Computer Science Foundation Exam

August 31, 2019

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	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 <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)

Consider a binary search tree where each node contains some key integer value and data in the form of a linked list of integers. The structures are shown below: the tree nodes and list nodes are dynamically allocated. We are going to eventually upgrade the structure, and when we do so, all of the dynamically allocated memory will be deleted (including all of the linked lists). Write a function called `deleteTreeList` that will take in the root of the tree, freeing all the memory space that the tree previously took up. Your function should take 1 parameter: a pointer to the root and it should return a null pointer representing the now empty tree.

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

typedef struct treeNode {
    struct treeNode * left;
    struct treeNode * right;
    int value;
    listNode * head;
} treeNode;

treeNode * deleteTreeList (treeNode * root) {




}
```

2) (10 pts) DSN (Linked Lists)

An alternate method of storing a string is to store each letter of the string in a single node of a linked list, with the first node of the list storing the first letter of the string. Using this method of storage, no null character is needed since the next field of the node storing the last letter of the string would simply be a null pointer. Write a function that takes in a pointer to a linked list storing a string and returns a pointer to a traditional C string storing the same contents. Make sure to dynamically allocate your string in the function and null terminate it before returning the pointer to the string. Assume that a function, length, exists already that you can call in your solution, that takes in a pointer to a node and returns the length of the list it points to. The prototype for this function is provided below after the struct definition.

```
typedef struct node {
    char letter;
    struct node* next;
} node;

int length(node* head);

char* toCString(node * head) {
















}
```

3) (5 pts) ALG (Stacks)

Evaluate the following postfix expression shown below, using the algorithm that utilizes an operand stack. Put the value of the expression in the slot provided and show the state of the operand stack (in this case the stacks should just have numbers in them) at each of the indicated points in the expression:

3 4 2 1 + * 24 ^{A} 6 / 9 ^{B} 4 - ^{C} + - /

A **B** **C**

Value of the Postfix Expression: _________

Computer Science Foundation Exam

August 31, 2019

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	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 <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)

A modified BST node stores the `sum` of the `data` values in its sub-tree. **Complete** writing the insert function shown below ***recursively***, so that it takes in a pointer to the root of a binary search tree, *root*, and an integer, *value*, inserts a node storing value in it into the tree and returns a pointer to the root of the resulting tree. Notice that this task is more difficult than a usual binary tree insert since the sum values in several nodes must be updated as well. The struct used to store a node is shown below.

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

bstNode* insert(bstNode * root, int value){


    if (root == NULL) {
        bstNode* res = malloc(sizeof(bstNode));

        res->data = ________;

        res->sum = _________;

        res->left = ________;

        res->right = _______;
        return res;
    }

    if (value <= root->data)

        _________________________________________________ ;
    else
        _________________________________________________ ;

    _________________________________________________ ;

    return root;
}
```

2) (5 pts) ALG (Heaps)

The array below shows the storage of a **Min-Heap** in the ***middle*** of an insert operation.

(a) (1 pt) What was the element that was in the process of being inserted?

(b) (4 pts) Draw a picture of the heap as a balanced binary tree ***after*** the completion of the insertion of the item.

Note that Index 0 is not shown, because the root of the Heap is at index 1.

Index	1	2	3	4	5	6	7	8	9
Heap Value	1	3	7	8	5	2	11	10	12

Element in the process of being inserted: ______

Picture of the heap as a balanced binary tree ***after*** insertion is complete:

3) (10 pts) ALG (AVL Trees)

(a) (8 pts) An AVL tree stores the grades of the class (in between 1 and 100 inclusive). Create an AVL tree by inserting the following values into an initially empty AVL Tree in the order given: 98, 87, 92, 81, 75, 68, and 100. Show the state of the tree ***after*** each insertion and draw a box around each of these results.

(b) (2 pts) What is the fewest and most number of comparisons for looking for a valid grade that is ***not*** within this tree?

Fewest number of comparisons = __________

Most number of comparisons = ____________

Computer Science Foundation Exam

August 31, 2019

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	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)

Give the Big-O run-times for each of the following operations in terms of the variables given in the description. When a particular implementation is not explicitly stated, assume an efficient implementation is used. In order to earn full credit, you must provide a simplified, asymptotically tight bound. (For example, if O(n) was the correct answer, O(5n) and O(n^2) would not receive full credit, even though both are technically correct.)

a) Merging a sorted array of size ***m*** with a sorted array of size ***n*** into one sorted array. ____________

b) Creating a heap out of ***n*** unsorted integers. ____________

c) **Worst case** run-time of running a Quick Sort on ***n*** integers. ____________

d) Inserting an element to the front of a linked list with ***n*** elements. ____________

e) Deleting ***m*** items, one by one, from an **AVL** tree which originally contains ***n*** items ($n \geq m$) ____________

f) A sequence of ***p*** push operations onto a stack that originally had ***n*** elements on it. (Assume the stack has enough space to handle the sequence of push operations.) ____________

g) **Average case** run time of an insertion sort on ***n*** unsorted integers. ____________

h) Calculating a^b mod ***c***, using fast modular exponentiation, assuming that each multiply and each mod operation take O(1) time. ____________

i) Pre-order traversal of a binary tree with height ***h*** and ***n*** nodes. ____________

j) **Worst case** run-time for searching for an element in a binary search tree with ***n*** nodes. ____________

2) (5 pts) ANL (Algorithm Analysis)

An algorithm to process input data about n cities takes O(n!) time. For n = 10, the algorithm runs in 10 milliseconds. How many ***seconds*** should the algorithm take to run for an input size of n = 12? Put a box around your final answer.

3) (10 pts) ANL (Recurrence Relations)

Use the iteration technique to solve the following recurrence relation in terms of n:

$$T(n) = 2T(n-1) + 2^n, for\ all\ integers\ n \geq 1$$
$$T(0) = 1$$

Please give an **exact closed-form answer in terms of n,** instead of a Big-Oh answer.

Computer Science Foundation Exam

August 31, 2019

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	Score
1	10	DSN	
2	5	ALG	
3	10	DSN	
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)

The longest increasing subsequence problem is as follows: given a sequence of integers, find the largest subset of those integers such that within the sequence, those integers are in increasing order. For example, for the sequence **2**, 9, 4, **3**, 7, **5**, **6**, **8**, has some increasing subsequences of length 5 (one of these is highlighted) but none of length 6, so the length of the longest increasing subsequence of this sequence is 5.

In order to solve this problem recursively, we have to reformulate the problem a little bit. Namely, our input will be:

1. An array, ***values***, storing the original sequence
2. An integer, ***k***, representing that we want to only consider the values in the array upto index k, including it.
3. An integer, ***max***, representing the maximum value allowed in the increasing sequence.

Our recursive function will return the length of the longest increasing subsequence of ***values***[0..***k***] such that no value in the increasing subsequence exceeds max.

Complete the implementation of this ***recursive*** function below:

```
int lis(int* values, int k, int max) {










}
```

2) (5 pts) ALG (Sorting)

Quick Sort is not a stable sort. This means that if there are two elements in the input array, a[i] and a[j], are considered equal before the sort is executed with i < j, the element originally stored in a[j] may appear ***before*** the element originally stored in a[i], after the sort completes. Namely, the relative order of equivalent elements may not be maintained in the sort. (For example, if we are sorting only by last name and in the original list "Doug Adams" appeared before "Sandy Adams", then after the sort, it's possible that "Sandy Adams" could appear before "Doug Adams".)

The reason this is the case is due to the partition function used in Quick Sort. Explain why the partition function doesn't maintain the stability property mentioned above and provide a specific example where stability isn't maintained. In your example, you can use letters with subscripts and sort the letters in alphabetical order.

3) (10 pts) ALG (Bitwise Operators)

Imagine a dating website that asks each user 20 yes/no questions and stores their answers in a single integer in between 0 and 2^{20}-1, with the answer to the kth question stored in bit 2^{k-1}, with the bit 0 representing the answer no and the bit 1 representing the answer yes for the corresponding question. (So for example, if one person's answers to questions 1, 3 and 4 were yes and the rest of the answers were no, the integer 1101_2 = 13 would be used to represent her 20 answers to the questions.) Consider the problem of finding the "best match" for a client out of a list of prospective matches. We consider a match **A** for the client to be better than match **B** if she shares more answers on corresponding questions with **A** than **B**. Write a function that takes in an integer representing the client's answers, an array of integers storing the answers of potential matches, and the length of that array, which returns the index into the array storing the best match for that client. If there are multiple best matches, you may return the index associated with any of them. A function has been given that you may call in your code, if you find it useful.

```
int count(int mask);

int bestMatch(int client, int* matches, int length) {











}

int count(int mask) {
    int res = 0, i;
    for (i=0; i<32; i++)
        if ((mask & (1<<i)) != 0)
            res++;
    return res;
}
```