國立嘉義大學 109 學年度

資訊工程學系碩士班招生考試試題

科目：資料結構

1. Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on. We want to compute a certain Fibonacci number Fib(n).
   (a)What are the **time** complexity and **space** complexity using an iterative algorithm? (5%)
   (b)What is the **space** complexity using a recursive algorithm? (5%)
   (c)What is the **time** complexity using a recursive algorithm? (10%)
   No explanation is required for (a) and (b). Show your work for (c).

2. Answer True or False for the following complexity questions.
   (a) $O(n^2) > O(n \log n)$ (2%)
   (b) $O(n!) > O(2^n)$ (2%)
   (c) $O(2^n) > O(n^2)$ (2%)
   (d) $O(n \log n) > O(2^n)$ (2%)
   (e) $O(n \log n) > O(n)$ (2%)

3. Graph traversal can be either depth first traversal or breadth first traversal. Please write pseudo code, without using recursion, for (a) depth first traversal (10%), and (b) breadth first traversal (10%).

4. The doubly linked list (*DLlist*) has the advantage of traversing in both forward and backward directions. Thus we can define each node in *DLlist* as follows. It has two link fields, one for forward direction and the other for backward direction.

```
typedef struct node *nodePointer;
typedef struct node {
        nodePointer llink;
        element data;
        nodePointer rlink;
        };
```

   (a) Assume that we want to add a new node between two nodes, *anode* and *bnode*, in *DLlist*. Draw a figure to show this insertion. Note that you should label the affected fields so that you show how each step in the insertion function is executed (eg., *newnode->llink*, *newnode->rlink*, etc.). (10%)
   (b) Write down the algorithms for the insertion and deletion functions in *DLlist*. Assume that you want to insert a new node to the right of *anode* and delete a node named *dnode* for a nonempty list. (15%)

5. Write the detailed steps of carrying out merge sort in the list (13, 4, 19, 38, 9, 27, 6, 11, 25, 7, 20). (10%)

6. A simplest known sorting method called "**counting sort**" is described as follows. Declare an array *count* and set *count*[*k*] to the number of key values that are less than *k*. Therefore, the record with key *k* can be placed in position *count*[*k*] of an output list. (You should beware of the possibility of equal key values. See an example in Figure 1.) Assume that all keys of the records are integers and their values are in range of [0, *d*], where *d* is a constant. Write an algorithm to sort a set of records with size *n* by the key value using this method (including how to determine the *count*[*k*]). (15%)

index	[0]	[1]	[2]	[3]	[4]	[5]	[8]	[7]	[8]	[9]	[10]	[11]
key of record	2	1	3	5	2	7	8	4	9	1	2	7

key (index)	0 [0]	1 [1]	2 [2]	3 [3]	4 [4]	5 [5]	6 [6]	7 [7]	8 [8]	9 [9]
count	0	0	2	5	6	7	8	8	10	11

Figure 1: An example of counting sort array with size 12 and key in range of [0, 9]