Computer Science Foundation Exam

May 18, 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)

Suppose we have an array of structures containing information about Cartesian points. The struct shown below contains two integers, one for the x coordinate and one for the y coordinate. For this problem, write a function, createPoints, to create some random Cartesian points with each coordinate set to a random integer in between 0 and 10, inclusive.

createPoints takes in the number of points to be created, *numPoints*. Your function should dynamically allocate an array of *numPoints* CartPoints structs and set each of their x and y coordinates with pseudorandom integer values in between 0 to 10, inclusive. You may assume that the random number generator has been seeded already. Your function should return a pointer to the array that was created and initialized.

```
typedef struct CartPoint {
     int x;
     int y;
} CartPoint;

CartPoint* createPoints(int numPoints) {
     int i;









}
```

2) (10 pts) ALG (Linked Lists)

Suppose we have a queue implemented as a doubly linked list using the structures shown below with head pointing to node at the front of the queue and tail pointing to the node at the end of the queue.

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

typedef struct queue {
     int size;
     node *head, *tail;
} queue;
```

Write an enqueue function for this queue. If the queue is already full, return 0 and take no other action. If the queue has not been created yet, return 0 and take no other action. If the queue isn't full, enqueue the integer `item` into the queue, make the necessary adjustments, and return 1. Since there is no fixed size, the queue will be considered full if a new node can't be allocated.

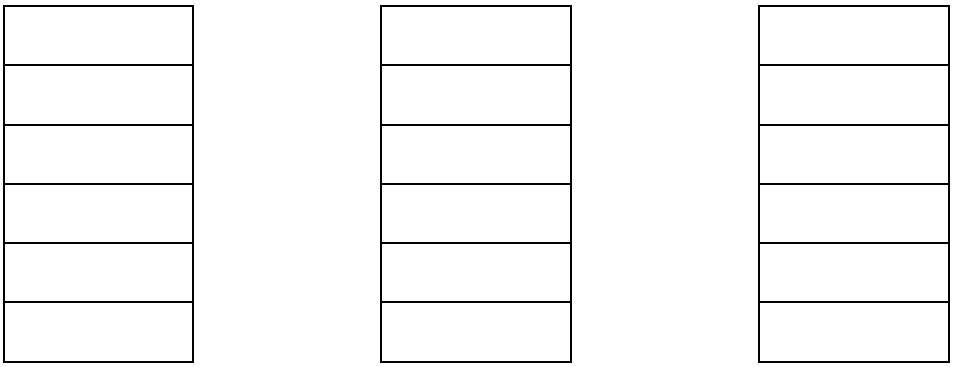
```
int enqueue(queue *thisQ, int item) {

     struct node *newNode = ____________________________________ ;

     if(thisQ == NULL) return 0;

     if(newNode == NULL) return 0;

     newNode->data = _________;

     newNode->next = _________;

     thisQ->size = ____________________;

     if(thisQ->head == NULL) {
          newNode->prev = ___________;
          thisQ->head = _____________;
          thisQ->tail = _____________;
          return 1;
     }

     ____________________________________________________;

     ____________________________________________________;

     ____________________________________________________;
     return 1;
}
```

3) (5 pts) DSN (Stacks)

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.

1 2 3

A + (B - C * (D + E)) - F * G



1 2 3

Resulting postfix expression:

Computer Science Foundation Exam

May 18, 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: _____

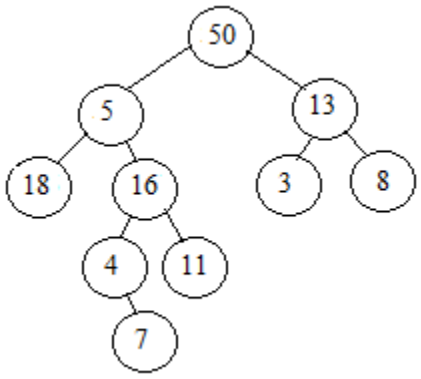
Question #	Max Pts	Category	Score
1	10	ALG	
2	5	ALG	
3	10	DSN	
TOTAL	25		

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) ALG (Binary Trees)

What does the function call `solve(root)` print out if root is pointing to the node storing 50 in the tree shown below? The necessary struct and function are provided below as well. Please fill in the blanks shown below. (Note: the left pointer of the node storing 50 points to the node storing 5, and all of the pointers shown correspond to the direction they are drawn in the picture below.)




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

int solve(bstNode* root) {

    if (root == NULL) return 0;

    int res = root->data;
    int left = solve(root->left);
    int right = solve(root->right);

    if (left+right > res)
        res = left+right;

    printf("%d, ", res);
    return res;
}
```

_____, _____, _____, _____, _____, _____, _____, _____, _____, _____,

2) (5 pts) ALG (Hash Tables)

Insert the following numbers (in the order that they are shown from left to right) into a hash table with an array of size 10, using the hash function, H(x) = x mod 10.

234 344 483 564 814

Show the result of the insertions, assuming any hash collisions are resolved through **quadratic probing.**

Index	**Value**
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

3) (10 pts) DSN (Tries)

In many word games, the player is given some tiles with letters and must form a word with those tiles. Given a trie that stores a dictionary of valid words and a frequency array storing information of the tiles a player has, determine the number of unique words she can form with those tiles. Complete the function shown below to solve the given problem. Note: the entry in freq[i] represents the number of tiles with the letter 'a' + i. **(Hint: recursing down the trie is exactly like placing a tile down, which means updating the freq array. When you have finished "trying a tile" you have to put it back into your pool, which means editing the freq array again.)**

```
typedef struct TrieNode {
    struct TrieNode *children[26];
    int flag; // 1 if the string is in the trie, 0 otherwise
} TrieNode;

int countWords(TrieNode* root, int freq[]) {

    int res = _____________________ ;

    int i;
    for (i=0; i<26; i++) {


        if ( _____________________ || _____________________________ )
            continue;


        ____________________________________________ ;


        res += ____________________________________________________ ;


        ____________________________________________ ;
    }

    return res;
}
```

Computer Science Foundation Exam

May 18, 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 <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 storing a table with indexes 0 to N-1, where $N = k^2$, for some positive integer k, that starts with all entries equal to 0 and allows two types of operations: (1) adding some value to a particular index, and (2) querying the sum of all the values in the table from index 0 through index *m*, for any positive integer $m < N$. One way to implement a "table" to handle these two operations is to store two separate arrays, *groups*, of size k and *freq*, of size N. *freq stores* the current value of each index in the table. For the array *groups*, index i ($0 \le i < k$) stores the sum of the values in *freq* from index ik to index (i+1)k-1. (For example, if N = 25, then *groups[2]* stores the sum of the values of freq, from *freq[10]* through *freq[14]*, inclusive.

Determine, ***with proof***, the run-time of implementing operation (1) on this table using this storage mechanism and determine, ***with proof***, the run-time of implementing operation (2) on this table using this storage mechanism. (For example, if N = 100 and we had a query with *m* = 67, to get our answer we would add *groups[0], groups[1], groups[2], groups[3], groups[4], groups[5], freq[60], freq[61], freq[62], freq[63], freq[64], freq[65], freq[66]* and *freq[67]*. Notice that since the ranges 0-9, 10-19, 20-29, 30-39, 40-49, and 50-59 are fully covered in our query, we could just use the groups array for each of those sums. We only had to access the freq array for the individual elements in the 60s.)

Your answers should be Big-Oh answers in terms of N as defined above.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm to process a query on an array of size n takes $O(\sqrt{n})$ time. For $n = 10^6$, the algorithm runs in 125 milliseconds. How many ***seconds*** should the algorithm take to run for an input size of n = 64,000,000?

3) (10 pts) ANL (Recurrence Relations)

Use the iteration technique to find a Big-Oh bound for the recurrence relation below. Note: you may find the following mathematical results helpful: $2^{log_3 n} = n^{log_3 2}$, and $\sum_{i=0}^{\infty}(\frac{2}{3})^i = 3$. You may use these without proof in your work below.

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

Computer Science Foundation Exam

May 18, 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 <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 (Recursive Coding)

Consider writing a recursive method that counts the number of paths from a starting (x, y) location on the Cartesian plane to an ending (x, y) location. Let the starting location be *(sx, sy)* and the ending location be *(ex, ey)*, where all four coordinates are integers with $sx \leq ex$ and $sy \leq ey$, and for each step on a valid path, either 1 must get added to the current x coordinate or 1 must get added to the current y coordinate. In addition, some given locations are disallowed as intermediate locations on the path. **Complete the function shown below** to solve this task. The input to the function takes in *sx*, *sy*, *ex*, *ey* and a two dimensional integer array named *allowed*, such that *allowed[x][y]* is set to 1 if a path is allowed to go on coordinate (x, y) or set to 0 otherwise. It is guaranteed that *(sx, sy)* and *(ex, ey)* are coordinates which are both inbounds and an inbounds function is provided for you. It's not guaranteed that both *(sx, sy)* and *(ex, ey)* are valid locations to be on. In this case, the answer is 0.

```
#define N 10
int inbounds(int x, int y);

int numpaths(int sx, int sy, int ex, int ey, int allowed[][N]) {

    if (!allowed[sx][sy]) return ___;

    if (sx > ex || sy > ey) return ___;

    if (sx == ex && sy == ey) return  ___;

    int res = ___ ;

    if ( _____________________________ )

        res += numpaths(____, ____, ____, ____, ___________);

    if ( _____________________________ )

        res += numpaths(____, ____, ____, ____, ___________);

    return res;
}

int inbounds(int x, int y) {
    return x >= 0 && x < N && y >= 0 && y < N;
}
```

2) (5 pts) DSN (Sorting)

In both Merge Sort and Quick Sort, in class we are taught to break down the sorting problem recursively such that the base case is a subarray of size 1 (or 0). It turns out that for both, on average, the implementation is *faster* if we have a base case with a subarray of size in between 20 and 50 and use a $O(n^2)$ sort (typically insertion sort) to sort the base case subarray. Even though insertion sort is $O(n^2)$, why does this modification to the algorithm result in a speed up for both Merge Sort and Quick Sort?

3) (10 pts) ALG (Backtracking)

Consider an arbitrary permutation of the integers 0, 1, 2, …, *n*-1. We define the "jumps" in a permutation array *perm* to be the set of values of the form *perm[i] - perm[i-1]*, with $1 \le i \le n-1$. For this problem you will write a backtracking solution count the number of permutations that can be created given a limited set of jumps. The function will take in arrays *perm*, representing the current permutation array, *used*, storing which items were used in the current permutation, *k*, the number of fixed items in the current permutation, *jumps*, an array storing the valid jumps allowed, and *len*, representing the length of the *jumps* array. The length of the *perm* and *used* arrays will be the constant N. Note that the jumps array contains both positive and negative values. For example, the permutation 3, 0, 2, 1 has the following jumps: -3, 2 and -1. **Complete the framework that has been given below to solve the problem.**

```
#include <stdio.h>
#define N 10

int numperms(int perm[], int used[], int k, int* jumps, int len) {
    int i, j, res = 0;

    if (k == N) return ___;
    for (i=0; i<N; i++) {

        if (used[i]) _______________;

        int flag = 0;
        if (k == 0)
            flag = ___;
        else {
            for (j=0; j < ____; j++)

                if ( _____________ == jumps[j])

                    flag = ___;
        }

        if (flag) {
            used[i] = ___;
            perm[k] = ___;
            res += numperms(perm, used, _____, jumps, len);
            used[i] = ___;
        }
    }

    return res;
}
```