Midterm Examination on Algorithms **April 27, 2000**
Teacher: Biing-Feng Wang

Problem 1: (10%, Growth of Functions)

(1) (5%) Use the definition of Θ to show that $n^3/2000-50n^2-100n+3 = \Theta(n^3)$.

(2) (5%) Let $f(n)$ be an arbitrary positive function. Prove or disprove $f(n)=\Theta(f(n/2))$

Problem 2: (10%, Algorithm Design)

Let x be an integer and S be a set of n integers ranging from 0 to $6n^{\lg\lg n}$. Design an efficient algorithm that determine whether or not there exist two integers in S whose sum is exactly x. Describe your algorithm in detail. What's the time complexity of your algorithm in O-notation.

Problem 3: (15%, Recurrences)

(1) (5%) Let $T(n)=\sqrt{n}\,T(\sqrt{n}) + n$. Find the tight asymptotic bound of $T(n)$ in Θ-notation by appealing to a recursive tree.

(2) (5%) Let $T(n)=28T(n/3) + n^3$. Find the tight asymptotic bound of $T(n)$ in Θ-notation by appealing to the master theorem.

(3) (5%) Let $T(n)=T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+1$. Prove that $T(n)=O(n)$ by substitution method.

Problem 4: (10%, Priority Queue)

Give an implementation of a priority queue S that supports the following three operations efficiently.

Insert(S, x): Insert an element x into S.
Maximum(S): Return the value of the maximum element in S.
Extract-Max(S): Return and remove the maximum element in S.

Describe your implementation in detail. How much time is required for each of the operations?

Problem 5: (10%, Divide-and-Conquer)

Give two examples for each of the following algorithm design strategies.

partition, divide-and-conquer, prune-and-search.

Problem 6: (10%, Fractional Knapsack Problem)

(1) (5%) Give an efficient algorithm to solve the fractional knapsack problem. What's the time complexity of your algorithm?

(2) (5%) Prove that your algorithm in (1) is correct.

Problem 7: (15%, Dynamic Programming)

Let $S=\{s_1, s_2, ..., s_n\}$ be a set of n integers ranging from 1 to n. Given an integer i, we say that S is *able to constitute* i if there is a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = i$. For example, letting $S=\{5, 1, 3, 7\}$, S is able to constitute 12 (=5+7) but is not able to constitute 2.

(1) (5%) Define a function $C(i, j)$ as follows.

$$C(i,j) = \begin{cases} \textit{ture} & \text{if } \{s_1, s_2, ..., s_i\} \text{ is able to constitute } j; \\ \textit{false} & \text{otherwise,} \end{cases}$$

where $0 \le i \le n$ and $0 \le j \le (s_1+s_2+ ...+s_n)$. Describe a recurrence of $C(i, j)$, including all boundary conditions.

(2) (4%) Give an algorithm that, given S and an arbitrary positive integer k, determine whether S is able to constitute k or not.

(3) (2%) What is the time complexity of your algorithm in (2).

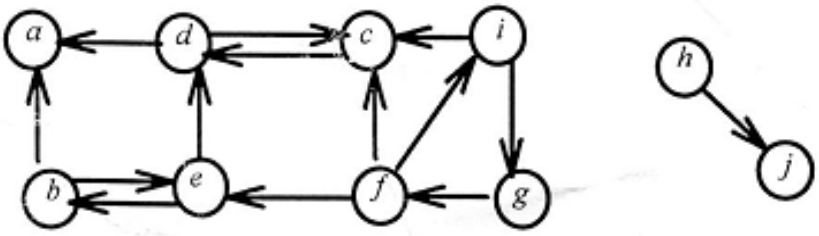
(4) (4%) Modify your algorithm in (2) such that in case that S is able to constitute k, a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = k$ is returned.

Problem 8: (10%, Disjoint Sets)

Prove that using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of m Make-Set, Union, and Find-Set operations, n of which are Make-Set operations, takes $O(m+n\lg n)$ time.

Problem 9: (10%, Elementary Graph Algorithms)

(1) (7%) Use the following graph as an example to illustrate how the problem of strongly connected components can be solved by performing depth-first search twice. (It is not necessary for you to prove the correctness.)



(2) (3%) What is the time complexity of the solution in (1).

Bonus: (5%, Hash function)

What is the condition that a good hash function should satisfy (approximately)? You should describe the condition by the mathematic equation given in the textbook.