$

\[ \begin{array}{cc}
G_1 & G_2 \\
\text{s}_1 & \text{s}_2 \\
\text{t}_1 & \text{t}_2 \\
\end{array} \]

one can construct a new flat graph by merging the $t$ vertex of $G_1$ with the $s$ vertex of $G_2$ (serial rule)

\[ \begin{array}{cc}
G_1 & G_2 \\
\text{s} & \text{t} \\
\end{array} \]

or by introducing new $s$ and $t$ vertices, connecting the new $s$ vertex to the $s$ vertices of $G_1$ and $G_2$, and connecting the $t$ vertices of $G_1$ and $G_2$ to the new $t$ vertex (parallel rule)

\[ \begin{array}{cc}
G_1 & G_2 \\
\text{s} & \text{t} \\
\end{array} \]

\[ \begin{array}{cc}
\text{w}_1 & \text{w}_3 \\
\text{w}_2 & \text{w}_4 \\
\end{array} \]

a) Let $n$ be the number of vertices in a flat graph $G$. Prove that the number of edge in $G$ is $\Theta(n)$.
b) How long would it take to compute the shortest path from $s$ to $t$ in $G$ using Dijkstra's algorithm.
c) Design and analyze a more efficient algorithm.
    hint: make use of the fact that graph is directed and acyclic (DAG)

Problem 9 (15 points)
A cut-point of a graph is an edge that if deleted, separates the graph. Given a graph $G = (V,E)$ describe a linear time algorithm that tests whether it has a cut-point.
4 Dynamic Programming

Problem 10 (20 points)

A game board consists of a row of $n$ fields, each consisting of two numbers. The first number can be any positive integer, while the second is 1, 2, or 3. An example of a board with $n = 6$ could be the following:

\[
\begin{array}{cccccc}
17 & 2 & 100 & 87 & 33 & 14 \\
1 & 2 & 3 & 1 & 1 & 1
\end{array}
\]

The object of the game is to jump from the first to the last field in the row. The top number of a field is the cost of visiting that field. The bottom number is the maximal number of fields one is allowed to jump to the right from the field. The cost of a game is the sum of the costs of the visited fields.

Let the board be represented in a two-dimensional array $B[n,2]$. The following recursive procedure (when called with argument 1) computes the cost of the cheapest game:

\[
\text{Cheap}(i)
\]

\[
\begin{align*}
\text{IF } i & >n \text{ THEN return } 0 \\
x &= B[i,1]+\text{Cheap}(i+1) \\
y &= B[i,1]+\text{Cheap}(i+2) \\
z &= B[i,1]+\text{Cheap}(i+3) \\
\text{IF } B[i,2] &= 1 \text{ THEN return } x \\
\text{IF } B[i,2] &= 2 \text{ THEN return } \min\{x,y\} \\
\text{IF } B[i,2] &= 3 \text{ THEN return } \min\{x,y,z\}
\end{align*}
\]

END Cheap

The algorithm above has an exponential running time. Describe and analyze a more efficient algorithm for finding the cheapest game.

\textbf{hint: build a table of game costs associated with all possible jumps from each field, starting with the fields in which the game cost is known. The three recursive calls to “Cheap” in the algorithm above then reduce to table lookups.}

Extra Credit (20 points)

You are given $n$ nuts and $m$ bolts with exactly one matching nut for each bolt. Assume that for any given nut and bolt pair, we have an operation available that can detect if they match or the nut is too small or too large. Give an efficient algorithm for finding all $n$ matching pairs of bolts and nuts.