Computer Science Foundation Exam

December 18, 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 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) (10pts) ANL (Algorithm Analysis)

Consider the recursive function `sum` shown below:

```
double sum(int* array, int low, int high){
    if (low == high)
        return array[low];
    int mid = (low+high)/2, left = 0, right = 0, i;
    for (i=low; i<=mid; i++) left += array[i];
    for (i=mid+1; i<=high; i++) right += array[i];
    if (left > right) return left + sum(array, low, mid);
    return right + sum(array, mid+1, high);
}
```

(a) (3 pts) Let T(n) represent the run time of the function call `sum(array, 0, n-1)`, where array is an integer array of size n. Write a recurrence relation that T(n) satisfies.

$$T(n) = T\left(\frac{n}{2}\right) + O(n), T(1) = 1$$

Grading: 2 pts T(n/2), 1 pt O(n).

(b) (7 pts) Using the iteration method, determine a closed-form solution (Big-Oh bound) for T(n). Assume T(1) = O(1).

$$T(n) = T\left(\frac{n}{2}\right) + cn$$
$$T(n) = T\left(\frac{n}{4}\right) + \frac{cn}{2} + cn$$
$$T(n) = T\left(\frac{n}{4}\right) + \frac{3cn}{2}$$
$$T(n) = T\left(\frac{n}{8}\right) + \frac{cn}{4} + \frac{3cn}{2}$$
$$T(n) = T\left(\frac{n}{8}\right) + \frac{7cn}{4}$$

After k steps, we have

$$T(n) = T\left(\frac{n}{2^k}\right) + \frac{(2^k - 1)cn}{2^{k-1}}$$

Plugging in $\frac{n}{2^k} = 1$, we have

$$T(n) = T(1) + \frac{(n-1)cn}{\frac{n}{2}}$$

$$T(n) = 1 + 2c(n-1)$$

$$T(n) = O(n)$$

Grading: 1 pt each iteration (3 pts total), 2 pts general form with k, 2 pts finishing problem. Note in place of *cn* students can write *O(n)* or even *n*, with no penalty.

2) (10 pts) ANL (Algorithm Analysis)

(a) (5 pts) A matrix factorization algorithm that is run on a input matrix of size *n* x *n*, runs in O(n^3) time. If the algorithm takes 54 seconds to run for an input of size 3000 x 3000, how long will it take to run on an input of size 1000 x 1000?

Let T(n) be the run-time of the algorithm on a matrix input of size n x n. We have:

$$T(3000) = c3000^3 = 54\ sec$$
$$c = \frac{54}{27 \times 10^9} sec = \frac{2}{10^9} sec$$

We desire to find T(1000):

$$T(1000) = c(1000^3) = \frac{2\ sec}{10^9} \times 10^9 = 2\ sec$$

Grading: 2 pts solving for c, 2 pts plugging in c, 1 pt for simplifying to 2 sec. Ratio method is valid as well, map points accordingly.

(b) (5 pts) A string algorithm with inputs of lengths *n* and *m* runs in O(n^2m) time. If the algorithm takes 2 seconds to run on an input with *n* = 1000 and *m* = 500, how long will the algorithm take to execute on an input with *n* = 250 and *m* = 1000?

Let T(n, m) be the run-time of the algorithm on strings inputs with lengths n and m. We have:

$$T(1000, 500) = c(1000^2)(500) = 2\ sec$$
$$c = \frac{2\ sec}{5 \times 10^8}$$

We desire to find T(250, 1000):

$$T(250, 1000) = c(250^2)(1000) = \frac{2\ sec}{5 \times 10^8} \times 625 \times 10^5 = \frac{250}{1000} sec = .25\ sec$$

Grading: 2 pts solving for c, 3 pts obtaining final answer. Ratio method is valid as well, map points accordingly.

3) (10 pts) DSN (Linked Lists)

Write a **recursive** function, `aboveThreshold`, that takes in a pointer to the front of a linked list storing integers, and an integer, `limit`, and returns the number of values stored in the linked list that are strictly greater than `limit`. For example, if the function was called on a list storing 3, 8, 8, 6, 7, 5, 7, 9 and `limit` equaled 6, then the function should return 5, since the 2nd, 3rd, 5th, 7th and 8th values in the list are strictly greater than 6. (Notice that we don't count the 4th element.)

Use the struct definition provided below.

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

int aboveThreshold(node* front, int limit) {

    if (front == NULL) return 0;

    int cnt = 0;
    if (front->data > limit) cnt = 1;

    return cnt + aboveThreshold(front->next, limit);
}
```

Grading conceptually: 3 pts for the base case, 3 pts for adding 1 in the case that the first item is greater than the limit, 3 pts for adding in the appropriate recursive call, 1 pt for returning. Max of 3 pts for an iterative solution.

4) (10 pts) DSN (Binary Trees)

We define the ***offcenter value*** for each node in a binary tree as being the absolute value of the difference between the height of its left subtree and the height of its right subtree. For example, for an AVL tree, each node has an offcenter value of 0 or 1. Also, note that we define the offcenter value of a null node to be 0. Write a function, `maxOffCenterValue` that computes the maximum offcenter value of any node in a tree pointed to by root. To make your task easier, assume that the height of each node is stored in the corresponding struct for that node in the component `height`.

Using the struct definition given below, complete the function in the space provided.

```
#include <math.h>

typedef struct treenode {
    int value;
    int height;
    struct treenode *left;
    struct treenode *right;
} treenode;

int max(int a, int b) {
    if (a > b) return a;
    return b;
}

int maxOffCenterValue(treenode* root) {

    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return 0;
    if (root->left == NULL || root->right == NULL)
        return root->height;

    int lVal = maxOffCenterValue(root->left);
    int rVal = maxOffCenterValue(root->right);
    int res = max(lVal, rVal);

    int cur = abs(root->left->height - root->right->height);
    return max(res, cur);
}
```

Grading: 1 pt NULL case, 1 pt 1 node case, 2 pts root has one child, 1 pt for left rec call, 1 pt for right rec call, 2 pts for current node calculation, 2 pts for getting max of all three.

5) (10 pts) ALG (Sorting)

Write the code for any one of the following O(n^2) sorts: Bubble Sort, Insertion Sort, Selection Sort in a single function below. Your code should sort the array from smallest to largest. (Namely, after your code finishes array[i] ≤ array[i+1] for all i, 0 ≤ i < length-1.) Please provide the name of the sort you are choosing to implement and fill in the function prototype below.

```
void bubblesort(int* array, int length) {
    int i,j;
    for (i=length-1; i>0; i--) {
        for (j=0; j<i; j++) {
            if (array[j] > array[j+1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
}

void insertionsort(int* array, int length) {
    int i,j;
    for (i=1; i<length; i++) {
        j = i;
        while (j>0 && array[j] < array[j-1]) {
            int temp = array[j];
            array[j] = array[j-1];
            array[j-1] = temp;
            j--;
        }
    }
}

void selectionsort(int* array, int length) {
    int i,j;
    for (i=length-1; i>=0; i--) {
        int bestJ = 0;
        for (j=1; j<=i; j++) {
            if (array[j] > array[bestJ])
                bestJ = j;
        }
        int temp = array[i];
        array[i] = array[bestJ];
        array[bestJ] = temp;
    }
}
```

Grading: 3 pts outer loop structure, 4 pts inner loop structure, 3 pts appropriate swapping, 1 pt off for sorting properly but using the wrong name for the sort or writing an extra function.