# Computer Science Foundation Exam

**January 14, 2017**

**Section I A**

**DATA STRUCTURES**

**SOLUTION**

**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 | 7 | |
| 3 | 10 | DSN | 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)

A catalogue of ***apps*** and their price is stored in a text file. Each line of the file contains the name of an app (1-19 letters) followed by its price with a space in between. Write a function called ***makeAppArray*** that reads the *app information* from the file and stores it in an array of app pointers. Your function should take 2 parameters: a pointer to the file containing the app information and an integer indicating the number of ***apps*** in the file. It should return a pointer to the array of ***apps***. An ***app*** is stored in a struct as follows:

```
typedef struct{
  char name[20];
  float price;
} app;
```

Make sure to allocate memory dynamically. The function signature is:

```
app** makeAppArray(FILE* fp, int numApps) {

    app** appArray = (app**)malloc(numApps * sizeof(app*)); //3 pts
    int i;
    for(i=0; i < numApps; i++){
        appArray[i] = (app*)malloc(sizeof(app));   // 2 pts
        fscanf(fp, "%s", appArray[i]->name);  // 2 pts
        fscanf(fp, "%f", &(appArray[i]->price)); // 2 pts
     }
     return appArray; // 1 pt
}
```

**Grading notes: the casts aren't necessary, no points off for forgetting to declare i, no points off if only one percent code is wrong, 1 pt off if both percent codes are wrong, take only 1 pt off total if the syntax for reading from a file is incorrect, take off only 1 pt total if they use a dot instead of an array, they can order these statements differently - they can allocate ALL of the space before reading anything in.**

**2)** (5 pts) ALG (Linked Lists)

Consider the following function that takes in as a parameter a pointer to the front of a linked list(`list`) and the number of items in the list(`size`). ***node*** is defined as follows:

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

int mystery(node* list, int size) {
    node* prev = list;
    node* temp = list->next;

    while (temp != NULL) {
        if (list->data == temp->data) {
            prev->next = temp->next;
            free(temp);
            size--;
            temp = prev->next;
        }
        else {
            prev = prev->next;
            temp = temp->next;
        }
    }
    return size;
}
```

If **`mystery(head, 7)`**, is called, where head is shown below, what will the function return and draw a picture of the resulting list, right after the call completes?

```
 +-----+   +-----+   +-----+   +-----+   +-----+   +-----+   +-----+
 | 26  |-->| 39  |-->| 26  |-->| 20  |-->| 26  |-->| 32  |-->| 39  |-->NULL
 +-----+   +-----+   +-----+   +-----+   +-----+   +-----+   +-----+
  ^ head
```

```
Adjusted List
-------------
```

```
        +-----+   +-----+   +-----+   +-----+   +-----+
        | 26  |-->| 39  |-->| 20  |-->| 32  |-->| 39  |-->NULL
        +-----+   +-----+   +-----+   +-----+   +-----+
         ^ head
```

The function returns 5.

**Grading: 2 pts return value (all or nothing), 3 pts list, give partial for list as you see fit.**

**3)** (10 pts) DSN (Queues)

A queue is implemented as an array. The queue has the 2 attributes, ***front*** and ***size***. ***front*** is the index in the array where the next element to be removed from the queue can be found, if the queue is non-empty. (If the queue is empty, front may be any valid array index from 0 to 19.) ***size*** is the total number of elements currently in the queue. For efficient use of resources, elements can be added to the queue not just at the end of the array but also in the indices at the beginning of the array before front. Such a queue is called a circular queue. A circular queue has the following structure:

```
typedef struct {
    int values[20];
    int front, size;
} cQueue;
```

Write an enqueue function for this queue. If the queue is already full, 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 the array size is hard-coded to be 20 in the struct above, you may use this value in your code and assume that is the size of the array values inside the struct.

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

    if (thisQ->size == 20)          // 2 pts
        return 0;                   // 1 pt

    // 5 pts: 1 values, 3 pts index, 1 pt item
    thisQ->values[(thisQ->front+thisQ->size)%20] = item;

    thisQ->size++;                  // 1 pt
    return 1;                       // 1 pt
}
```

**Grading notes: If they forget thisQ-> each time, take off 2 pts total.**

# Computer Science Foundation Exam

**January 14, 2017**

**Section I B**

**DATA STRUCTURES**

**SOLUTION**

**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 | | 7 | |
| 2 | 5 | | 3 | |
| 3 | 10 | | 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)

Michael took CS 1 last semester. During the Winter break he thought that it would be cool to keep track of all of the new words that he learned while reading a novel. He has stored all of his words (all 1-19 lowercase letters only) in alphabetic order in a binary search tree (BST). The nodes of his BST are stored in the following structure:

```
typedef struct bstNode {
   struct bstNode *left, *right;
   char word[20];
} bstNode;
```

Michael wants to count the number of words in his binary search tree that come before a specified word in alphabetical order. Write a **recursive** function `countBefore` which takes in a pointer to the root of a binary search tree storing the words and a string `target` (of 1-19 lowercase letters only) and returns the number of words in the tree that ***come before*** `target`, alphabetically.

```
int countBefore(bstNode* root, char target[]){

    if (root == NULL) return 0;                 // 2 pts

    if (strcmp(target, root->word) <= 0)        // 2 pts
        return countBefore(root->left);         // 2 pts

    // 1 pt return, 1 pt 1, 1 pt left, 1 pt right
    return 1 + countBefore(root->left) + countBefore(root->right);
}
```

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

(a) (3 pts) A set of students' names are stored in a hash table implemented as an array of size 25. Their grades out of 100 are used as input to the hashing function. Suggest one hash function that can be used to store the names. Would your function cause collisions? Explain your answer.

Hash function stores the student's name in the array index score%25.
Hashmap[score%25] = $\text{name}_{\text{score}}$ **(Grading: 2 pts, note, MANY answers are valid here!!!)**

This function can cause collisions since 2 distinct scores can has to the same index or because two different students can earn the exact same score! **(Grading: 1 pt)**

(b) (2 pts) If the following students have the grades shown, and your hash function given in (a) is used, draw the state of the hash map after these 3 entries are inserted into the table. (Note: No need to show all 25 array slots, just clearly label the index and contents of each of the non-empty array slots.)

**Mary 60**
**Ben 75**
**Dona 13**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Ben | ........... | ........... | Mary | | | Donna | ........... | | |
| 0 | | | 10 | 11 | 12 | 13 | | 23 | 24 |

**Grading: 2 pts for correct response (BASED ON THEIR HASH FUNCTION), 1 pt if some of their answer is correct**

**3)** (10 pts) DSN (AVL Trees)

(a) (8 pts) Create an AVL tree by inserting the following values in the order given: 38, 72, 58, 16, 3, 24, 8, and 15. Show the state of the tree after each insertion.

Note: it the solution below, rotations aren't shown, just the final answers after the appropriate rotations. Steps where rotations were necessary are marked with an astericks at the root of the rotation.

```
38        38         58*          58           58            38*
           \         /  \        /  \         /  \          /   \
           72      38    72    38    72    16*    72      16     58
                               /           / \           / \      \
                             16           3   38        3  24      72
```

```
      38                    38
     /  \                  /  \
   16    58              16    58
   / \     \             / \     \
  3   24    72         8*   24    72
   \                   / \
    8                 3   15
```

**Grading: 1 pt for each tree, as long as the insertion on step k was of equal difficulty to the correct insertion, give the point as long as the insertion is correct based on their answer for step k-1.**

(b) (2 pts) Draw the state of the tree after the deletion of the node containing the value 16.

There are two possible answers here. One may replace the 16 with either the 15 or 24 and then delete the physical node where the 15 or 24 was stored, respectively. The answer on the left is what occurs when we replace 16 with 15 and the answer on the right is what occurs when we replace 16 with 24:

# Computer Science Foundation Exam

**January 14, 2017**

**Section II A**

**ALGORITHMS AND ANALYSIS TOOLS**

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

**SOLUTION**

| Question # | Max Pts | Category | Passing | Score |
|---|---|---|---|---|
| 1 | 5 | ANL | 3 | |
| 2 | 10 | ANL | 7 | |
| 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)** (5 pts) ALG (Algorithm Analysis)

Consider the following function with integer inputs n and m:

```
void solveit(int* array, int n, int m) {

    int i, res = 0;
    for (i=0; i<n; i++) {
        int low = 0, high = m;
        while (low < high) {
            int mid = (low+high)/2;
            if (f(mid) < array[i])
                low = mid+1;
            else
                high = mid;
        }
        printf("%d\n", low);
    }
}
```

You may assume that the function f that is called from solveit defines a monotonically increasing function that runs in O(1) time. With proof, determine the run-time of this function in terms of n and m.

The outer loop runs n times. The inner loop runs a binary search between 0 and m, subdividing that interval by 2 with each loop iteration. The most number of times this loop could run is roughly lg m. (A more formal proof simply let's k equal the number of times the loop runs and solves for k in the equation $\frac{m}{2^k} = 1$. Multiply both sides to get $m = 2^k$. By definition of log, $k = log_2 m$.) It follows that the run time is O(nlgm).

**Grading: 1 pt for outer loop, 3 pts for inner loop, 1 pt for multiplying. (Note: to award the full three points, student must either point out that code is a binary search or that repeated division by 2 is going on to the interval high-low. They don't need to set up the equation with k, the number of iterations the loop runs.)**

**2)** (10 pts) ANL (Algorithm Analysis)

(a) (5 pts) An algorithm to process an array of size n takes O(n$^2$) time. If the algorithm takes 113 ms to process an array of size 10,000 how long will it take to process an array of size 100,000, in seconds?

Let the algorithm with input size n take T(n) time where $T(n) = cn^2$. Using the given information, we have:

$$T(10000) = c10000^2 = 113ms$$
$$c = (113 \times 10^{-8}ms)$$

Now, let's plug in n = 100,000:

$$\begin{aligned} T(100000) &= (113 \times 10^{-8}ms) \times 100000^2 \\ &= 113 \times 10^2 ms \\ &= 11300ms \\ &= 11.3\ seconds \end{aligned}$$

**Grading: 2 pts for solving for c, 2 pts for solving for T(100,000), 1 pt for converting to seconds.**

11.3 seconds

(b) (5 pts) A search algorithm on an array of size n runs in O(lg n) time. If 200,000 searches on an array of size 2$^{18}$ takes 20 ms, how long will 540,000 searches take on an array of size 2$^{20}$ take, in milliseconds?

Let T(n) represent the time for a single search on an array of size n, where $T(n) = clog_2 n$. Using the given information, we have:

$$200000 \times T(2^{18}) = 200000 \times c \times log_2 2^{18} = 20ms$$
$$18 \times 200000c = 20ms$$
$$c = \frac{1}{180000}ms$$

Now, we would like to find 540000T(2$^{20}$):

$$\begin{aligned} 540000 \times T(2^{20}) &= 540000 \times \frac{1}{180000}ms \times log_2 2^{20} \\ &= 3 \times 20ms \\ &= 60ms \end{aligned}$$

**Grading: 2 pts for solving for c, 2 pts for solving for result, 1 pt for simplifying**

60 milliseconds

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

Find the Big-Oh solution to the following recurrence relation using the iteration technique. Please show all of your work, including 3 iterations, followed by guessing the general form of an iteration and completing the solution. Full credit will only be given if all of the work is accurate (and not just for arriving at the correct answer.)

$$T(n) = 4T\left(\frac{n}{2}\right) + n, T(1) = 1$$

First, iterate three times:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &= 4\left[4T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n \\ &= 16T\left(\frac{n}{4}\right) + 2n + n \\ &= 16T\left(\frac{n}{4}\right) + 3n \\ &= 16\left[4T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 3n \\ &= 64T\left(\frac{n}{8}\right) + 4n + 3n \\ &= 64T\left(\frac{n}{8}\right) + 7n \end{aligned}$$

In general, after k iterations, we will have $T(n) = 4^kT\left(\frac{n}{2^k}\right) + (2^k - 1)n$. We want to plug in the value of k for which $\frac{n}{2^k} = 1$,which is when $n = 2^k$. In this case, note that $n^2 = (2^k)^2 = 2^{2k} = 4^k$. Plugging in, we find:

$$\begin{aligned} T(n) &= n^2T(1) + (n-1)n \\ &= n^2 + n^2 - n \\ &= 2n^2 - n \\ &= O(n^2) \end{aligned}$$

**Grading: 2 pts for second iteration (16T(n/4)+3n), 2 pts for third iteration (64T(n/8+7n), 2 pts for general form, 2 pts for value to plug into general form, 2 pts for final solution.**

# Computer Science Foundation Exam

**January 14, 2017**

**Section II B**

**ALGORITHMS AND ANALYSIS TOOLS**

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

**SOLUTION**

| 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)

Write a recursive function that returns 1 if an array of size n is in sorted order from smallest to largest with all values less than or equal to max, and 0 otherwise. Note: If array a stores 3, 6, 7, 7, 12, then `isSorted(a, 12, 5)` should return 1 but `isSorted(a, 11, 5)` should return 0. If array b stores 3, 4, 9, 8, then `isSorted(b, 20, 4)` should return 0, since 9 is bigger than 8 but appears before it.

```
int isSorted(int* array, int max, int n) {
    if (n == 0) return 1;                          // 3 pts
    if (array[n-1] > max) return 0;                // 3 pts
    return isSorted(array, array[n-1], n-1);       // 4 pts
}
```

**Grading Notes: Lots of ways to solve this problem. Map their solution to the cases laid out here:**

**0/1 size case: 3 pts**
**Checking last element too big: 3 pts**
**Recursive call if last pair (or first pair) is valid: 4 pts**

**The points are mapped so that many items could be off by a tiny bit, so adjust the points as necessary. If there are two small errors in two different places each worth 2 pts, just take off 1 point for one of the two errors and let the other one go.**

Note: If max was not a parameter to the function and we wanted to simply write a recursive function that determined if the array specified by the parameters was sorted or not, the following function would suffice:

```
int isSortedAlt(int* array, int n) {
    if (n < 2) return 1;
    if (array[n-1] < array[n-2]) return 0;
    return isSortedAlt(array, n-1);
}
```

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

Consider running a Merge Sort on the array shown below. Show the state of the array **right before** the last Merge is performed. (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 | 7 |
|---|---|---|---|---|---|---|---|---|
| value | 13 | 8 | 9 | 2 | 1 | 17 | 6 | 5 |

Your answer:

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| value | 2 | 8 | 9 | 13 | 1 | 5 | 6 | 17 |

**Grading: 5 pts for correct answer, 2 pts for sorted pairs (8 13 2 9 1 17 5 6), 2 pts for (2 8 9 13 1 17 6 5), 2 pts for (1 2 8 9 13 17 6 5), 0 pts for all others**

**3) (**10 pts) ALG (Backtracking)

A D-digit divisible number is a positive integer of D digits (with no leading digits zero) such that each of its prefixes of k digits is a number divisible by k. For example, 52240 is a 5-digit divisble number because 5 is divisible by 1, 52 is divisible by 2, 522 is divisible by 3, 5224 is divisible by 4 and 52240 is divisible by 5. Assume that there exists a function as specified below:

```
int kDigitPrefixValue(char* number, int k);
```

such that if number is storing the string version of an integer that is at least k digits long, then the function will return the integer value of the first k digits represented in number. For example, `kDigitPrefixValue("52240", 4)` will return the integer 5224.

Complete the recursive function below so that it will print out all 6-digit divisible numbers. (A complete wrapper function is provided for you, so just fill out the blanks in the recursive function.)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int kDigitPrefixValue(char* number, int k);
void printkDivisibleRec(char* number, int k);
void wrapper(int numdigits);

int main() {
    wrapper(6);
    return 0;
}

void wrapper(int numdigits) {
    char* tmp = malloc(sizeof(char)*(numdigits+1));
    int i;
    for (i=0; i<numdigits; i++) tmp[i] = '0';
    tmp[numdigits] = '\0';
    printkDivisibleRec(tmp, 0);
    free(tmp);
}

void printkDivisibleRec(char* number, int k) {
    if (k == strlen(number)) {
        printf("%s\n", number);
        return;
    }
    int i = k == 0 ? 1 : 0;
    for (; i < 10 ; i++) {                    Grading: 1 pt per blank

        number[ k ] = (char)( i +'0');

        int prefix = kDigitPrefixValue(number, k+1 );

        if ( prefix %( k+1 ) == 0 )

            printDivisibleRec(number, k+1  );
    }
}
```