國立台灣海洋大學資訊工程學系博士班

101 學年度第一學期博士班資格考命題卷

科目：演算法
日期：2013/01/17

1. (25%) Asymptotic notation: for computational complexity analysis, we define an **asymptotic upper bound** of a given function $g(n)$, denoted by $O(g(n))$, to be a set of functions

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0\}.$$

   (a) (5%) Similarly, we can define an **asymptotic lower bound** of a given function $g(n)$, denoted by $(g(n))$, to be a set of functions $\Omega(g(n))$.
   Please complete the definition of $\Omega(g(n))$ with the format similar to the definition of $O(g(n))$ mentioned above.

   (b) (5%) Also, an **asymptotic exact bound** of a given function $g(n)$, denoted by $\Theta(g(n))$, is a set of function which is an **asymptotic upper and lower bound** of a given function $g(n)$ at same time. Please give a formal definition of **asymptotic exact bound** $\Theta(g(n))$ with the format similar to the definition of $O(g(n))$ mentioned above.

   (c) (10%) Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.
      i. (5%) $T(n) = 2T(n/2) + n^4$.
      ii. (5%) $T(n) = T(7n/10) + n$.

   (d) (5%) Suppose given a problem P to solve, and there are two algorithms, Algo-A and Algo-B, available to solve the problem. Algo-A is with complexity of $O(n)$ and Algo-B is $O(n \log n)$. Will Algo-A always solve the problem faster than Algo-B for all possible problem instances of P? Why or Why not?

2. (25%) Dynamic programming and greedy-algorithm: knapsack problem.
   The knapsack problem is a problem in combinatorial optimization: Given a set of items $x_1, x_2, \ldots, x_n$, each item x_i is with a weight w_i and a value v_i. The objective of the problem is to determine which items should be included in a collection so that the total weight is less than or equal to a given limit W and the total value is as large as possible.

   Two versions of the Knapsack problem can be defined: fractional Knapsack problem and binary Knapsack problem. For binary knapsack problem, we assume each item is not dividable. That is, if an item is selected than it should be taken wholly. In contrast, for fraction Knapsack problem, it is allowed to take any portion of an item. The weight (value) of the fractional item is proportion to the weight (value) of the original one.

(a) Fractional Knapsack problem.

i. (4%) Show that the fractional knapsack problem can be solved by greedy-choice property.

ii. (4%) Provide a pseudo code to solve the fractional knapsack problem.

iii. (2%) What is the complexity of your proposed algorithm.

(b) (15%) The binary knapsack problem is far more complicate to solve in term of complexity, comparing with fractional version. A similar dynamic programming solution for the **0-1 knapsack problem** runs in **pseudo-polynomial time**, if proper setting as follows. Assume $w_1, w_2, \ldots, w_n, W$ are strictly positive integers. Define $m[i, w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i. We can define $m[i, w]$ recursively as follows:

- $m[0, w] = 0$.
- $m[i, w] = 0$.
- $m[i, w] = m[i-1, w]$, if $w_i > w$ (the new item is more than the current weight limit).
- $m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i)$ if $w_i \leq w$.

KnapSack(n, w)

1 $\triangleright$ assume $w[i](v[i])$ is weight (value) of item x_i
2 **if** ($n = 0$ or $W = 0$)
3 **then return** 0
4 **if** $(w[n] > W)$
5 **then return** KnapSack$(n-1, w)$
6 **else return** $\max($KnapSack$(n-1, w),$ KnapSack$(n-1, W-w[n]) + v[n])$

i. (5%) Please find the time complexity of Procedure KnapSack. (hint: for worst cast analysis, we may assume that W is large enough to cover most of the objects)

ii. (7%) From (i), you might find that the time complexity is high. The problem can be solved by rewriting the procedure with a non-recursive version of the algorithm. Please give the non-recursive of the Procedure KnapSack (4%), and find the time complexity of your code. (3%)

iii. (3%) Please explain why we can use dynamic programming strategy to solve this problem.

3. (10%) Please describe briefly the following sorting algorithms along with their time complexities. Which of them are stable sorting algorithms? Which of them are in-place sorting algorithms?

(a) Quicksort

(b) Heapsort

4. (15%) Describe the **Floyd-Warshall algorithm** that, given an $n \times n$ matrix W representing the edge weights of an n-vertex directed graph $G = (V, E)$, solves the all-pairs shortest-paths problem in $\Theta(n^3)$ time.

5. (10%) Show that the subtree size of the root node in an n-element heap is at most $2n/3$.

6. (15%) Describe an $O(n)$-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$, determines the k numbers in S that are closest to the median of S.