Computer Science Foundation Exam

August 10, 2012

Section I B

COMPUTER SCIENCE

**NO books, notes, or calculators may be used,
and you must work entirely on your own.**

SOLUTION

Question #	Max Pts	Category	Passing	Score
1	10	ANL	7	
2	10	DSN	7	
3	10	DSN	7	
4	10	ALG	7	
5	10	ALG	7	
TOTAL	50			

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (10 pts) ALS (Algorithm Analysis)

(a) (4 pts) Determine, **with proof**, the run-time of the following function in terms of the formal parameters a and b:

```
int f(int a, int b) {

     int i,j, sum = 0;

     for (i=0; i<a; i++) {
          j = b;
          while (j > 0) {
               j = j/2;
               sum++;
          }
     }
     return sum;
}
```

The outer loop runs a times. (1 pt) The inner loop always runs the same number of times because j always starts off equal to b. Since we are repeatedly dividing by 2, we run the loop k times where $2^k \sim b$. Thus, roughly $k = \log_2 b$. (2 pts) It follows that the total run-time is O(algb). (1 pt)

<u>O(alg b)</u>

(b) (6 pts) Algorithm A runs in $O(n^2)$ time, where n is the input size. On an input of size 10000 Algorithm A takes 42 ms to complete. How long would it be expected for Algorithm A to complete on an input of size 30000? **Please show all of your work.**

Let T(n) equal the run time of algorithm A. Then, $T(n) = cn^2$. (2 pts) Using the given information, we have:

$T(10000) = c(10000)^2 = 42$ ms, so $c = (42/10000^2)$ ms. (2 pts)

$T(30000) = c(30000)^2 = \frac{42(30000)^2}{(10000)^2} ms = 42(\frac{30000}{10000})^2 ms = 42(3)^2 ms = 378\ ms$ (2 pts)

<u>378 ms</u>

2) (10 pts) DSN (Recursive Algorithms)

The Catalan Numbers are a sequence of numbers seen in many combinatorial problems. The recursive definition of the Catalan Numbers, where C_i represents the i^{th} Catalan Number, is as follows:

$$C_0 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$$

Write a **recursive** function that calculates the appropriate Catalan Number using the function header provided below. Note, `catalan(0)` should return 1, `catalan(1)` should return 1, `catalan(2)` should return 2 and `catalan(3)` should return 5.

```
int catalan(int n) {

    if (n == 0)                    // Base case 2 pts, may include 1.
        return 1;

    int sum = 0, i;                // 1 pt

    for (i=0; i<n; i++)            // 2 pts
        sum = sum + catalan(i)*catalan(n-i-1); // 4 pts
    return sum;                    // 1 pt

}
```

3) (10 pts) DSN (Linked Lists)

Write a function that operates on an existing linked list of 0 or more integers. The function will have two parameters passed in: the head of the list (**`front`**) and an integer value (**`num`**). Your function should create a new node storing `num`, insert this node to the back of the linked list pointed to by `front`, and return a pointer to the head of the resulting list.

```
struct node {
    int data;
    struct node *next;
};

struct node* insertToBack(struct node *front, int num) {

    // 3 points for fully creating the node.
    struct node* temp = (struct node*)(malloc(sizeof(struct node)));

    temp->data = num;
    temp->next = NULL;

    // 2 points for this special return case.
    if (front == NULL)
        return temp;

    //3 pts to access the last node.
    struct node* iter = front;

    while (iter->next != NULL)
        iter = iter->next;

    iter->next = temp;  // 1 pt - to link
    return front;       // 1 pt - to return




}
```

4) (10 pts) ALG (Tracing)

What is printed out by running the following program? Fill in the result in the boxes below:

```
#include <stdio.h>

#define SIZE 10

int main() {

    int f[] = {2, 8, 1, 3, 5, 0, 9, 7, 4, 6};

    int i, g[SIZE], h[SIZE];

    for (i=0; i<SIZE; i++)
        g[i] = f[f[i]];

    for (i=0; i<SIZE; i++)
        h[i] = f[g[i]];

    for (i=0; i<SIZE; i++)
        printf("%d ", h[i]);
    printf("\n");

    return 0;
}
```

8	**5**	**4**	**3**	**2**	**1**	**9**	**7**	**0**	**6**

Grading: 1 point per blank, no exceptions.

5) (10 pts) ALG (Sorting)

Consider the following buggy implementation of insertion sort:

```
void sort(int array[], int length) {

    int i,j;

    for (i=1; i<length; i++) {

        int j = i;
        while (array[j-1] > array[j]) {
            int temp = array[j-1];
            array[j-1] = array[j];
            array[j] = temp;
            j--;
        }

    }
}
```

(a) (4 pts) Give an input array of size 5 that might not be properly sorted by this function. (Your input should be such that it may either cause a run-time error or an incorrect answer, depending on the system upon which, the function is executed.)

2	**1**	**4**	**7**	**5**

Grading: Give full credit to any array with a non-minimal value in the first slot. Give 0 points otherwise.

(b) (6 pts) Suggest a fix for the issue so the sort works properly for all possible input arrays. Explain why your change fixes the error previously caused.

Change the while loop as follows:

```
while (j > 0 && array[j-1] > array[j])  // 4 pts
```

This will prevent an array out of bounds error if j happens to get set to 0 with the statement j--. Thus, if j does become j with that statement, the check to see if j is greater than 0 will terminate the while loop via short-circuiting, so array[j-1] never gets evaluated. (2 pts)