Computer Science Foundation Exam

August 14, 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 recursive function `diminish` shown below:

```
double diminish(int m, int n){
    if (n == 0)
        return m;
    return  1.0/2*diminish(m,n-1)
}
```

(a) (3 pts) Let T(n) represent the run time of the function `diminish`. Write a recurrence relation that T(n) satisfies.

$$T(n) = T(n-1) + O(1)$$

Grading: 1 pt for each component. Note that the last component may be any positive integer constant and still receive full credit.

(b) (6 pts) Using the iteration method, determine a closed-form solution (Big-Oh bound) for T(n).

$$T(n) = T(n-1) + O(1)$$
$$T(n) = T(n-2) + O(1) + O(1)$$
$$T(n) = T(n-3) + O(1) + O(1) + O(1)$$
$$T(n) = T(n-k) + kO(1)$$

Plugging in k = (n-1), we find:

$$T(n) = T(n-(n-1)) + (n-1)O(1)$$
$$T(n) = T(1) + (n-1)O(1)$$
$$T(n) = 1 + (n-1)O(1)$$
$$T(n) = O(n)$$

Grading: 3 pts for couple iterations and generalization, 3 pts for rest - allow with or without Big-Oh notation. Give full credit to any function that is O(n), most likely ones are n, 2n, 3n, with a plus or minus 1, potentially.

(c) (1 pt) In terms of the values of `m` and `n`, respectively, what does the function call `diminish(m,n)` return? (You may assume that `m` and `n` are both positive.)

$$diminish(m,n) = \frac{m}{2^n}$$

Grading: 1 pt for correct answer, 0 otherwise

2) (10 pts) ANL (Algorithm Analysis)

(a) (5 pts) An algorithm for searching for a housing contract in a database of *n* records takes *O(lg n)* time. When $n = 2^{20}$, one million searches can be performed in one fifth of a second. If we increase the database to size $n = 2^{25}$, how long will 500,000 searches take?

Let T(n) represent the time one search takes. Thus, $T(n) = clgn$, for some constant *c*. Using the given information, we have:

$$10^6 T(n) = .2sec = 10^6 clg(2^{20})$$
$$.2sec = 10^6(20)c$$
$$c = 10^{-8}sec$$

We are being asked to find $500000T(2^{25})$:

$$500000T(2^{25}) = 5(10^5)(10^{-8}sec)\lg(2^{25})$$
$$= 5(10^5)(10^{-8}sec)25$$
$$= 125(10^{-3}sec)$$
$$= 125ms, or\ .125\ sec$$

Grading: 1 pt setting up valid equation, 2 pts solving for constant, 2 pts for plugging into second part and getting the answer - can be represented in any unit of time, though sec and ms are probably going to be the most common.

(b) (5 pts) A shortest distance algorithm on an $n \times m$ street grid runs in *O(nm)* time. If the algorithm takes 2 seconds to run on a 4000 × 3000 sized grid, how long will it take on a grid of size 2000 × 18000 sized grid?

Let T(n, m) represent the time algorithm takes . Thus, $T(n, m) = cnm$, for some constant *c*. Using the given information, we have:

$$T(4000,3000) = 2sec = c(4000)(3000)$$
$$c = \frac{1}{6}10^{-6}sec$$

Now we solve for T(2000, 18000):

$$T(2000,18000) = \left(\frac{1}{6}10^{-6}\text{sec}\right)(2000)(18000) = \frac{36}{6}sec = 6\ sec$$

Grading: 1 pt setting up valid equation, 2 pts solving for constant, 2 pts for plugging into second part and getting the answer - can be represented in any unit of time, though sec is probably going to be the most common.

3) (10 pts) DSN (Linked Lists)

Write a function, `moveFrontToBack`, that takes in a pointer to the front of a ***doubly*** linked list storing an integer, moves the first node of the list to the back of the list and returns a pointer to the new front of the list. If the list contains fewer than two elements, the function should just return the list as it is. (Note: prev points to the previous node in the list and next points to the next node in the list.)

Use the struct definition provided below.

```
typedef struct dllnode {
    int value;
    struct dllnode* prev;
    struct dllnode* next;
} dllnode;

dllnode* moveFrontToBack(dllnode* front) {

    if (front == NULL || front->next == NULL)      // 2 pts
        return front;

    dllnode* newfront = front->next;

    dllnode* back = newfront;                      // 3 pts iterating to back
    while (back->next != NULL)
        back = back->next;

    back->next = front;                            // 1 pt
    front->prev = back;                            // 1 pt
    front->next = NULL;                            // 1 pt
    newfront->prev = NULL;                         // 1 pt

    return newfront;                               // 1 pt
}
```

Grading conceptually: 2 pts base cases, 3 pts iterating to back, 5 pts reattaching things and returning.

4) (10 pts) DSN (Binary Trees)

Mark and his buddy Travis have devised a password scheme to secure files that they send among themselves. Their scheme hides the password in a string of English letters. The password is the alphabetically ordered sequence of the consonants in the string. So as not to have to compute the password each time, Mark has written a function called `printPassword`, which takes the letters of the original string stored in a binary search tree and prints out the password. For example, if the string in the message is **mental**, the password printed out would be **lmnt**. Or if the string was ***fragile***, then the password would be ***fglr.*** You may call the following function in your solution:

```
// Returns 1 if c is a consonant, 0 otherwise.
int isConsonant(char c)
```

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

```
typedef struct treenode {
    char ch;
    struct treenode *left;
    struct treenode *right;
} treenode;

void printPassword(treenode* root) {

    if (root != NULL) {                    // 2 pts
        printPassword(root->left);         // 2 pts
        if (isConsonant(root->ch))         // 2 pts
            printf("%c", root->ch);        // 2 pts
        printPassword(root->right);        // 2 pts
    }
}
```

5) (10 pts) ALG (Sorting)

(a) (4 pts) Consider sorting the array below in ascending order using Bubble Sort. Show the contents of the array after each iteration of the outer loop.

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

Grading: 1 pt for each of the first four lines, only award the point if the whole line is correct.

(b) (6 pts) Please provide the best case and worst case run times (Big-O) for each of the following three sorting algorithms, in terms of n, the number of elements being sorted.

Sort	**Best Case**	**Worst Case**
Merge Sort	O(nlgn)	O(nlgn)
Quick Sort	O(nlgn)	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$

Grading: 1 pt for each. Each is either correct or not correct. Accept if O() is left out.