Computer Science Foundation Exam

January 12, 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	Passing	Score
1	10	DSN	7	
2	10	ALG	7	
3	5	ALG	3	
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 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 (Dynamic Memory Management in C)

This problem relies on the following struct definition:

```
typedef struct Employee
{
  char *first; // Employee's first name.
  char *last;  // Employee's last name.
  int ID;      // Employee ID.
} Employee;
```

Consider the following function, which takes three arrays – each of length *n* – containing the first names, last names, and ID numbers of *n* employees for some company. The function dynamically allocates an array of *n* Employee structs, copies the information from the array arguments into the corresponding array of structs, and returns the dynamically allocated array.

```
Employee *makeArray(char **firstNames, char **lastNames, int *IDs, int n)
{
  int i;
  Employee *array = malloc(__________________________________________);

  for (i = 0; i < n; i++)
  {
    array[i].first = malloc(_______________________________________________);

    array[i].last = malloc(________________________________________________);

    strcpy(array[i].first, firstNames[i]);
    strcpy(array[i].last, lastNames[i]);
    array[i].ID = IDs[i];
  }

  return array;
}
```

a) Fill in the blanks above with the appropriate arguments for each *malloc()* statement.

b) Next, write a function that takes a pointer to the array created by the *makeArray()* function, along with the number of employee records in that array (*n*) and frees all the dynamically allocated memory associated with that array. The function signature is as follows:

```
void freeEmployeeArray(Employee *array, int n)
{
```

2) (10 pts) ALG (Linked Lists)

Consider the following code:

```
void doTheThing(node *head, node *current)
{
  if (current == NULL)
    return;

  else if (current == head->next)
  {
    if (current->data == head->next->next->data)
      doTheThing(head, head->next->next->next);
    else if (current->data == head->next->next->data + 1)
      doTheThing(head, head->next->next->next->next);
    else if (current->data == head->next->next->data + 5)
      doTheThing(head, current->next->next->next);
    else if (current->data == head->next->next->data + 10)
      doTheThing(head, head->next);
    else
      doTheThing(head, current->next);
  }

  else
    doTheThing(head, current->next);
}
```

Draw a linked list that simultaneously satisfies **both** of the following properties:

1. The linked list has **exactly four nodes**. Be sure to indicate the integer value contained in each node.

2. If the linked list were passed to the function above, the program would either crash with a segmentation fault, get stuck in an infinite loop, or crash as a result of a stack overflow (infinite recursion).

Note: When this function is first called, the head of your linked list will be passed as *both* arguments to the function, like so:

```
doTheThing(head, head);
```

Hint: Notice that all the recursive calls always pass *head* as the first parameter. So, within this function, *head* will always refer to the actual head of the linked list. The second parameter is the only one that ever changes.

3) (5 pts) ALG (Stacks and Queues)

Consider the following function:

```
void doTheThing(void)
{
  int i, n = 9;  // Note: There are 9 elements in the following array.
  int array[] = {3, 18, 58, 23, 12, 31, 19, 26, 3};

  Stack *s1 = createStack();
  Stack *s2 = createStack();
  Queue *q = createQueue();

  for (i = 0; i < n; i++)
    push(s1, array[i]);

  while (!isEmptyStack(s1))
  {
     while (!isEmptyStack(s1))
       enqueue(q, pop(s1));   // pop element from s1 and enqueue it in q
     while (!isEmptyQueue(q))
       push(s2, dequeue(q));  // dequeue from q and push onto s2

     printf("%d ", pop(s2));  // pop from s2 and print element

     while (!isEmptyStack(s2))
       push(s1, pop(s2));     // pop from s2 and push onto s1
  }
  printf("Tada!\n");

  freeStack(s1);
  freeStack(s2);
  freeQueue(q);
}
```

What will be the exact output of the function above? (You may assume the existence of all functions written in the code, such as *createStack()*, *createQueue()*, *push()*, *pop()*, and so on.)

Computer Science Foundation Exam

January 12, 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	Passing	Score
1	5	DSN	3	
2	10	ALG	7	
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 *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) (5 pts) DSN (Binary Trees)

Write a **recursive** function to print a postorder traversal of all the integers in a binary tree. The node struct and function signature are as follows:

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

void print_postorder(node *root)
{
```

2) (10 pts) ALG (Minheaps)

a) Show the result of inserting the value 24 into the following minheap.

```
        33
      /    \
    41      89
   /  \    /
 77   50  92
```

b) Show the result of deleting the root of the following minheap.

```
        33
      /    \
    41      89
   /  \    /
 77   50  92
```

c) Using big-oh notation, what is the **worst-case** runtime for deleting the minimum element from a minheap that has *n* nodes?

3) (10 pts) ALG (AVL Trees)

a) Show the result of inserting 37 into the following AVL tree:

```
          84
        /    \
      25      106
    /    \      \
  12      39     212
        /
      30
```

b) Using big-oh notation, give the **best-case** runtime for inserting a new element into an AVL tree with *n* nodes:

c) Using big-oh notation, give the **worst-case** runtime for inserting a new element into an AVL tree with *n* nodes:

d) Using big-oh notation, give the **best-case** runtime for inserting a new element into a binary search tree with *n* nodes:

e) Using big-oh notation, give the **worst-case** runtime for inserting a new element into a binary search tree with *n* nodes:

Computer Science Foundation Exam

January 12, 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	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 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) ANL (Algorithm Analysis)

With proof, determine the Big-Oh run time of the function, f, below, in terms of the input parameter n. (Note: You may use results from algorithms studied previously in COP 3502 without restating the full proof of run time.)

```
#include <math.h>

int f(int array[], int n) {
    return frec(array, 0, n-1);
}

int frec(int array[], int lo, int hi) {

    if (lo == hi) return array[lo];

    int left = frec(array, lo, (lo+hi)/2);
    int right = frec(array, (lo+hi)/2+1, hi);

    int i, lCnt = 0, rCnt = 0;
    for (i=lo; i<=hi; i++) {
        if (abs(array[i]-left) < abs(array[i]-right))
            lCnt++;
        else
            rCnt++;
    }
    if (lCnt > rCnt) return lCnt;
    return rCnt;
}
```

2) (5 pts) ANL (Algorithm Analysis)

An algorithm processing a two dimensional array with R rows and C columns runs in $O(RC^2)$ time. For an array with 100 rows and 200 columns, the algorithm processes the array in 120 ms. How long would it be expected for the algorithm to take when processing an array with 200 rows and 500 columns? Please express your answer in ***seconds.***

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

Determine the following summation in terms of n (assume n is a positive integer 2 or greater), expressing your answer in the form $an^3 + bn^2 + cn$, where a, b and c are rational numbers. (Hint: Try rewriting the summation into an equivalent form that generates less algebra when solving.)

$$\sum_{i=n^2-3}^{n^2+n-4} (i+4)$$

Computer Science Foundation Exam

January 12, 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	Passing	Score
1	5	DSN	3	
2	10	ALG	7	
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 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.

1) (5 pts) DSN (Recursive Coding)

Mathematically, given a function f, we recursively define $f^k(n)$ as follows: if k = 1, $f^1(n) = f(n)$. Otherwise, for k > 1, $f^k(n) = f(f^{k-1}(n))$. Assume that a function, f, which takes in a single integer and returns an integer already exists. Write a ***recursive*** function fcomp, which takes in both n and k (k > 0), and returns $f^k(n)$.

```
int f(int n);

int fcomp(int n, int k) {















}
```

2) (10 pts) ALG (Sorting)

(a) (5 pts) Consider using Merge Sort to sort the array shown below. What would the state of the array be right before the ***last*** call to the Merge function occurs?

index	0	1	2	3	4	5	6	7	8	9
value	20	15	98	45	13	83	66	51	88	32

Answer:

index	0	1	2	3	4	5	6	7	8	9
value										

(b) (5 pts) An inversion in an array, *arr*, is a distinct pair of values *i* and *j*, such that *i* < *j* and *arr[i] > arr[j]*. The function below is attempting to count the number of inversions in its input array, *arr*, of size *n*. Unfortunately, there is a bug in the program. Given that the array passed to the function has all distinct values, what will the function always return (no matter the order of values in the input array), in terms of n? Also, suggest a quick fix so that the function runs properly. (Note: analyzing inversions is important to studying sorting algorithm run times.)

```
int countInversions(int arr[], int n) {    // line 1
    int i, j, res = 0;                     // line 2
    for (i = 0; i < n; i++) {              // line 3
        for (j = 0; j < n; j++) {          // line 4
            if (arr[i] > arr[j])           // line 5
                res++;                     // line 6
        }                                  // line 7
    }                                      // line 8
    return res;                            // line 9
}                                          // line 10
```

Return value of the function in terms of n: ________________________

Line number to change to fix the function: _____________

Line of code to replace that line: ____

3) (10 pts) DSN (Bitwise Operators)

In this problem we will consider buying a collection of 20 figurines, labeled 0 through 19, inclusive. The figurines come in packages. Each package has some non-empty subset of figurines. We can represent the contents of a single package using an integer in between 1 and $2^{20} - 1$, inclusive, where the bits that are on represent which figurines are in the package. For example, the integer $22 = 2^4 + 2^2 + 2^1$, would represent a package with figurines 1, 2 and 4. Each month, one package comes out. You greedily buy every package until you have all 20 figurines. Write a function that takes in an array of integers, *packages*, and its length, *n*, where *packages[i]* stores an integer representing the contents of the package on sale during month i, and returns the number of months you will have to buy packages to complete the set. It is guaranteed that each figurine belongs to at least one of the packages and that each value in the array packages is in between 1 and 2^{20}-1, inclusive. **<u>For full credit, you must use bitwise operators.</u>**

```
int monthsTillComplete(int packages[], int n) {

}
```