Computer Science Foundation Exam

December 16, 2016

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	5	ALG	3	
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 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)

Consider the following struct, which contains a string and its length in one nice, neat package:

```
typedef struct smart_string {
  char *word;
  int length;
} smart_string;
```

Write a function that takes a string as its input, creates a new *smart_string* struct, and stores a **new copy of that string** in the *word* field of the struct and the length of that string in the *length* member of the struct. The function should then return a pointer to that new *smart_string* struct. Use dynamic memory management as necessary. The function signature is:

```
smart_string *create_smart_string(char *str) {










}
```

Now write a function that takes a *smart_string* pointer (which might be NULL) as its only argument, frees all dynamically allocated memory associated with that struct, and returns NULL when it's finished.

```
smart_string *erase_smart_string(smart_string *s) {










}
```

2) (5 pts) DSN (Linked Lists)

Consider the following function, which takes the head of a linked list as its only input parameter:

```
node *funky(node *head) {
  if (head == NULL)
    return head;
  if (head->next != NULL && (head->next->data % 2) == 0) {
    head->next = yknuf(head->next->next, head->next);
    head = funky(head->next->next);
  }
  else if (head->next != NULL)
    head->next = funky(head->next);
  return head;
}

node *yknuf(node *n1, node *n2) {
  n2->next = n1->next->next;
  n1->next = n2;
  return n1;
}
```

Suppose someone passes the head of the following linked list to the *funky()* function:

```
 +----+   +----+   +----+   +----+   +----+   +----+
 | 31 |-->| 27 |-->| 84 |-->| 50 |-->| 40 |-->| 32 |-->NULL
 +----+   +----+   +----+   +----+   +----+   +----+
```
^ head

The function call is: `funky(head);`

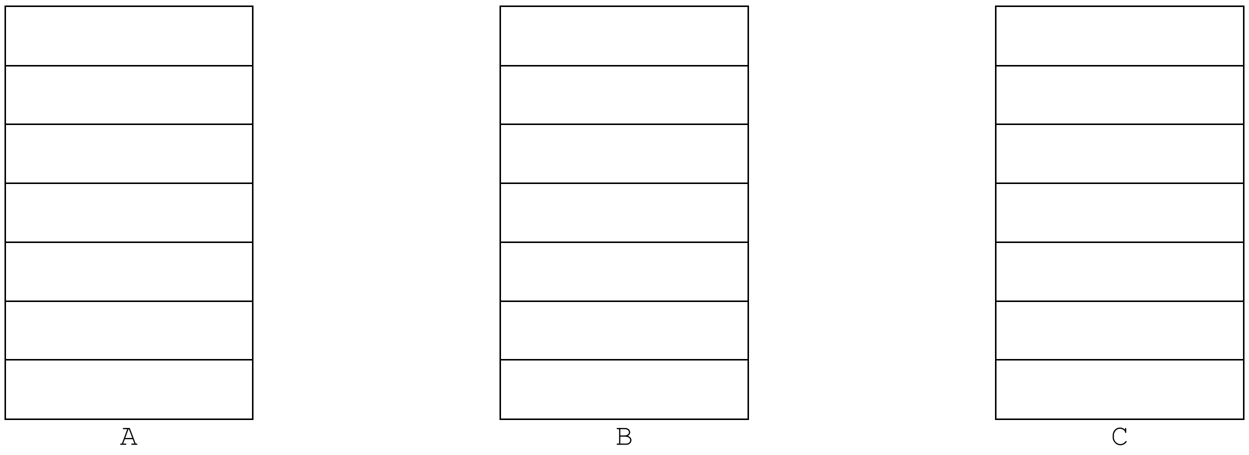
This program is going to crash spectacularly, but before it does, it will change the structure of the linked list a bit. Trace through the function call(s) and draw a new diagram that shows how the links in this linked list will be arranged at the moment when the program crashes. (In particular, show where the next pointer for each node except the one storing 32 will point.)

```
 +----+   +----+   +----+   +----+   +----+   +----+
 | 31 |   | 27 |   | 84 |   | 50 |   | 40 |   | 32 |
 +----+   +----+   +----+   +----+   +----+   +----+
```
^ head

3) (10 pts) DSN (Stacks: Infix to Postfix Conversion)

Convert the following from an infix expression to a postfix expression. Show the state of the operator stack at each of the indicated points (A, B, and C):

```
14  +  18  *   9  -  3  +  ( 4   -  8 )  *  ( 9  -  6 )  /  2
            ^                   ^                      ^
            A                   B                      C
```



Give the final postfix expression here:

____

What is the final value of this postfix expression?

____

Computer Science Foundation Exam

December 16, 2016

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	10	DSN	7	
2	5	ALG	3	
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. 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)

A binary search tree is considered "lopsided" if the root's left subtree height and right subtree height differ by more than one (i.e., the left subtree is more than one level deeper or shallower than the right subtree). This is different from the definition of "balanced" that comes up in relation to AVL trees, because the "lopsided" property only applies to the root of the tree – not every single node in the tree.

Write a function, *isLopsided()*, that takes the root of a binary search tree and returns 1 if the tree is lopsided, and 0 otherwise. You may write helper functions as you see fit. The node struct and function signature are as follows:

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

int isLopsided(node *root) {
```

```
}
```

2) (5 pts) ALG (Advanced Data Structures: Binary Heaps)

(a) (2 pts) Is the following tree a valid minheap? If so, give an array representation of this minheap. If not, explain why it's not a minheap.

```
         3
       /   \
      93   44
     /  \
   207  99
```

(b) (3 pts) Insert the value 3 into the following minheap. Clearly show each step of the process

```
         2
       /   \
      28   48
     / \   / \
   48  68 78 99
```

3) (10 pts) DSN (Advanced Tree Structures: Tries)

Write a recursive function that takes the root of a trie and counts how many odd-lengthed strings there are in the trie. For example, if the trie contains the empty string (“”), “bananas”, “avocados”, and “randomness”, the function should return 1, because only one of those strings has a length that is odd (“bananas”).

We will make our initial call to your function like so: `countOddStrings(root, 0);`

Part of the fun in this problem is figuring out what to do with that second parameter.

Please do **NOT** write any helper functions. Restrict yourself to the function whose signature is given below.

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

int countOddStrings(TrieNode *root, int k) {









}
```

Computer Science Foundation Exam

December 16, 2016

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) ALG (Algorithm Analysis)

Consider the following function:

```
int* makeArray(int n) {
    int* array = calloc(n, sizeof(int));
    int i, j;
    for (i=0; i<n; i++)
        for (j=i; j<n; j = j+i+1)
            array[j]++;
    return array;
}
```

(a) (1 pt) Assuming that the function is called with a value of n = 12 or greater, what will array[11] store when the array is returned from the function?

______

(b) (3 pts) In general, what will array[k] store when the function completes, assuming the function was called with an input value of k+1 or greater?

(c) (2 pts) Write a summation that provides a tight upper bound on the number of times the line of code array[j]++ runs when the function is called with the input value n.

(d) (4 pts) Utilizing the fact that $\sum_{i=1}^{n} \frac{1}{i} = O(lgn)$, determine the run time of the function makeArray for an input of size n. (Note: This run time is equal to the summation from part c.)

2) (5 pts) ANL (Algorithm Analysis)

An image processing algorithm takes O(n^3) time to run to filter an n x n pixel picture. If it takes 8 seconds to process a 1024 x 1024 pixel picture, how long will it take to process a 1536 x 1536 pixel picture?

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

(a) (5 pts) Determine the following sum in terms of n: $\sum_{i=1}^{2n-1}(3i-2)$.

(b) (5 pts) Let $T(n) = 3T\left(\frac{n}{2}\right) + n^2$. In using the iteration technique (3 steps) to solve the recurrence, we arrive at an equation of the form: $T(n) = AT\left(\frac{n}{8}\right) + Bn^2$. Find A and B.

Computer Science Foundation Exam

December 16, 2016

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	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.

1) (10 pts) DSN (Recursive Coding)

A derangement is a permutation of the integers 1, 2, 3, ..., n such that for all i, $1 \leq i \leq n$, the value in the i^{th} location isn't i. For example, (2, 1, 4, 3) is a derangement of 4 items since the first item isn't 1, the second item isn't 2, the third item isn't 3 and the fourth item isn't 4. But (3, 1, 5, 4, 2) is NOT a derangement of 5 items since the 4th item on this list is 4. Complete the code below so that it prints out all derangements of size n ($2 \leq n \leq 10$), where n is entered by the user.

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

void printD(int n);
void printDRec(int n, int* perm, int* used, int k);
void print(int* perm, int length);

int main() {
    int n;
    printf("Enter the size of your derangement(2-10).\n");
    scanf("%d", &n);
    printD(n);
    return 0;
}
void printD(int n) {
    int perm[MAX];
    int used[MAX];
    int i;
    for (i=0; i<MAX; i++) used[i] = 0;
    printDRec(___ , __________ , __________ , ______ );
}

void printDRec(int n, int* perm, int* used, int k) {
    if (k == n) {
        print(perm, n);
        return;
    }

    int i;
    for (i=0; i<n; i++) {
        if ( ______________________________________________ ) {

            perm[ ___ ] =  ____ ;

            used[ _____ ] =  _____ ;
            printDRec(n, perm, used, k+1);

            used[ ____ ] = ____ ;
        }
    }
}

void print(int* perm, int length) {
    int i;
    for (i=0; i<length; i++)
        printf("%d ", perm[i]+1);
    printf("\n");
}
```

2) (10 pts) ALG (Sorting)

(a) (6 pts) In quick sort, when running the partition function, the first step is to choose a random partition element. In some implementations, instead of just choosing a random element, 3 or 5 random elements are chosen and the median of those elements is then selected as the partition element, as opposed to making the partition element a single randomly selected item. What is the potential benefit of using this strategy (median of 3 or median of 5) versus the default strategy of just choosing a single random element?

(b) (4 pts) The best case run time of an insertion sort of n elements is O(n) and the worst case run time of an insertion sort is O(n^2). Describe how to (a) construct a list of n distinct integers that, when sorted by insertion sort, gets sorted in the best case run time, and (b) construct a list of n distinct integers that, when sorted by insertion sort, gets sorted in the worst case run time.

3) (5 pts) ALG (Bitwise Operators)

What is the output of the following C program?

```
#include <stdio.h>

int main() {

    int x = 13, y = 27, z = 74;
    printf("x^y = %d\n", x^y);
    printf("x&z = %d\n", x&z);
    printf("x&(y|z) = %d\n", x&(y|z));
    printf("x|y|z = %d\n", x|y|z);

    int i, sum = 0;
    for (i=0; i<10; i++) {
        if ((x & (1<<i)) != 0) sum++;
        if ((y & (1<<i)) != 0) sum++;
        if ((z & (1<<i)) != 0) sum++;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

```
x^y = _____

x&z = _____

x&(y|z) = _____

x|y|z = _____

sum = _____
```