Computer Science Foundation Exam

August 28, 2021

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	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 <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)

Suppose we have an array to store all of the holiday presents we have purchased for this year. Now that the holidays are over and all the presents have been given out, we need to delete our list. Our array is a dynamically allocated array of structures that contains the name of each present and the price. The name of the present is a dynamically allocated string to support different lengths of strings. Write a function called delete_present_list that will take in the present array and free all the memory space that the array previously took up. Your function should take 2 parameters: the array called present_list and an integer, num, representing the number of presents in the list and return a null pointer representing the now deleted list. (Note: The array passed to the function may be pointing to NULL, so that case should be handled appropriately.)

```
struct present {
     char *present_name;
     float price;
};

struct present* delete_present_list(struct present* present_list, int
num) {









}
```

2) (5 pts) ALG (Linked Lists)

Suppose we have a singly linked list implemented with the structure below and a function that takes in the head of the list.

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

int whatDoesItDo (node * head) {
     struct node * current = head;
     struct node * other, *temp;

     if (current == NULL)
        return head;

     other = current->next;

    if (other == NULL)
        return head;

    other = other->next;
    temp = current->next;
    current->next = other->next;
    current = other->next;

    if (current == NULL) {
        head->next = temp;
        return head;
    }

    other->next = current->next;
    current->next = temp;

    return head;
}
```

If we call whatDoesItDo(head) on the following list, show the list after the function has finished.

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

3) (10 pts) ALG (Queues)

Suppose we wish to implement a queue using an array. The structure of the queue is shown below.

```
struct queue {
    int *array;
    int num_elements;
    int front;
    int capacity;
};
```

The queue contains the array and three attributes: the current number of elements in the array, the current front of the queue, and the maximum capacity. Elements may be added to the queue not just at the end of the array but also in the indices at the beginning of the array before front. Such a queue is called a circular queue.

Write a function to implement the dequeue functionality for the queue, while ensuring that no null pointer errors occur. Your function should take in 1 parameter: a pointer to the queue. Your function should return the integer that was dequeued. If the queue is NULL or if there are no elements to dequeue, your function should return 0.

```
int dequeue(struct queue * q) {









}
```

Computer Science Foundation Exam

August 28, 2021

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	ALG	
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.

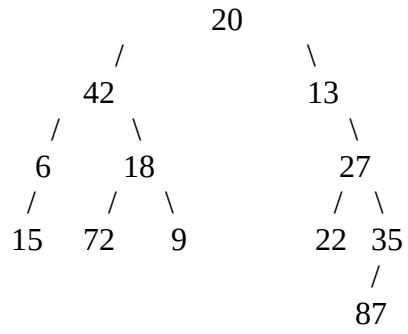
1) (10 pts) ALG (Binary Trees)

Consider a function that takes in a pointer to a binary tree node and returns a pointer to a binary tree node defined below:

```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} btreenode;

btreenode* somefunction(btreenode* root) {
    if (root == NULL) return NULL;
    somefunction(root->left);
    somefunction(root->right);
    btreenode* tmp = root->left;
    root->left = root->right;
    root->right = tmp;
    return root;
}
```

Let the pointer tree point to the root node of the tree depicted below:



If the line of code `tree = somefunction(tree)` were executed, draw a picture of the resulting binary tree that the pointer tree would point to.

2) (5 pts) ALG (Heaps)

Suppose we construct a minheap where each node contains a string, and we order our strings according to the following rules:

1. Given strings *a* and *b*, we say $a < b$ if *a* has fewer characters than *b*.
2. If two strings *a* and *b* have the same length, we say $a < b$ if *a* comes before *b* in alphabetical order.

Furthermore, suppose all the strings in our minheap contain lowercase letters only (so, no punctuation, spaces, uppercase letters, and so on), and we do not allow any duplicate strings into the minheap.

Given two arbitrary strings in our minheap, *x* and *y*, can we safely say that if *x* is a prefix of *y*, then *y* must be in one of *x*'s subtrees? Note that *x* may not be stored at the root of the minheap. If so, explain why this must be the case. If not, draw a minheap of strings that very clearly shows this is not necessarily the case (and clearly label which string is *x* and which string is *y* in your counterexample).

3) (10 pts) ALG (AVL Trees)

Draw an AVL tree **of integers** and designate a single node in the AVL tree such that, if that node were to be deleted, two rebalance operations (not one double rotation, but two separate operations at two different nodes) would occur. Clearly label the node to delete which would precipitate those operations and show the result of deleting that node. (Thus, you should have two drawings, a before drawing of the original tree with the node to be deleted clearly designated, and an after drawing showing what the tree looks like after the node is deleted and goes through 2 rebalance operations.)

Computer Science Foundation Exam

August 28, 2021

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	10	ANL	
3	5	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 <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.h, stdio.h, math.h, string.h) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

Consider the following problem: You are given a set of weights, $\{w_0, w_1, w_2, \ldots, w_{n-1}\}$ and a target weight T. The target weight is placed on one side of a balance scale. The problem is to determine if there exists a way to use some subset of the weights to add on either side of the balance so that the scale will perfectly balance or not. For example, if T = 12 and the set of weights was {6, 2, 19, 1}, then one possible solution would be to place the weights 6 and 1 on the same side of the balance as 12 and place the weight 19 on the other side.

Below is a function that solves this problem recursively, with a wrapper function to make the initial recursive call. In terms of n, the size of the input array of weights, **with proof,** determine the **worst case** run time of the wrapper function. **(Note: Since only the run time must be analyzed, it's not necessary to fully understand WHY the solution works. Rather, the analysis can be done just by looking at the structure of the code.)**

```
int makeBalance(int weights[], int n, int target) {
    return makeBalanceRec(weights, n, 0, target);
}

int makeBalanceRec(int weights[], int n, int k, int target) {
    if (k == n) return target == 0;
    int left = makeBalanceRec(weights, n, k+1, target-weights[k]);
    if (left) return 1;
    int right = makeBalanceRec(weights, n, k+1, target+weights[k]);
    if (right) return 1;
    return makeBalanceRec(weights, n, k+1, target);
}
```

2) (10 pts) ANL (Algorithm Analysis)

A program processing an array of size 100 took 50 ms to finish and on an array of size 1000 it took 75 ms to finish. What Big Oh runtime would be most reasonable for the program? (Hint: make a couple guesses to the function and see if those guesses are consistent with the run-times listed.)

3) (5 pts) ANL (Summations)

What is the simplified closed form of the following summation in terms of n? Please show each step of work. **(Note: the bounds on the inner summation are NOT a misprint!!!)**

$$\sum_{a=0}^{n}\left(\sum_{b=a}^{a} 4b\right)$$

Computer Science Foundation Exam

August 28, 2021

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	5	DSN	
2	10	DSN	
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 <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 (Recursive Coding)

Write a ***recursive*** function that returns the number of bits set to 1 in the binary representation of its input parameter, *n*. For example numBitsOn(17) should return 2, since 17 in binary is 10001. **(Note: In order to receive full credit, your function's run time must be O(b), where b is the total number of bits in n. Since this isn't the bitwise operator question, you don't HAVE to use bitwise operators for full credit, but that's probably the most natural route to the solution.)**

```
int numBitsOn(int n) {




}
```

2) (10 pts) DSN (Sorting)

The critical part of the Quick Sort algorithm is the partition. One important part of the partition is the selection of the partition element. In general, it's better to have a "middle value" relative to the other values in the subarray to be sorted to be chosen as the partition element. One simple strategy to improve the probability of a good partition element is to select three items at random from the subarray to be sorted and use the middle value of those three as the partition element. In this particular problem, complete the function below so that it takes in an array, a low index to the array and a high index to the array, designating a subarray, generates three random indexes in between low and high inclusive (let these be indexA, indexB and indexC), and the returns the corresponding index (either indexA, indexB or indexC) to the middle value of the three designated values array[indexA], array[indexB] or array[indexC]. A function randInt(a, b) is provided for you to call, which returns a random integer in between a and b, inclusive. **(Note: To make the problem a bit easier, there is no need to make sure that indexA, indexB and indexC are all distinct.)**

```
// Pre-condition: low and high are valid indexes into array with
//                high - low > 10
int getPartitionIndex(int array[], int low, int high) {









}

int randInt(int a, int b) {
    int base = ((rand()%32768)<<15) + (rand()%32768);
    return a + base%(b-a+1);
}
```

3) (10 pts) DSN (Backtracking)

Consider printing out all strings of x A's and y B's, where x ≥ y-1 such that no two consecutive letters are Bs, in alphabetical order. For example, if x = 5 and y = 3, one of the valid strings printed would be AABABABA. One way to solve this problem would be to use backtracking, where a string is built up, letter by letter (first trying A, then trying B in the current slot). **Complete the code below to implement this backtracking solution idea.** The correct condition for when you can place As is already in the code. (Hint: You can only place Bs if there are Bs left to place. If there are Bs left, then you must ensure that if there is a previous letter, it is not a B.)

```
#include <stdio.h>
#include <stdlib.h>

void printAll(char buffer[], int k, int a, int b);
void printWrapper(int x, int y);

// Prints all strings with exactly x A's and y B's in alphabetical
// order.
void printWrapper(int x, int y) {
    char* buffer = malloc(sizeof(char)*(x+y+1));
    buffer[x+y] = '\0';
    printAll(buffer, 0, x, y);
    free(buffer);
}

void printAll(char buffer[], int k, int x, int y) {

    if (x == 0 && y == 0) {
        printf("%s\n", buffer);
        return;
    }

    if (x > y-1) {
        buffer[k] = 'A' ;

        printAll(buffer, ______ , ______ , ______ );
    }

    if ( _______ && ( _______ || (_______ && ____________________) ) ){

        buffer[k] = 'B' ;

        printAll(buffer, ______ , ______ , ______ );
    }
}
```