國立臺灣師範大學 111 學年度碩士班招生考試試題

科目：**軟體基礎**　　適用系所：**資訊工程學系**

注意：1.本試題共 4 頁，請依序在答案卷上作答，並標明題號，不必抄題。2.答案必須寫在指定作答區內，否則依規定扣分。

1. (5 points) Let T be a rooted binary tree whose vertex set is $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q\}$, and let $(p, n, b, j, q, k, i, c, h, o, l, d, m, f, e, a, g)$ and $(n, p, b, h, j, i, q, k, c, g, o, f, m, l, d, a, e)$ be its post-order traversal and in-order traversal, respectively. The pre-order traversal of T is __(a)__.

2. (15 points) We want to analyze the average-time complexity of insertion sort. Assume that there is a sequence $s = s_1, s_2, \ldots, s_n$ of n distinct integers to sort (into ascending order), and assume that the n integers range from 1 to n. The time complexity of insertion sort depends on the number of *inversions* of s, where an inversion is a pair (s_i, s_j) such that $i < j$ and $s_i > s_j$. Let $t(n)$ be the total number of inversions of all length-n sequences satisfying the above conditions, for $n \geq 1$. Then the average-time complexity is $O(t(n)/n!)$. It can be derived that $t(1) = 0$ and $t(n) = a(n) + b(n) \cdot t(n\text{-}1)$ for $n \geq 2$, where $a(n) =$ __(a)__ and $b(n) =$ __(b)__. Thus, $t(n)/n! =$ __(c)__.
   Note: The requested answers for this question are closed formulas. Fundamental operations that can be used include arithmetic operations (+, −, ×, /), factorial (!), and binomial coefficients.

3. (20 points) Let G be a graph of n vertices and m edges, with vertices numbered from 1 to n. The following procedure Search() traverses all the vertices and edges of G.

```
Search(G)
1.  for each vertex u of G
2.      do visit[u] ← False
3.  Initialize(D)
4.  Insert(D, 1)
5.  while Empty(D) = False
6.      do u ← Extract(D)
7.          if visit[u] = False
8.              then visit[u] ← True
9.                   for every neighbor v of u
10.                      do Insert(D, v)
```

   If one chooses different data structures for D, then the vertices may be visited in different order. Below are two possible data structures, X and Y, that can be used for D. Each of X and Y is implemented using a sufficiently large array. You may assume that both X and Y have size much greater than $m+n$.

Data structure X:	Data structure Y:
Initialize(X) 1. to_insert ← 0 2. to_delete ← 0 Empty(X) 1. **if** to_insert = to_delete 2. **then return** True 3. **return** False Insert(X, a) 1. X[to_insert] ← a 2. to_insert ← to_insert+1 Extract(X) 1. **if** Empty(X) = True 2. **then return** NIL 3. to_delete ← to_delete+1 4. **return** X[to_delete−1]	Initialize(Y) 1. to_insert ← 0 2. to_delete ← 0 Empty(Y) 1. **if** to_insert = to_delete 2. **then return** True 3. **return** False Insert(Y, a) 1. Y[to_insert] ← a 2. to_delete ← to_insert 3. to_insert ← to_insert+1 Extract(Y) 1. **if** Empty(Y) = True 2. **then return** NIL 3. to_insert ← to_delete 4. **if** to_delete > 0 5. **then** to_delete ← to_delete−1 6. **return** Y[to_insert]

Data structure X is a fundamental data structure called __(a)__, and taking X for D results in a search procedure called __(b)__. Data structure Y is another fundamental one called __(c)__, and taking Y for D results in a search procedure called __(d)__.

4. (10 points) A max binary heap is a "nearly complete binary tree" such that the key of a node is greater than or equal to those of its children. For convenience, we assume that each element is the key of itself. If we use an array to implement, where elements are indexed from 1 to n, then for an element with index i its children are the elements with indices $2i$ and $2i$+1. Below is the pseudocode for building a max binary heap. We use $A[i]$ to denote the element with index i.

Build-Max-Heap(A)	Heapify(A, i)
1. **for** $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1 2. **do** Heapify(A, i)	1. $l \leftarrow 2i$ 2. $r \leftarrow 2i+1$ 3. max$\leftarrow i$ 4. **if** $l \leq n$ and $A[l] > A[\text{max}]$ 5. **then** max$\leftarrow l$ 6. **if** $r \leq n$ and $A[r] > A[\text{max}]$ 7. **then** max $\leftarrow r$ 8. **if** max $\neq i$ 9. **then** exchange $A[i]$ and $A[\text{max}]$ 10. Heapify(A, max)

Please analyze the time complexity of Build-Max-Heap(A). You have to explain how your result is derived.

5. (15 points) Please solve the following recurrence relations.

   (a) $T(n) = T(n/3) + 1$ (5 points)

   (b) $T(n) = T(n/3) + T(2n/3) + 3n$ (5 points)

   (c) $T(n) = T(n\text{-}2) + T(n\text{-}1) + 1$ (5 points)

6. (20 points) The knapsack problem is described as follows. Given n distinct items of weights $w_1, w_2, \ldots, w_n$, values $v_1, v_2, \ldots, v_n$ and a knapsack of capacity W, find the items to include in the knapsack so that the total weight is less than or equal to W and the total value is as large as possible.

   (a) (7 points) Please describe a brute-force approach for solving the problem. Analyze the time complexity.

   (b) (5 points) To design a dynamic programming approach for solving the problem, we create a table $F[i, j]$, which stores the value of an optimal solution to a sub-instance, i.e., the value of the most valuable subset of the first i items that fit into the knapsack of capacity j. Please write down the recurrence.

   (c) (5 points) Continuing (b), please fill the table F for the following instance, supposing that each item exactly has one quantity.

capacity $W = 6$

item	weight	value
A	2	4
B	3	2
C	1	2
D	4	3

(d) (3 points) Consider the instance described in (c): which of the following greedy choices would give the optimal solution? (allow multiple choices)

i. Take the item that has the highest value
ii. Take the item that has the lowest weight
iii. Take the item that has the highest value/weight

7. (15 points) The Floyd-Warshall algorithm solves the all-pairs shortest-paths problem on a directed graph. The pseudo code is given as follows:

```
Floyd-Warshall(W)
1   n ← rows[W]
2   D^(0) ← W
3   for k ← 1 to n
4       do for i ← 1 to n
5           do for j ← 1 to n
6               d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
7   return D^(n)
```

(a) (5 points) Explain the meaning of $d_{ij}^{(k)}$.

(b) (5 points) Given a graph with the weight matrix W:

$$\begin{bmatrix} 0 & 3 & \infty & 2 & 6 \\ 5 & 0 & 4 & 2 & \infty \\ \infty & \infty & 0 & 5 & \infty \\ \infty & \infty & 1 & 0 & 4 \\ 5 & \infty & \infty & \infty & 0 \end{bmatrix}$$

please write down $D^{(1)}$.

(c) (5 points) Does the Floyd-Warshall algorithm give the correct result if the input graph has negative weights (but no negative cycle)? Briefly explain your answer.