Computer Science Foundation Exam

May 19, 2018

Section I A

DATA STRUCTURES

SOLUTION

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

Name: ____

UCFID: ____

NID: ____

Question #	Max Pts	Category	Score
1	5	DSN	
2	10	DSN	
3	10	ALG	
TOTAL	25	----	

You must do all 3 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>. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (5 pts) DSN (Dynamic Memory Management in C)

Suppose we have an array of structures containing information about our group for a group project. Each index should contain a group member's name and phone number. The structure is shown below: names are stored as dynamically allocated strings and phone numbers are stored as integers. When the semester is over, we will delete this array. Write a function called deleteGroup that will take in this array and delete all the information, freeing all the memory space that the array previously took up. Your function should take 2 parameters: a pointer to the beginning of the array and an integer indicating the number of group members. It should return a null pointer representing the now empty array.

```
typedef struct GroupMember {
     char *name;
     int phoneNumber;
} GroupMember;

GroupMember* deleteGroup (GroupMember  *group, int numMembers) {
     int i;

     for(i=0; i<numMembers; i++)                         //1 pt
          free(group[i].name);                           //2 pts

     free(group);                                        //1 pt

     return NULL;                                        //1 pt
}
```

2) (10 pts) ALG (Linked Lists)

Suppose we have a linked list implemented with the structure below. Write a function that will take in a pointer to the head of list and inserts a node storing -1 after each even value in the list. If the list is empty or there are no even values in the list, no modifications should be made to the list. (For example, if the initial list had 2, 6, 7, 1, 3, and 8, the resulting list would have 2, -1, 6, -1, 7, 1, 8, -1.)

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

void markEven(node *head) {

    node* tmp = head;

    while (tmp != NULL) {      // 2 pts iter whole list

        while (tmp != NULL && tmp->data%2 != 0) //3pts find next even
            tmp = tmp->next;

        if (tmp != NULL) {                    // 1 pt no null error
            node* newnode = malloc(sizeof(node));
            newnode->data = -1;               // 2 pts make new node
            newnode->next = tmp->next;        // 2 pts patch it into
            tmp->next = newnode;              // list
            tmp = newnode;
        }
    }
}
```

3) (10 pts) DSN (Stacks)

(a) (6 pts) Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (1, 2, and 3) in the infix expression.

A + B * (1 **(C / D) + E * F** 2 **)** 3 *** G**

	*	
	+	
(	(	
*	*	*
+	+	+
1	2	3

Resulting postfix expression:

A	**B**	**C**	**D**	**/**	**E**	**F**	*****	**+**	*****	**G**	*****	**+**							

Grading: 1 point for each stack, 3 points for the whole expression (partial credit allowed.)

(b) (4 pts) Whenever a recursive function is called, the function calls go onto a call stack. The depth of the call stack is the number of different recursive calls on the stack at a particular point in time, which indicates the number of different recursive calls that have started, but have not completed. What is the maximum stack depth of the call stack when the function fib(10) is executed? Is this maximum stack depth equal to the number of times the recursive function, fib, is called? Assume the implementation of the Fibonacci function shown below:

```
int fib(int n) {
    if (n < 2) return n;
    return fib(n-1) + fib(n-2);
}
```

Maximum Stack Depth: **10** **(Grading: 2 pts for 10 or 9, 1 pt to be within 3, 0 pts otherwise)**

Is Max Stack Depth equal to the # of recursive calls? YES [✓ **NO**]
(Circle the correct answer.) **Grading: 2 pts correct, 0 otherwise**

Computer Science Foundation Exam

May 19, 2018

Section I B

DATA STRUCTURES

SOLUTION

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

Name: ______________________________

UCFID: ______________________________

NID: ______________________________

Question #	Max Pts	Category	Score
1	10	DSN	
2	5	ALG	
3	10	ALG	
TOTAL	25		

You must do all 3 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. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (10 pts) DSN (Binary Search Trees)

Complete writing function shown below ***recursively***, so that it takes in a pointer to the root of a binary search tree, *root*, and an integer, *value*, and returns the number of nodes in the tree that are divisible by *value*. The struct used to store a node is shown below.

```
typedef struct bstNode {
   struct bstNode *left, *right;
   int data;
} bstNode;

int countDiv(bstNode *root, int value){

    if (root == NULL) return 0;              //2 pts

    // 4 pts, 2 pts for each recursive call.
    int res = countDiv(root->left, value) +
              countDiv(root->right, value);

    // 2 pts for checking divisibility, 1 pt for adding 1
    if (root->data % value == 0)
        res++;

    // 1 pt for returning.
    return res;
}
```

2) (5 pts) ALG (Heaps)

Suppose we are storing integers in a **Max-Heap** using the tree representation of heaps. The following tree shows the Max-Heap after 5 insertions. Show the result of inserting 109 into this heap, showing each step of the process. (Hint: You should draw 3 separate tree pictures.) Then, place each value from the resulting heap of 6 values into the array in the appropriate indexes corresponding to how a heap is typically stored in an array.

```
            100
           /   \
         90     98
        /   \
    60       71
```

```
      Step 1                      Step 2                     Step 3
       100                         100                        109
     /       \                   /       \                  /       \
  90           98      ->     90           109      ->    90          100
 /   \        /              /   \        /              /   \        /
60    71   109             60     71   98              60     71   98
```

Index	1	2	3	4	5	6
Heap Value	109	90	100	60	71	98

Grading: 1 pt for each drawing, 2 pts for correct array, 1 pt for minor error in the array, 0 pt for major error in array.

3) (10 pts) DSN (AVL Trees)

(a) (8 pts) Create an AVL tree by inserting the following values into an initially empty AVL Tree in the order given: 7, 8, 54, 13, 35, 66, 50, and 12. Show the state of the tree after each insertion.

```
   7          7           7*          8            8
               \           \         / \          / \
                8           8  ->   7   54       7   54
                             \                      /
                              54                   13

     8                 8                8*               35
    / \               / \              / \              /  \
   7   54*   ->      7   35           7   35    ->     8    54
       /                / \              / \          / \     \
      13               13  54           13  54       7  13     66
        \                                     \
         35                                    66

     35                35
    /  \              /  \
   8    54           8    54
  / \   / \         / \   / \
 7  13 50  66      7  13 50  66
                      /
                     12
```

Grading: Students should show each insertion step for **1 pt each.** Imbalances should be detected and corrected for after inserting 54, 35, and 66; detected at 7, 54, and 8 respectively.

(b) (2 pts) Draw the state of the tree after the deletion of the node containing the value 7.

```
      35
     /  \
   12    54
  / \    / \
 8  13  50  66
```

Deleting 7 creates an imbalance at 8 that must be corrected.

Grading: 1 pt for a valid BST without 7, **1 pt** for it being the correct BST without 7 (0 pts if either 7 is still in it or it's not a valid BST.)

Computer Science Foundation Exam

May 19, 2018

Section II A

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Score
1	10	ANL	
2	5	ANL	
3	10	ANL	
TOTAL	25		

You must do all 3 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. For each coding question, assume that all of the necessary includes (stdlib.h, stdio.h, math.h, string.h) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

Consider the following algorithm to find the smallest item in a list of *n* distinct integers:

1. Pick an element, *x*, from the list at random.
2. Go through every other element in the list. If an element is less than *x* put it in list 1, and if it's more than *x*, put it in list 2. (Note: Since all elements in the list are distinct, none will equal *x*.)
3. If list 1 is empty, then return *x*, since it's the smallest element. If list 1 is NOT empty, go back to the first step, only using list 1.

In terms of *n*, what is the best case run-time of this algorithm? In terms of *n*, what is the worst case run-time of this algorithm? Please give justifications (both words and equations) for both answers. (Note: 8 points out of the 10 come from the justifications and the actual Big-Oh answers are only worth 1 point each.)

The best case run time is O(n). In the best case, the very first time we pick a random element, it turns out to be the smallest. In this case, we do n-1 comparisons in step 2, and then when list 1 is empty, we produce our result.

The worst case run time is O(n^2). In the worst case what will happen is that we always pick the largest element in step 1. So, the first time, we do n-1 comparisons, yielding list 1 with n-1 values. This is followed by n-2 comparisons, yielding list 1 with n-2 values, and so forth. The total number of comparisons would be (n-1) + (n-2) + (n-3) + … + 1, or more formally, $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ comparisons. There are a few other steps that weren't counted (picking the random element and checking to see if list 1 is empty), but these are relatively minor steps. The function above can be expressed as O(n^2), since it's roughly $\frac{1}{2}n^2$.

Grading:

1 pt for best case answer, 3 pts for justification - should include possibility of x being minimum off the bat and the corresponding run-time analysis of that case, give partial as necessary.

1 pt for worst case answer, 5 pts for justification - 2 pts for describing worst case scenario, 2 pts for obtaining the appropriate summation, 1 pt for solving the summation and concluding with the run time.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm to process an array of size n takes O($n\sqrt{n}$) time. For n = 640,000, the algorithm runs in 256 milliseconds. How many ***seconds*** should the algorithm take to run for an input size of n = 1,000,000?

Let the algorithm with input array size n have runtime $T(n) = cn\sqrt{n}$, where c is some constant.

Using the given information, we have:

$$T(640000) = c(640000)\sqrt{640000} = 256ms$$

$$c(640000)(800) = 256ms$$

$$c(512 \times 10^6) = 256ms$$

$$c = \frac{256ms}{512 \times 10^6}$$

$$c = \frac{1ms}{2 \times 10^6}$$

Now, solve for the desired information:

$$T(1000000) = c(1000000)\sqrt{1000000}$$

$$= \frac{1ms}{2 \times 10^6} \times 10^6 \times 10^3$$

$$= \frac{10^3 ms}{2} = \frac{1second}{2} = \frac{1}{2}second$$

Grading: 2 pts solving for c, 2 pts for plugging 10^6 and canceling to get to 1000/2 ms, 1 pt to answer 1/2 second as the question requests.

3) (10 pts) ANL (Summations)

Recall that $\sum_{i=0}^{n-1} 2^i = 2^n - 1$.

(a) (8 pts) Using this result, determine a closed-form solution in terms of *n*, for the summation below.

(b) (2 pts) Determine the numeric value of the summation for n = 9.

$$\sum_{i=0}^{n}(\sum_{j=0}^{i-1} 2^j)$$

(a)

$$\sum_{i=0}^{n}(\sum_{j=0}^{i-1} 2^j) = \sum_{i=0}^{n}(2^i - 1)$$

$$= \sum_{i=0}^{n} 2^i - \sum_{i=0}^{n} 1$$

$$= 2^{n+1} - 1 - (n+1)$$

$$= 2^{n+1} - n - 2$$

(b) Plugging in n = 9 into the closed-form solution obtained in part (a), we get:

$$\underline{2^{9+1} - 9 - 2 = 1024 - 11 = \mathbf{1013}}$$

Grading: Part A -2 pts for inner sum, 2 pts split sum, 1 pt left sum, 2 pts right sum, 1 pt simplifying difference, Part B - 2 pts correct answer, 1 pt plug in correct but made an arithmetic error, 0 otherwise

Computer Science Foundation Exam

May 19, 2018

Section II B

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Score
1	10	DSN	
2	10	DSN	
3	5	ALG	
TOTAL	25		

You must do all 3 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. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (10 pts) DSN (Recursive Coding)

Consider writing a recursive method that raises a polynomial to an exponent, calculating each of its coefficients mod a given integer. One way this method could work is checking to see if the exponent is even. If so, raise the polynomial to half of the original power. Then, take that result (a polynomial) and multiply it by itself for the result. If the original exponent, *exp*, was odd, then we could simply first raise the polynomial to *exp*-1, and then take that result and multiply it by the original polynomial. Implement this algorithm recursively below. You are given the code for the multiply function and should call it accordingly. A polynomial is stored as an array of integers, where poly[i] is the coefficient to x^i. In the function signature, len is the length of the array poly, so poly is of degree len-1, exp is the exponent to which we are raising the polynomial and mod is the modulus by which we are calculating each coefficient.

```
int* power(int* poly, int len, int exp, int mod) {
    if (exp == 0) {
        int* res = malloc(sizeof(int));
        res[0] = 1;
        return res;
    }
    if (exp == 1) {
        int* res = malloc(len*sizeof(int));
        int i;
        for (i=0; i<len; i++) res[i] = poly[i]%mod;
        return res;
    }
    if (exp%2 == 0) {
        int* tmp = power(poly, len, exp/2, mod);

        int* prod = multiply(tmp, (len-1)*exp/2+1,

                             tmp, (len-1)*exp/2+1, mod)
        free(tmp);
        return prod;
    }
    int* tmp = power(poly, len, exp-1, mod);
    int* prod = multiply(tmp, (len-1)*(exp-1)+1,
                         poly, len, mod);
    free(tmp);
    return prod;
}

int* multiply(int* poly1, int len1, int* poly2, int len2, int mod) {
    int* res = calloc(len1+len2-1, sizeof(int));
    int i, j;
    for (i=0; i<len1; i++)
        for (j=0; j<len2; j++)
            res[i+j] = (res[i+j] + poly1[i]*poly2[j])%mod;
    return res;
}
```

Grading: 1 pt per slot, all or nothing per slot.

2) (10 pts) DSN (Sorting)

Consider sorting students, where each student has a first name, last name and a unique ID number. Assume all names contain uppercase letters only, so that strcmp does an alphabetic comparison for the purposes of this problem. Complete the code below so that it sorts students by last name (A to Z), breaking ties by first name (A to Z), and finally breaking ties between students with identical first and last names by ID number (smallest to largest). For example, if we have the students (EMILIO SANCHEZ 17), (ANDREA SANCHEZ 22), and (EMILIO SANCHEZ 10), the correct ordering would be ANDREA SANCHEZ, followed by EMILIO SANCHEZ 10, with EMILIO SANCHEZ 17 last.

```
typedef struct student {
    char first[20];
    char last[20];
    int ID;
} student;

void sort(student** list, int len) {
    int i,j;
    for (i=len-1; i>0; i--) {
        for (j=0; j<i; j++) {
            if (cmp(list[j], list[j+1]) > 0) {
                student* tmp = list[j];
                list[j] = list[j+1];
                list[j+1] = tmp;
            }
        }
    }
}

int cmp(student* a, student* b) {

    if (strcmp(a->last, b->last) != 0)
        return strcmp(a->last, b->last);

    if (strcmp(a->first, b->first) != 0)
        return strcmp(a->first, b->first);

    return a->ID - b->ID;
}
```

Grading: 3 pts for breaking ties properly by last name, 3 pts for breaking ties properly by first name, 4 pts for breaking ties by ID. If no string functions are used but the logic is correct via regular relational operators, then take off 4 pts total for not properly calling the functions.

3) (5 pts) ALG (Bitwise Operators)

What is the output of the following program?

```
#include <stdio.h>

int main() {

    int n = 182, i = 0;

    while (n > 0) {
        if ((n & (1<<i)) > 0) {
            printf("%d\n", (1<<i));
            n ^=(1<<i);
        }
        i++;
    }
    return 0;
}
```

```
2
4
16
32
128
```

Grading: 1 pt per correct number listed. 1 pt off per extra number listed, cap at 0.