Computer Science Foundation Exam

May 21, 2022

Section A

BASIC DATA STRUCTURES

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

Name: _____

UCFID: _____

Question #	Max Pts	Category	Score
1	5	ALG	
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) ALG (Dynamic Memory Management in C)

The following function has 5 memory management issues, each one occurring on a different one of the 17 labeled lines of code. Please clearly list which five lines of code have the errors on the slots provided below. **Please list exactly five unique line numbers in between 1 and 17, inclusive. <u>An automatic grade of 0 will be given to anyone who lists MORE than 5 line numbers.</u>**

```
int n = 10;                                    // line 1
int *p1, *p2, *p3, **p4;                       // line 2
char str1[100] = "test string";                // line 3
char *str2;                                    // line 4
strcpy(str2, str1);                            // line 5
p1 = (int *)malloc(n * sizeof(int));           // line 6
p2 = (int *)malloc(n * sizeof(int));           // line 7
for(int i=0; i<n; i++)                         // line 8
     p1[i] = rand()%100;                       // line 9
p2 = p1;                                       // line 10
*p3 = 50;                                      // line 11
p4 = (int **) malloc(n * sizeof(int*));        // line 12
for(int i=0; i<n; i++)                         // line 13
     p4[i] = -5;                               // line 14
free(p1);                                      // line 15
free(p2);                                      // line 16
free(p4);                                      // line 17
```

Lines with Memory Management Errors: _____ , _____ , _____ , _____ , _____

2) (10 pts) DSN (Linked Lists)

The structure of each node of a singly linked list is shown below.

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

Write a function insertAfterN, that takes the head of a linked list, and two integers M and N (M ≠ N) and inserts M after all the nodes containing N.

For example, if M = 200 and N = 6, the linked list 3, 6, 4, 6, 6, 5 will be changed to 3, 6, 200, 4, 6, 200, 6, 200, 5.

```
void insertAfterN(node* head, int M, int N) {




}
```

3) (10 pts) DSN (Stacks)

A word is considered a palindrome if the reverse of the word is the same as the original word. For example: the word “test” is not a palindrome as its reverse “tset” is not the same as “test”. On the other hand, the word “racecar” is a palindrome as its reverse is the same as “racecar”. Some other examples of palindromes are “hannah”, “level”, “madam”, and “yay.”

Write a function that will take a string in the parameter and returns 1, if the string is a palindrome, otherwise returns 0. **You have to use stack operations during this process.** (Credit isn’t awarded for correctly solving the problem, but for utilizing the stack in doing so.)

Assume the following stack definition and the functions already available to you. The stack will be extended automatically if it gets full (so you, don’t have to worry about it). The top of the stack is controlled by your push and pop operation as usual stack operations.

```
void initialize(stack* s); // initializes an empty stack.
int push(stack* s, char value); //pushes the char value to the stack
int isEmpty(stack* s); // Returns 1 if the stack is empty, 0 otherwise.
char pop(stack* s); // pops and returns character at the top of the stack.
char peek(stack* s); // returns character at the top of the stack.
```

Note: pop and peek return 'I' if the stack s is empty.

```
int isPalindrome(char *str) {
    struct stack s;
    initialize(&s);
    int len = strlen(str);









}
```

Computer Science Foundation Exam

May 21, 2022

Section B

ADVANCED DATA STRUCTURES

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

Name: _____

UCFID: _____

Question #	Max Pts	Category	Score
1	10	DSN	
2	10	ANL	
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 (Binary Trees)

Demonstrate your understanding of recursion by rewriting the following recursive function, which operates on a binary tree, as an **iterative** function. In doing so, you must abide by the following restrictions:

1. Do not write any helper functions in your solution.
2. Do not make any recursive calls in your solution.

```
int foo(node *root) {
    if (root == NULL) return 1;
    if (root->left == NULL && root->right == NULL) return 2;
    if (root->left == NULL) return 3 * foo(root->right);
    if (root->right == NULL) return 4 * foo(root->left);
    if (root->right->data > root->left->data) return 5 * foo(root->right);
    return 6 * foo(root->left);
}

int iterative_foo(node *root) {









}
```

2) (10 pts) ANL (Hash Tables)

This question asks you to explore the **best-** and **worst-case** runtimes for adding *r* new elements to a hash table that already contains *q* elements. In answering the questions below, assume the following:

1. Generating the initial hash value for any given key takes O(1) time.
2. We are using quadratic probing.
3. Our hash table is at least half empty, and the length of the table is prime.
4. There is enough space in the hash table to allow for all *r* new elements to be inserted without triggering a table expansion.
5. If the specific placement of the *q* elements that are already in the hash table is relevant, you may assume that they are placed in a way that would facilitate the situation you are describing that leads to the best- or worst-case runtime for the *r* new elements being added to the table.

Note that this question is not just asking for the runtime for adding a single element. We want the runtime for adding **all** *r* elements to the hash table. While the *q* elements already in the table may impact the runtime for adding the *r* new elements, you do not have to account for the runtime it took to add those *q* elements. Focus only on the cost of adding the *r* **additional** elements.

a. (2 pts) In big-oh notation, what is the **best-case** runtime for adding *r* new elements to the table?

b. (1 pt) What situation leads to the best-case runtime you listed in part (a)?

c. (5 pts) In big-oh notation, what is the **worst-case** runtime for adding *r* new elements to the table?

d. (2 pts) What situation leads to the worst-case runtime you listed in part (c)?

3) (5 pts) ALG (AVL Trees)

Suppose we randomly shuffle the six words in the list below and insert them into an AVL tree. (In other words, we insert them in random order – not necessarily the order given – with each of those words ending up in the AVL tree exactly once.)

Fill in the blank next to each word to indicate whether it could **ever** end up at the root of the resulting AVL tree ("yes") or not ("no"). (If you answer "no" for a given word, you are saying it could **never** end up at the root of the resulting AVL tree.)

You may assume the AVL tree is ordered alphabetically. So, all the words in the left subtree of "apple" would have to come before "apple" in alphabetical order, and all the words in its right subtree would have to come after "apple" in alphabetical order.

_____ apple

_____ mango

_____ papaya

_____ banana

_____ mulberry

_____ blueberry

^
Fill in each blank with "**yes**" or "**no**" to indicate whether the word could serve as the root of an AVL tree that results from shuffling these six words and inserting them into an AVL tree in random order.

Computer Science Foundation Exam

May 21, 2022

Section C

ALGORITHM ANALYSIS

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

Name: _____

UCFID: _____

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)

What is the worst case Big-Oh runtime for the function **f,** in terms of its input parameter n? You may assume that the array pointed to by arr is of length n. (Grading note: 2 pts will be awarded for the answer, 8 pts for the proof of the answer. Your proof must include either summations or recurrence relations related to the code below.)

```
int f(int* arr, int n, int minVal) {
    return fHelp(arr, 0, n-1, minVal);
}

int fHelp(int* arr, int low, int high, int minVal) {
    if (low > high) return 0;
    if (low == high) return arr[low] >= minVal;

    int mid = (low+high)/2;
    int left = fHelp(arr, low, mid, minVal);
    int right = fHelp(arr, mid+1, high, minVal);
    int res = left;
    if (right > left)
        res = right;

    int alt = 0, i;
    for (i=mid; i>=low; i--) {
        if (arr[i] < minVal) break;
        alt++;
    }
    for (i=mid+1; i<=high; i++) {
        if (arr[i] < minVal) break;
        alt++;
    }

    if (alt > res) res = alt;
    return res;
}
```

2) (5 pts) ANL (Algorithm Analysis)

A program that runs an O(Nlog(N)) algorithm to sort an array of N polygons takes 10 seconds to sort 1,000,000 polygons. How long, **in milliseconds**, would it be expected for the program to take when sorting 1,000 polygons?

3) (10 pts) ANL (Recurrence Relations)

Using the iteration technique, determine a closed-form solution for the following recurrence relation in terms of n. Note: Your answer should be **EXACT** and not a Big-Oh bound.

$$T(0) = 1$$
$$T(n) = 4T(n-1) + 2^n, for\ integers\ n > 0$$

Computer Science Foundation Exam

May 21, 2022

Section D

ALGORITHMS

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

Name: _____

UCFID: _____

Question #	Max Pts	Category	Score
1	10	DSN	
2	10	ALG	
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 (Recursive Coding)

The Towers of Hanoi Problem involves starting with a tower of ***n*** disks (of all different radii) on a single pole and transferring those disks to a different pole. There are only 3 poles that can hold the disks. The disks can only be moved one at a time, from one pole to another, and a disk must always be the only disk on the pole or be placed on top of a disk that is larger. The initial configuration of disks starts with all of the disks sorted in order on a single pole, with the smallest disk on top. For the purposes of this problem, let the radii of the disks be 1 cm, 2 cm, 3 cm, …, ***n*** cm. Let the cost of a single move simply equal the radius of the disk being moved. Define the cost of a solution to the puzzle to be the sum of the costs of the moves to solve the puzzle. Complete the function below, ***using recursion***, so that it takes in a single integer, ***n,*** the number of disks for the puzzle, and returns the minimal cost for transferring the ***n*** disks from the starting pole to either of the other poles. (Note: the actual value of the starting and ending poles don't affect the answer, so long as they are different poles. The code is relatively short and one can do some math to get an O(1) solution, but what is being tested here is the ability to take a problem, break it down recursively, and implement that solution. Thus, the grading criteria described below is focused on rewarding the skill that's being tested, though an alternate, more efficient solution exists.)

Grading Note: Any iterative or non-recursive solution will get a maximum of 3 points, a recursive solution that takes $\theta(2^n)$ time will get a maximum of 7 points. To receive full credit, your solution must be recursive and run in $\theta(n)$ time, where n represents the input value to the function.

```
int towersCost(int n) {




}
```

2) (10 pts) ALG (Sorting)

(a) (5 pts) Consider running an Insertion Sort on the array shown below. How many swaps will execute for the duration of the algorithm running on the array shown below? Explain how you got your answer.

35	25	15	29	39	22	19	6	21

Reasoning:

Number of Swaps: _____

(b) (5 pts) List the **<u>average case</u>** run time of each of the following sorting algorithms, in terms of n, the number of items being sorted. (Please provide Big-Oh bounds.)

(i) Insertion Sort _____

(ii) Selection Sort _____

(iii) Heap Sort _____

(iv) Merge Sort _____

(v) Quick Sort _____

3) (5 pts) ALG (Bitwise Operators)

Determine the value of each of these arithmetic expressions in C. Please use the space below for your scratch work.

(i) 43 | 96 _____

(ii) 117 & 74 _____

(iii) 76 ^ 49 _____

(iv) 11 << 2 _____

(v) 330 >> 8 _____