Computer Science Foundation Exam

May 8, 2015

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	ANL	7	
3	10	DSN	7	
4	10	DSN	7	
5	10	ALG	7	
TOTAL	50		35	

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 <u>not</u> 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 <u>be neat</u>.

1) (10pts) ANL (Algorithm Analysis)

Consider the following function shown below:

```
double harmonic(int n) {
    if (n == 1)
        return 1.0;
    return 1.0/n + harmonic(n-1);
}
```

(a) (3 pts) Let T(n) be the runtime of `harmonic` with an input of size n. Write a recurrence relation that T(n) satisfies, assuming that a constant number of simple operations takes 1 unit of time.

T(n) = T(n-1) + O(1), because our recursive call has input n-1 and the extra work we do is constant. (Grading: 1 pt for each component.)

T(1) = O(1), because it is computed in constant time.

(b) (7 pts) Use the iteration technique to solve for T(n). Note: You may assume T(1) = 1.

T(n) = T(n-1) + 1
T(n) = T(n-2) + 1 + 1 = T(n-2) + 2 **(Grading - 1 pt)**
T(n) = T(n-3) + 1 + 2 = T(n-3) + 3 **(Grading -1 pt)**

In general, we have:

T(n) = T(n-k) + k **(Grading - 2 pts)**

Plugging in k = n-1, we have:

T(n) = T(n-(n-1)) + (n-1) **(Grading - 1 pt)**
= T(1) + (n-1) **(Grading - 1 pt)**
= 1 + n - 1 = n. **(Grading - 1 pt)**

2) (10 pts) ANL (Algorithm Analysis)

(a) (5 pts) An algorithm runs in $O(\lg_2 n)$ time. It takes 36 ns for an input size of 8. If another run of the algorithm takes 48 ms, how large was the input? Simplify your answer to a single integer.

Let T(n) be the run time of the algorithm. We have:

T(n) = C * lg n **(Grading - 1 pt)**
T(8) = C * lg 8 = 36 ns **(Grading - 1 pt)**
C = 36/3 = 12 ns **(Grading - 1 pt)**

Let x be experimental input size

T(x) = C * lg x = 48
lg x = 48 ns/12 ns =4 **(Grading - 1 pt)**
$x = 2^4 = 16$ **(Grading - 1 pt)**

(b) (5 pts) A search algorithm performs a single search on a database of *n* elements in $O(\sqrt{n})$ time. If 1,000,000 of these searches can be performed on a database of size 10000 in 8 seconds, how long would 500,000 searches take on a database of size 640000?

Let $T(n) = c\sqrt{n}$ be the run time of one search. Thus, a million searches would take 1000000T(n). The initial given input size is n = 10000. Thus, we have:

$$1000000c\sqrt{10000} = 8sec$$
$$1000000c(100) = 8sec$$
$$10^8 c = 8sec$$
$$c = \frac{8}{10^8} sec = 80\ ns$$

The desired run time for the new set of searches is:

$$500000c\sqrt{640000} = 5(10^5) \times \frac{8sec}{10^8} \times 800 = 4(10^8) \times \frac{8sec}{10^8} = 32sec$$

Grading: 3 pts for solving for c, 2 pts for plugging in this value to solve for the new time.

3) (10 pts) DSN (Linked Lists)

Given a pointer to a singly linked list in which each node contains an integer, write a function called **`getNumOdd`** that

a) counts the number of odd values in the list
b) inserts a new node at the front of the list to indicate the number of odd values found
c) returns a pointer to the updated list.

Note: Your input list may be empty, but pointer returned must point to a list with at least one element.

Use the struct definition provided below.

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

node* getNumOdd(node* front) {

    node  *temp = front;                    // 1 pt
    int count = 0;                          // 1 pt

    while(temp!= NULL) {                    // 1 pt

        if(temp->value %2 != 0)             // 1 pt
            count++;                        // 1 pt
        temp = temp->next;                  // 1 pt
    }

    temp = malloc(sizeof(struct node));     // 1 pt
    temp -> value == count;                 // 1 pt
    temp->next = front;                     // 1 pt
    return temp;                            // 1 pt









}
```

4) (10 pts) DSN (Binary Trees)

Write a function that is given a pointer to the root of a valid binary search tree with unique elements and prints out a list of all the odd numbers stored in the binary search tree, in descending order. Use the struct definition and function prototype provided.

```
typedef struct treenode {
    int data;
    struct treenode *left;
    struct treenode *right;
} treenode;

void printOddDescending(treenode* root) {

    if (root == NULL) return;                // 1 pt

    printOddDescending(root->right);         // 2 pts

    if (root->data%2 == 1)                   // 2 pts
        printf("%d ", root->data);           // 1 pt

    printOddDescending(root->left);          // 2 pts

                                             // 2 pts for correct
                                             // order of calls.
}
```

5) (10 pts) ALG (Sorting)

1. (a) (6 pts) Given the array below, show the state of the array after each iteration of the loop in the bubble sort algorithm

Index	0	1	2	3	4	5	6
Value	25	83	14	72	70	65	11

Index	0	1	2	3	4	5	6
1st iteration	**25**	**14**	**72**	**70**	**65**	**11**	**83**
2nd iteration	**14**	**25**	**70**	**65**	**11**	**72**	**83**
3rd iteration	**14**	**25**	**65**	**11**	**70**	**72**	**83**
4th iteration	**14**	**25**	**11**	**65**	**70**	**72**	**83**
5th iteration	**14**	**11**	**25**	**65**	**70**	**72**	**83**
6th iteration	**11**	**14**	**25**	**65**	**70**	**72**	**83**

Grading: 1 pt per row, row must be perfectly correct to get the point.

(b) (4 pts) What are the average case and worst case run times of each of the following sorting algorithms, for sorting n items? Please give a Big-Oh bound for each.

Sort	Average Case Run Time	Worst Case Run Time
Insertion Sort	**$O(n^2)$**	**$O(n^2)$**
Selection Sort	**$O(n^2)$**	**$O(n^2)$**
Merge Sort	**O(nlgn)**	**O(nlgn)**
Quick Sort	**O(nlgn)**	**$O(n^2)$**

Grading: 1/2 pt per item, round down.