# Computer Science Foundation Exam

**August 26, 2017**

**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 | 5 | DSN | 3 | |
| 2 | 10 | DSN | 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 <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 (Dynamic Memory Management in C)

There is something terribly wrong with the code given below: it has two memory leaks. After carefully inspecting the code, answer the questions below.

```
1:   int main(void)
2:   {
3:     char *str1 = malloc(sizeof(char) * 16);
4:     char *str2 = malloc(sizeof(char) * 16);
5:
6:     str1[0] = 'p';
7:     str1[1] = 'a';
8:     str1[2] = 's';
9:     str1[3] = 's';
10:    str1[4] = ',';
11:    str1[5] = '\0';
12:
13:    printf("%s ", str1);
14:    str2 = str1;
15:    printf("%s ", str2);
16:    str2 = NULL;
17:    strcpy(str1, "pass the exam!");
18:    printf("%s\n", str1);
19:
20:    free(str1);
21:    free(str2);
22:
23:    return 0;
24:  }
```

(a) (3 pts) Draw a picture that indicates the relevant state of memory ***after*** line 14 has completed. (Draw a rectangular box to indicate dynamically allocated memory.)

(b) (1 pt) Explain why line 14 causes a memory leak.

(c) (1 pt) Why is it possible for the code to crash on line 21?

**2)** (10 pts) DSN (Linked Lists)

Write a **recursive** function that takes in the head of a linked list and frees all dynamically allocated memory associated with that list. You may assume that **all** the nodes in any linked list passed to your function (including the head node) have been dynamically allocated. It's possible that your function might receive an empty linked list (i.e., a NULL pointer), and you should handle that case appropriately.

Note that your function must be recursive in order to be eligible for credit.

The linked list node struct and the function signature are as follows:

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

void destroy_list(node *head) {











}
```

**3)** (10 pts) ALG (Stacks) Suppose we pass the string "cupcake" to the following function. What will the function's output be, and what will the stacks *s1* and *s2* look like when the function terminates? You may assume the stack functions are written correctly and that the stacks are designed for holding characters.

```
void string_shenanigans(char *str)
{
   int i, len = strlen(str);
   char *new_string = malloc(sizeof(char) * (len + 1));
   Stack s1, s2;
   init(&s1); // initializes stack s1 to be empty
   init(&s2); // initializes stack s2 to be empty

   for (i = 0; i < len; i++) {
      push(&s1, str[i]);  // this pushes onto stack s1
      push(&s2, str[i]);  // this pushes onto stack s2
   }

   for (i = 0; i < len; i++) {
      if (i % 2 == 0) {
         // Note: pop() returns the character being removed from the stack.
         if (!isEmpty(&s1))
            new_string[i] = pop(&s1);
         if (!isEmpty(&s1))
            push(&s2, pop(&s1));
      }
      else {
         pop(&s2);
         new_string[i] = pop(&s2);
      }
   }

   new_string[len] = '\0';
   printf("%s\n", new_string);
   free(new_string);
}
```

| | | |
|---|---|---|
| | | |
| *printf()* output | final contents of *s1* (please label 'top' for clarity) | final contents of *s2* (please label 'top' for clarity) |

# Computer Science Foundation Exam

**August 26, 2017**

**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 | ALG | 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) ALG (Binary Search Trees)

(a) (6 pts) Following are three traversals produced by the exact same **binary search tree**. Using your powers of inference, determine which one is which. (Fill in each blank with "in-order," "pre-order," "post-order.") ***(Note: All sets of answers which don't contain each traversal exactly once will automatically be awarded 0 points.)***

____________________ traversal: 18 14 12 9 31 24 19 22 23 36

____________________ traversal: 9 12 14 23 22 19 24 36 31 18

____________________ traversal: 9 12 14 18 19 22 23 24 31 36

(b) (1 pt) What value must be at the root of the BST that produced the traversals listed in part (a)?

(c) (1 pt) Using big-oh notation, what is the best-case runtime for searching for a particular value in a BST with *n* nodes?

(d) (1 pt) Using big-oh notation, what is the worst-case runtime for searching for a particular value in a binary tree (a regular old binary tree, not necessarily a BST) with *n* nodes?

(e) (1 pt) Using big-oh notation, what is the worst-case runtime for searching for a particular value in a BST with *n* nodes?

**2)** (5 pts) ALG (Hash Tables)

Use the following hash function to insert the given elements into the hash table below. Use **quadratic probing** to resolve any collisions. You may assume that the correct table size (in this case, 10) is always passed to the function with the key that is being hashed.

```
int hash(int key, int table_size)
{
     int a = (key % 100) / 10;
     int b = key % 10;

     return (a + b) % table_size;
}
```

**Keys to insert (one by one, in the following order):** 2555, 1523, 5893, 800, 956

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**3)** (10 pts) DSN (Tries)

Write a recursive function that takes the root of a trie, *root*, and a single character, *alpha*, and returns the number of strings in the trie that contain that letter. You may assume that letter is a valid lowercase alphabetic character ('a' through 'z').

Note that we are *not* simply counting how many times a particular letter is represented in the trie. For example, if the trie contains only the strings "apple," "avocado," and "persimmon," then the following function calls should return the values indicated:

```
countStringsWithLetter(root, 'p') = 2
countStringsWithLetter(root, 'm') = 1
```

The TrieNode struct and function signature are given below. You may assume that the variable numwords accurately stores the number of valid words stored (# of nodes within the trie with flag set to 1) in all trie structs.

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

int countStringsWithLetter(TrieNode *root, char alpha) {









}
```

# Computer Science Foundation Exam

**August 26, 2017**

**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)

Consider the problem of taking n sorted lists of n integers each, and combining those lists into a single sorted list. For ease of analysis, assume that n is a perfect power of 2. Here are two potential algorithms to solve the problem:

Algorithm A: Run the Merge Algorithm, defined between two lists, on lists 1 and 2 to create a single merged list. Then rerun the algorithm on this merged list and list 3, creating a merged list of all items from lists 1, 2 and 3. Continue in this fashion, running the Merge Algorithm n-1 times, always between the currently "growing" list and the next list to be merged into it, until list n is merged in, creating a single sorted list.

Algorithm B: Pair up the lists into $\frac{n}{2}$ pairs of lists of size n. Run the Merge Algorithm on each of these pairs. Once this phase finishes, there will be $\frac{n}{2}$ lists with 2n integers. With the new lists, repeat the process until we are left with a single sorted list.

With sufficient work and proof, determine the Big-Oh run time, in terms of n, of both of these algorithms. Clearly put a box around your final answer for both algorithms.

**2)** (5 pts) ANL (Algorithm Analysis)

An algorithm processing an array of size n runs in $O(n\sqrt{n})$ time. For an array of size 10,000 the algorithm processes the array in 16 ms. How long would it be expected for the algorithm to take when processing an array of size 160,000? Please express your answer in ***seconds,*** writing out exactly three digits past the decimal.

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

Let *a*, *b*, *c*, and *d*, be positive integer constants with *a* < *b*. ***Without using the arithmetic sum formula***, prove that

$$\sum_{i=a}^{b}(ci + d) = \frac{(c(a + b) + 2d)(b - a + 1)}{2}$$

# Computer Science Foundation Exam

**August 26, 2017**

**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 | 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.**

**1)** (10 pts) DSN (Recursive Coding)

Define the weighted sum of an integer array a[0], a[1], ..., a[n-1] to be $\sum_{i=1}^{n}(ia[i-1])$. For example, the weighted sum of the array [7, 5, 8] would be $1 \times 7 + 2 \times 5 + 3 \times 8 = 41$. Write a ***recursive*** function that takes in an array `numbers` and its length `n`, and returns its weighted sum. You may assume that there will be no issues with integer overflow.

```
int weightedSum(int numbers[], int n) {









}
```

**2)** (5 pts) ALG (Sorting)

Show the contents of the following array after each iteration of Insertion Sort. The result after the last iteration has been included. (Note: due to the nature of this question, relatively little partial credit will be awarded for incorrect answers.)

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Initial | 17 | 22 | 16 | 5 | 18 | 14 | 2 |
| 1st iter | | | | | | | |
| 2nd iter | | | | | | | |
| 3rd iter | | | | | | | |
| 4th iter | | | | | | | |
| 5th iter | | | | | | | |
| 6th iter | 2 | 5 | 14 | 16 | 17 | 18 | 22 |

**3) (**10 pts) DSN (Bitwise operators)

Two useful utility functions when dealing with integers in their binary representation are

(a) `int lowestOneBit(int n)` - returns the value of the lowest bit set to 1 in the binary representation of n. (eg. `lowestOneBit(12)` returns 4, `lowestOneBit(80)` returns 16.)

(b) `int highestOneBit(int n)` - returns the value of the highest bit set to 1 in the binary representation of n. (eg. `highestOneBit(12)` returns 8, `highestOneBit(80)` returns 64.) **Note: You may assume that the input is less than $10^9$. The largest positive bit value in an integer is equal to $2^{30} > 10^9$.**

The pre-condition for the first function is that n must be a positive integer. The pre-condition for the second function is that n must be a positive integer less than $10^9$. Write both of these functions in the space below. To earn full credit, you ***must*** use bitwise operators when appropriate. (Namely, there are ways to solve this question without using bitwise operators, but these solutions will NOT receive full credit.)

```
int lowestOneBit(int n) {













}
```

```
int highestOnebit(int n) {













}
```