Computer Science Foundation Exam

August 8, 2014

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)

List tight Big-Oh bounds(worst case), in terms of the variables used in the description, for each of the following algortihms/operations. If not specified, assume an efficient implementation.

(a) Inserting an element into an AVL tree with *n* elements. O(lg n)

(b) Dequeing an element from a queue of *n* elements, where the queue is implemented with an array. O(1)

(c) Adding an *m* bit number to a *n* bit number. O(m+n)

(d) Sorting *n* numbers using a Quick Sort, where the partition element is always the left-most item in the designated subarray. $O(n^2)$

(e) Inserting an item into a heap containing n^2 items. O(lg n)

(f) Searching for an element in a binary tree of *n* elements. O(n)

(g) Printing out each of the permutations of the numbers 1, 2, ..., n. Note: Assume printing one value takes O(1) time. O(n(n!))

(h) Running a floodfill on an n x n array. (Note: an example of a floodfill is the recursive clear in Minesweeper.) $O(n^2)$

(i) Searching for a number in a sorted array of *n* numbers. O(lg n)

(j) Deleting the first element of a linked list of *n* elements. O(1)

Grading: 1 pt each per answer

2) (10 pts) ANL (Algorithm Analysis)

(a) (5 pts) An algorithm for sorting student records runs in $\theta(n^2)$ time. It takes 20 ms to sort 10,000 student records. How much time will it take, approximately, in ms, to sort 40,000 student records?

Let $T(n) = cn^2$, be the run-time of the algorithm for input size n, for some constant c.

We know that $T(10000) = c(10000)^2 = 20$ ms, so $c = \frac{20}{10^8} = 2 \times 10^{-7}\ ms$. (Grading: 2 pts, no need to simplify.)

$T(40000) = c(40000)^2 = 2 \times 10^{-7} ms \times 16 \times 10^8 = 320 ms$ (Grading: 3 pts)

(b) (5 pts) An algorithm for finding a valid schedule of *n* events runs in $\theta(\ (n+1)!\ )$ time. For $n = 10$, the algorithm takes 50 ms. How long will it take, in seconds, approximately, to run on an input with $n = 12$?

Let $T(n) = c(n+1)!$, be the run-time of the algorithm for input size n, for some constant c.

We know that $T(10) = c(11!) = 50$ ms. (Grading 1 pt)

$T(12) = c(13!) = c(13)(12)(11!) = (c(11!))(156) = (50 \text{ ms})(156) = 7800 \text{ ms} = 7.8$ seconds. (Grading: 3 pts to get 7800 ms, 1 pt to convert to seconds.)

3) (10 pts) DSN (Linked Lists)

Write a function, `getValue`, that takes in a pointer to the front of a linked list storing an integer, with one digit stored in each node and returns the value of the number represented by the linked list. You may assume that the length of the initial linked list passed to the function is in between 0 and 9 items (so no need to worry about overflow), and that the digit field of each struct is in between 0 and 9, inclusive. (Note: An empty linked list has value 0.)

Use the struct definition provided below.

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

int getValue(node* number) {

    int total = 0;                                  // 1 pt
    while (number != NULL) {                        // 2 pts
        total = 10*total + number->digit;           // 4 pts
        number = number->next;                      // 2 pts
    }
    return total;                                   // 1 pt

// 4 pt line is as follows: 1 pt for 10* 1 pt for + 2 pts for
//                          number->digit.

}
```

4) (10 pts) DSN (Binary Trees)

The depth of a node in a binary tree is the distance of that node from the root. Write a **recursive** function that takes in a pointer to the root of a binary tree and returns the ***sum of the depths*** of the nodes of the tree. (For example, a complete binary tree of 7 nodes has 1 node with depth 0, 2 nodes with depth 1 and 4 nodes with depth 2, for a sum of depths of nodes of 0 + 2(1) + 4(2) = 10. Use the struct definition and function prototype given below. You may also assume that curDepth in sumDepthRec represents the depth of root within the whole binary tree.

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

double sumDepth(treenode* root) {
    return sumDepthRec(root, 0);
}

double sumDepthRec(treenode* root, int curDepth) {

    // Grading: 3 pts
    if (root == NULL) return 0;

    // Grading: 1 pt return, 2 pts curDepth, 2 pts recLeft,
    //          2 pts recRight
    return curDepth + sumDepthRec(root->left, curDepth+1) +
                      sumDepthRec(root->right, curDepth+1);
}
```

5) (10 pts) ALG (Sorting)

(a) (5 pts) Consider sorting the array below using Insertion Sort. Show the contents of the array after each iteration of the outer loop.

Original	12	8	9	1	6	4
1st iteration	**8**	**12**	**9**	**1**	**6**	**4**
2nd iteration	**8**	**9**	**12**	**1**	**6**	**4**
3rd iteration	**1**	**8**	**9**	**12**	**6**	**4**
4th iteration	**1**	**6**	**8**	**9**	**12**	**4**
5th iteration	**1**	**4**	**6**	**8**	**9**	**12**

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

(b) (5 pts) Show the result of running an in-place partition using index 0 as the partition element on the array shown below. (Note: Recall that in this algorithm, we use low and high index variables, sweeping through both sides until the low and high indexes cross over.)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Values	45	48	83	7	1	77	37	61	39	75	54	23	64	42	65	93

Depending on which in-place partition is used, there are various possible answers. Three are provided below. Follow the grading criteria presented.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sol 1	**37**	**42**	**23**	**7**	**1**	**39**	**45**	**61**	**77**	**75**	**54**	**83**	**64**	**48**	**65**	**93**
Sol 2	**42**	**23**	**39**	**7**	**1**	**37**	**45**	**61**	**83**	**75**	**54**	**48**	**64**	**93**	**65**	**77**
Sol 3	**7**	**1**	**37**	**39**	**23**	**42**	**45**	**61**	**93**	**75**	**54**	**48**	**64**	**77**	**65**	**83**

Grading: 1 pt for having 45 in the correct location, 2 pts for everything in indexes 0 – 5 being less than 45 and 2 pts for everything in indexes 7 – 15 being greater than 45.