Computer Science Foundation Exam

January 13, 2018

Section I A

DATA STRUCTURES

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

Name: _____

UCFID: _____

NID: _____

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	5	ANL	3	
3	10	DSN	7	
TOTAL	25		17	

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) (10 pts) DSN (Dynamic Memory Management in C)

The struct, `dataTOD`, shown below, is used to collect data from different devices connected to the CPU. Every time the data is updated a new buffer containing the structure's data is created and populated.

```
typedef struct dataTOD {
    int seconds;      // seconds since midnight
    double data;      // data sample
    char * dataName;  // data name (optional)
} dataTOD;
```

(a) (8 pts) Write the code necessary to create and initialize the members of `dataTOD` in a function named `init_dataTOD` that returns a pointer to the newly created buffer. Return NULL in the event a buffer cannot be created. Otherwise, set the seconds and data values according to the corresponding input parameters to `init_dataTOD`, dynamically allocate the proper space for dataName and then copy the contents of name into it (not a pointer copy) and a return a pointer to the newly created struct.

```
dataTOD * init_dataTOD(int sec, double val, char* name){









}
```

(b) (2 pts) Complete the function below so that it frees all the dynamically allocated memory pointed to by its formal parameter zapThis. You may assume that the pointer itself is pointing to a valid struct and its dataName pointer is pointing to a dynamically allocated character array.

```
void free_dataTOD(dataTOD *zapThis){




}
```

2) (5 pts) DSN (Linked Lists)

Given the linked list structure named `node`, defined in lines 1 through 4, and the function named `eFunction` defined in lines 6 through 14, answer the questions below.

```
 1 typedef struct node {
 2         int data;
 3         struct node * next;
 4 } node;
 5
 6 node* eFunction(node* aNode){
 7    if(aNode == NULL) return NULL;
 8    if(aNode->next == NULL) return aNode;
 9
10    node* rest = eFunction(aNode->next);
11    aNode->next->next = aNode;
12    aNode->next = NULL;
13    return rest;
14 }
```

(a) (1 pt) Is this function recursive? (Circle the correct answer below.)

YES NO

(b) (2 pts) What does the function eFunction do, in general to the list pointed to by its formal parameter, aNode?

(c) (2 pts) What important task does line 12 perform?

3) (10 pts) ALG (Stacks) Consider evaluating a postfix expression that only contained ***positive*** integer operands and the addition and subtraction operators. (Thus, there are no issues with order of operations!) Write a function that evalulates such an expression. To make this question easier, assume that your function takes an array of integers, `expr`, storing the expression and the length of that array, `len`. In the array of integers, all positive integers are operands while -1 represents an addition sign and -2 represents a subtraction sign. Assume that you have a stack at your disposal with the following function signatures. Furthermore, assume that the input expression is a valid postfix expression, so you don't have to ever check if you are attempting to pop an empty stack. Complete the evaluate function below.

```
void init(stack* s); // Initializes the stack pointed to by s.
void push(stack* s, int item); // Pushes item onto the stack pointed
                               // to by s.
int pop(stack* s); // Pops and returns the top value from the stack
                   // pointed to by s.

int eval(int* expr, int len) {

    stack s;
    init(&s);
    int i;









}
```

Computer Science Foundation Exam

January 13, 2018

Section I B

DATA STRUCTURES

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

Name: _____

UCFID: _____

NID: _____

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	5	ALG	3	
3	10	ALG	7	
TOTAL	25		17	

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) (10 pts) DSN (Binary Search Trees)

Write a ***recursive*** function to find the ***leaf node*** in a binary search tree storing the minimum value. (Thus, of all leaf nodes in the binary search tree, the function must return a pointer to the one that stores the smallest value.) If the pointer passed to the function is NULL (empty tree), the function should return NULL.

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

bstNode* find_min_leaf(bstNode* root) {




}
```

2) (5 pts) ALG (Binary Heaps)

The array below stores a minimum binary heap. Draw the tree version of the corresponding binary heap. Then, remove the minimum value and show the resulting heap, in tree form. (Note: Index 0 isn't shown because index 1 stores the value at the root/top of heap.)

Index	1	2	3	4	5	6	7	8	9	10	11
Value	2	16	3	22	17	12	13	23	30	18	20

3) (10 pts) ALG (AVL Trees)

Show the result of inserting the following values into an initially empty AVL tree:

10, 7, 3, 22, 16, 13, 5, 18, 20 and 19.

Draw a box around your result ***after each insertion.***

Computer Science Foundation Exam

January 13, 2018

Section II A

ALGORITHMS AND ANALYSIS TOOLS

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

Name: _____

UCFID: _____

NID: _____

Question #	Max Pts	Category	Passing	Score
1	10	ANL	7	
2	5	ANL	3	
3	10	ANL	7	
TOTAL	25		17	

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) (10 pts) ANL (Algorithm Analysis)

With proof, determine the Big-Oh run time of the function, f, below, in terms of the input parameter n:

```
int f(int array[], int n) {

    int i, t = 0, a = 0, b = n-1;
    while (a < b) {

        for (i=a; i<=b; i++)
            t += array[i];

        if (array[a] < array[(a+b)/2])
            b = (a+b)/2-1;
        else
            a = (a+b)/2+1;

    }
    return t;
}
```

In your work, you may use the following result: $\sum_{i=0}^{\infty}(\frac{1}{2})^i = 2$.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm processing an array of size n runs in $O(n^3)$ time. For an array of size 500 the algorithm processes the array in 200 ms. How long would it be expected for the algorithm to take when processing an array of size 1,500? Please express your answer in ***seconds.***

3) (10 pts) ANL (Summations and Recurrence Relations)

Using the iteration technique, find a tight Big-Oh bound for the recurrence relation defined below:

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2, \text{ for } n > 1$$
$$T(1) = 1$$

Hint: You may use the fact that $\sum_{i=0}^{\infty}(\frac{3}{4})^i = 4$ and that $3^{log_2 n} = n^{log_2 3}$, and that $log_2 3 < 2$.

Computer Science Foundation Exam

January 13, 2018

Section II B

ALGORITHMS AND ANALYSIS TOOLS

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

Name: _____

UCFID: _____

NID: _____

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	5	ALG	3	
3	10	DSN	7	
TOTAL	25		17	

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>.

1) (10 pts) DSN (Recursive Coding)

Write an ***efficient recursive*** function that takes in a ***sorted*** array `numbers`, two integers, `low` and `high`, representing indexes into the array, and another integer, `value`, and returns the index in the array where `value` is found in the array in between index `low` and `high`, inclusive. If `value` is NOT found in the array in between indexes `low` and `high`, inclusive, then the function should return -1.

```
int search(int numbers[], int low, int high, int value) {




}
```

2) (5 pts) ALG (Sorting)

(a) (3 pts) Explain why, in the worst case, Quick Sort runs more slowly than Merge Sort.

(b) (2 pts) In practice, Quick Sort runs slightly faster than Merge Sort. This is because the partition function can be run "in place" while the merge function can not. More clearly explain what it means to run the partition function "in place".

3) (10 pts) DSN (Backtracking)

Consider the problem of placing 8 kings on an 8 x 8 chessboard, so that no two of the kings can attack each other ***AND no two kings are on the same row or column***. (Recall that a King can move one space in each of the eight possible directions of movement: up, down, left, right or any of the four diagonals.) Complete the code skeleton below so that it prints out each solution to the 8 Kings problem. (Note: assume that the function print, which isn't included, prints out the solution that corresponds to a particular permutation of kings. For example, the permutation {2, 4, 6, 1, 3, 5, 7, 0} represents kings at the following locations (0, 2), (1, 4), (2, 6), (3, 1), (4, 3), (5, 5), (6, 7), and (7, 0).)

```
#include <stdio.h>
#include <math.h>
#define SIZE 8

void go(int perm[], int k, int used[]);
void print(int perm[]);

int main() {
    int perm[SIZE];
    int used[SIZE];
    int i;
    for (i=0; i<SIZE; i++) used[i] = 0;
    go(perm, 0, used);
    return 0;
}

void go(int perm[], int k, int used[]) {

    if ( ___________________ ) {
        print(perm);
        return;
    }

    int i;
    for (i=0; i<SIZE; i++) {

        if ( ________________________________________ ) continue;

        if ( ________________________________________ ) continue;

        perm[k] = ____ ;
        used[ _____ ] = 1 ;

        go(perm, _______, used);

        used[ i ] = ___ ;
    }
}
```