Computer Science Foundation Exam

January 16, 2021

Section I A

DATA STRUCTURES

SOLUTION

Directions: You may either directly edit this document, or write out your answers in a .txt file, or scan your answers to .pdf and submit them in the COT 3960 Webcourses for the Assignment "Section I A". Please put your *name, UCFID and NID* on the top left hand corner of each document you submit. Please aim to submit 1 document, but if it's necessary, you may submit 2. Clearly mark for which question your work is associated with. If you choose to edit this document, please remove this cover page from the file you submit and make sure your *name, UCFID and NID* are on the top left hand corner of the next page (first page of your submission).

Question #	Max Pts	Category	Score
1	5	DSN	
2	10	DSN	
3	10	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 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) DSN (Dynamic Memory Management in C)

Suppose we have a function that is designed to take in a large string and trim it down to only the needed size. The function is called trim_buffer. It takes in 1 parameter: the buffer, which is a string with a max size of 1024 characters. It returns a string that is only the size needed to represent the valid characters in the buffer. The function is implemented below.

Identify all of the errors (there are multiple errors) with the following trim_buffer function.

```
#define BUFFERSIZE 1024

// Pre-condition: buffer has a '\0' character at or before index
//                BUFFERSIZE-1.
// Post-condition: returns a pointer to a dynamically allocated
//                 string that is a copy of the contents of buffer,
//                 dynamically resized to the appropriate size.
char * trim_buffer(char * buffer) {
     char *string;
     int length;

     while (length < BUFFERSIZE && buffer[length] != '\0')
               length++;

     string = malloc(sizeof(char) * (length));

     length = 0;
     while ((string[length] = buffer[length]) != '\0')
               length++;

     return;
}
```

Errors to find:

Allocating length characters instead of length + 1 characters – **Grading: 3 points**
Length should be initialized before the first while loop – **Grading: 1 point**
Return value should be string instead of empty – **Grading: 1 point**

2) (10 pts) ALG (Linked Lists)

Suppose we have a singly linked list implemented with the structure below. Write a **recursive** function that takes in the list and returns 1 if the list is non-empty AND **all** of the numbers in the list are even, and returns 0 if the list is empty OR contains at least one odd integer. (For example, the function should return 0 for an empty list, 1 for a list that contains 2 only, and 0 for a list that contains 3 only.)

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

int check_all_even(struct node *head) {

     // Grading: 2 pts
     if (head == NULL)
        return 0;

     // Grading: 4 pts, we have to have this here to
     // differentiate between an empty and non-empty list.
     // 2 pts for checking next is NULL, 1 pt for each return.
     if (head->next == NULL) {
        if (head->data % 2 == 0)
            return 1;
        else
            return 0;
     }

     // Grading: 1 pt if, 1 pt return
     if (head->data % 2 != 0)
        return 0;

     // Grading: 2 pts
     return check_all_even(head->next);


}
```

3) (10 pts) ALG (Queues)

Consider the circular array implementation of a queue named Q, implemented with the structure shown below.

```
struct queue {
    int *array;
    int num_elements;
    int front;
    int capacity;
};
```

Suppose the queue is created with a capacity of 5 and front and num_elements are initialzed to 0. Trace the status of the queue by showing the valid elements in the queue and the position of front after each of the operations shown below. Indicate front by making bold the element at the front of the queue.

1. enqueue(Q, 50);
2. enqueue(Q, 34);
3. enqueue(Q, 91);
4. x = dequeue(Q);
5. enqueue(Q, 23);
6. y = dequeue(Q);
7. enqueue(Q, y);
8. enqueue(Q, 15);
9. enqueue(Q, x);
10. x = dequeue(Q);

After stmt #1:

front

50				

After stmt #2:

front

50	34			

After stmt #3:

front

50	34	91		

After stmt #4:

front

	34	91		

After stmt #5:

front

	34	91	23	

After stmt #6:

front

		91	23	

After stmt #7:

front

		91	23	34

After stmt #8:

front

15		**91**	23	34

After stmt #9:

front

15	50	**91**	23	34

After stmt #10:

front

15	50		**23**	34

Grading: 1 pt per array, must be perfectly correct to get the point.

Computer Science Foundation Exam

January 16, 2021

Section I B

DATA STRUCTURES

SOLUTION

Directions: You may either directly edit this document, or write out your answers in a .txt file, or scan your answers to .pdf and submit them in the COT 3960 Webcourses for the Assignment "Section I B". Please put your *name, UCFID and NID* on the top left hand corner of each document you submit. Please aim to submit 1 document, but if it's necessary, you may submit 2. Clearly mark for which question your work is associated with. If you choose to edit this document, please remove this cover page from the file you submit and make sure your *name, UCFID and NID* are on the top left hand corner of the next page (first page of your submission).

Question #	Max Pts	Category	Score
1	10	DSN	
2	5	ALG	
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 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.

Name: ______________________
UCFID: ______________________
NID: ________________________

1) (10 pts) DSN (Binary Trees)

Write a function named *find_below*() that takes a pointer to the root of a binary tree (*root*) and an integer value (*val*) and returns the greatest value in the tree that is strictly less than *val*. If no such value exists, simply return *val* itself. Note that the tree passed to your function will **not** necessarily be a binary **search** tree; it's just a regular binary tree.

For example:

```
        18            find_below(root, 196) would return 22
       /  \           find_below(root, 1) would return 1
      7    4          find_below(root, 4) would return 1
     / \              find_below(root, 22) would return 18
    1   22            find_below(root, 20) would return 18
         \            find_below(root, 8) would return 7
          8           find_below(root, -23) would return -23
```

You must write your solution in a **single** function. You cannot write any helper functions.

The function signature and node struct are given below.

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

int find_below(node *root, int val)
{
      int retval = val;
      int v1, v2;

      // Grading 2 pts for NULL case.
      if (root == NULL)
            return val;

      // Grading: 2 pts each, to make both recursive calls and store answers.
      v1 = find_below(root->left, val);
      v2 = find_below(root->right, val);

      // 4 pts total for this logic. 2 pts when answer is val, 1 pt each for v1, v2.
      if (root->data < val && (root->data > retval || retval == val)) retval = root->data;
      if (v1 < val && (v1 > retval || retval == val)) retval = v1;
      if (v2 < val && (v2 > retval || retval == val)) retval = v2;

      return retval;
}
// Note: Many ways to do the logic at the end.
```

2) (5 pts) ALG (Heaps)

Show the state of the following minheap after performing the *deleteMin* operation. (Instead of writing this with a pen or pencil, typing the result in text similarly to how the drawing of the heap below was constructed will suffice.)

```
       14
     /    \
   22      72
  /  \    /  \
38   40  80  90
```

Solution:

```
       22
     /    \
   38      72
  /  \    /
90   40  80
```

Grading:
1 pt for 22 being at the root
1 pt for 38 being left of root
1 pt for 90 being left of left of root
2 pts for 40, 72 and 80 being unchanged

3) (10 pts) DSN (Tries)

Write a function that takes the root of a trie (*root*) and returns the number of strings in that trie that **end** with the letter *q*. The *count* member of the node struct indicates how many occurrences of a particular string have been inserted into the trie. So, a string can be represented in the trie multiple times. If a string ending in *q* occurs multiple times in the trie, all occurrences of that string should be included in the value returned by this function.

The node struct and function signature are given below. You must write your solution in a **single** function. You cannot write any helper functions.

```
typedef struct TrieNode
{
   // Pointers to the child nodes (26 total).
   struct TrieNode *children[26];

   // Indicates how many occurrences of a particular string are contained
   // in this trie. If none, this is set to zero (0).
   int count;
} TrieNode;

int ends_with_q_count(TrieNode *root)
{

      int i;
      int retval = 0;

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

       // Grading: 1 pt loop to 26, 1 pt retval +=, 1 pt rec call
      for (i = 0; i < 26; i++)
            retval += ends_with_q_count(root->children[i]);

       // Grading: 2 pts NULL check, 2 pts retval +=
      // Note: 'q' - 'a' = 16. This can go before the for also.
      if (root->children['q' - 'a'] != NULL)
            retval += root->children['q' - 'a']->count;

       // Grading: 1 pt
      return retval;
}
```

Computer Science Foundation Exam

January 16, 2021

Section II A

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

Directions: You may either directly edit this document, or write out your answers in a .txt file, or scan your answers to .pdf and submit them in the COT 3960 Webcourses for the Assignment "Section II A". Please put your *name, UCFID and NID* on the top left hand corner of each document you submit. Please aim to submit 1 document, but if it's necessary, you may submit 2. Clearly mark for which question your work is associated with. If you choose to edit this document, please remove this cover page from the file you submit and make sure your *name, UCFID and NID* are on the top left hand corner of the next page (first page of your submission).

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 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.h, stdio.h, math.h, string.h) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

What is the run-time of the function `hash_func` shown below, in terms of *n*, the length of its input string? Please provide sufficient proof of your answer. (9 out of the 10 points are awarded for the proof. 1 point is awarded for the answer.)

```
#include <stdio.h>
#include <string.h>
#define MOD 1072373
#define BASE 256

int hash_func(char* str);
int hash_func_rec(char* str, int k);

int hash_func(char* str) {
    return hash_func_rec(str, strlen(str));
}

int hash_func_rec(char* str, int k) {
    if (k == -1) return 0;
    int sum = 0;
    for (int i=k-1; i>=0; i--)
        sum = (BASE*sum + str[i])%MOD;
    return (sum + hash_func_rec(str, k-1))%MOD;
}
```

The method to analyze the run time of a recursive function is to set up a recurrence relation equal to the run time of the function, and then solve that recurrence relation.

For this example, let T(k) be the run-time of hash_func_rec, where k is the second input parameter.

This function is a wrapper function for the function call hash_func_rec, which is initially called with an input parameter of k = n. We first see that if the input k == -1, the function immediately terminates. If k = 0, there will be a constant amount of work (for loop that runs once then the quick recursive call), thus, T(0) = 1. (Alternatively, one can write T(0) = c, for some constant c. For Big-Oh analysis, either will suffice.)

Otherwise, the for loop runs exactly k times. Then, the function makes a recursive call with the value k-1. It follows that the recurrence relation that governs this function is

T(k) = O(k) + T(k-1), T(0) = 1.

If we iterate the recurrence several times, it can be shown that the pattern is that

T(k) = $\sum_{i=1}^{k} O(i) \le \sum_{i=1}^{k} ci = \frac{ck(k+1)}{2} \in \boldsymbol{O}(k^2)$. Since we are asked to answer the question in terms of the variable n, where n is the length of the input string, the run-time of the algorithm is $\boldsymbol{O}(\boldsymbol{n}^2)$.

In general, a recurrence of the form T(n) = T(n-1) + f(n) with a constant value for a small input value of T will have the solution T(n) = $\sum_{i=1}^{n} f(i) + c$, for some constant c. (This can be shown by iterating down to a base case.)

Alternatively, one can note that if we "unroll" the recursion, the code effectively runs a nested set of loops where the first loop runs n times, the second loop runs n-1 times, etc., last loop runs once. From that observation, we obtain the same summation as the one shown above.

Grading: 2 pts for recognizing that the initial recursive call does O(n) work.
2 pts for recognizing that the effective input size to the recursive call is n-1, if the input size of the previous input was n.
2 pts for either setting up the recurrence relation or summation
4 pts for solving the recurrence relation or summation

If an answer of O(n^2) is given without any justification, award 1 pt as stated.)

2) (5 pts) ANL (Algorithm Analysis)

A sorting algorithm takes $O(n\sqrt{n})$ time to sort n values. The algorithm took .2 milliseconds to sort an array of 1000 values. How many seconds would it take to sort an array of size 900,000?

Let the run time of the algorithm be T(n). It follows that $T(n) = cn\sqrt{n}$, for some constant c. Use the given information to solve for c:

$T(1000) = c1000\sqrt{1000} = .2\ ms,$
$c = \frac{.2}{1000\times\sqrt{1000}}$

Now, let us find T(900000):

$$T(900000) = \frac{.2}{1000 \times \sqrt{1000}} \times (900000) \times \sqrt{900000} = 180\sqrt{900}ms = 180 \times 30ms = 5400ms$$

Converting 5400 ms to seconds, we get **5.4 seconds** as the final answer.

Grading: 1 pt to set up equation for c, 1 pt to solve for c, 1 pt to set up final equation, 1 pt to solve for the answer in milliseconds, 1 pt to cover to seconds.

3) (10 pts) ANL (Recurrence Relations)

What is the closed form solution to the following recurrence relation? Please use the iteration technique, show all of your work and provide your final answer in Big-Oh notation.

T(1) = 1
T(n) = 2T(n/4) + 1

Iterate the recurrence three times:

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

$T(n) = 2(2T\left(\frac{n}{16}\right) + 1) + 1$

$T(n) = 4T\left(\frac{n}{16}\right) + 3$ (two iterations)

$T(n) = 4(2T\left(\frac{n}{64}\right) + 1) + 3$

$T(n) = 8T\left(\frac{n}{64}\right) + 7$ (three iterations)

Now, let's make a guess as to the form of the recurrence after iterating k times based on the first three iterations:

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

Since we know T(1), we want to plug in the value of k for which $\frac{n}{4^k} = 1$, in for k. Solving, we find that $n = 4^k$. Taking the square root of both sides, we find $\sqrt{n} = \sqrt{4^k} = \sqrt{2^{2k}} = (2^{2k})^{\frac{1}{2}} = 2^k$. Substituting for both 4^k and 2^k, in the right hand of the recurrence, we get:

$$T(n) = \sqrt{n}T\left(\frac{4^k}{4^k}\right) + (\sqrt{n} - 1) = \sqrt{n}T(1) + (\sqrt{n} - 1) = \sqrt{n} + \sqrt{n} - 1 \in \boldsymbol{O}(\sqrt{\boldsymbol{n}})$$

Grading: 1 pt for first iteration
1 pt for second iteration
2 pts for third iteration
2 pts for general form guess
2 pts to plug in n = 4^k into general form (or equivalent)
2 pts to substitute and get to the final answer.

Computer Science Foundation Exam

January 16, 2021

Section II B

ALGORITHMS AND ANALYSIS TOOLS

ONLINE EXAM

Directions: You may either directly edit this document, or write out your answers in a .txt file, or scan your answers to .pdf and submit them in the COT 3960 Webcourses for the Assignment "Section II B". Please put your *name, UCFID and NID* on the top left hand corner of each document you submit. Please aim to submit 1 document, but if it's necessary, you may submit 2. Clearly mark for which question your work is associated with. If you choose to edit this document, please remove this cover page from the file you submit and make sure your *name, UCFID and NID* are on the top left hand corner of the next page (first page of your submission).

Question #	Max Pts	Category	Score
1	10	DSN	
2	5	ALG	
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 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.

Name: ______________________
UCFID: ______________________
NID: ________________________

1) (10 pts) DSN (Recursive Coding)

Imagine a Towers of Hanoi puzzle with 4 towers, labeled A, B, C and D, with a tower of n disks, starting on tower A, to be moved to tower B, using the usual rules of the puzzle. One strategy to solve the puzzle would be to move the k smallest disks recursively to tower D, where all 4 towers are used. Then, with the remaining n - k disks, use the usual strategy (since tower D is unavailable), which will take exactly $2^{n-k} - 1$ moves, to transfer the bottom n - k disks to tower B. Finally, now that you can use all 4 towers again, recursively transfer the k smallest disks on tower D to tower B, completing the puzzle. Sonia has decided that she wants the value of k to be set at (3n)/4, using integer division. For this question, write a recursive function that takes in n, the number of disks in the game, and returns the number of moves that it will take to solve the game using Sonia's strategy. A function prototype with pre and post conditions is provided below. (**Note: In order to get full credit you MUST NOT USE the pow function in math.h because it returns a double which has inherent floating point error. Please manually use integers to calculate an exponent or bitwise operators.)**

```
// Pre-condition: 1 <= n <= 115 (ensures no overflow)
// Post-condition: Returns the number of moves Sonia's strategy
//                 will take to solve a Towers of Hanoi with n
//                 disks with 4 towers.
int fourTowersNumMoves(int n) {

    // Grading: 2 pts
    if (n == 1) return 1;

    // Grading: 2 pts to calculate this split somewhere.
    int split = (3*n)/4;

    // Grading: 1 pt return, 1 pt 2*, 1 pt rec call
    // 3 pts calculation of (2 to the power n-split)-1.
    return 2*fourTowersNumMoves(split) + (1<<(n-split)) - 1;


}
```

2) (5 pts) ALG (Sorting)

The code below is a buggy implementation Selection Sort.

```
void buggySelectionSort(int array[], int n) {

    for (int i=n-1; i>=0; i--) {

        int best = array[0];
        for (int j=1; j<=i; j++) {
            if (array[j] > best)
                best = array[j];
        }

        array[i] = best;
    }
}
```

(a) Conceptually, the variable best is storing the wrong thing. What should it store instead?

Best is storing the largest number in the array upto index i, but best should really store **the index** where the largest value upto index i is being stored.

Grading: 2 pts, all or nothing.

(b) If we fix the code so that best stores what it ought to, conceptually, we will have to change both the if statement inside of the j for loop as well as the assignment statement inside of the if. (With these two changes, best will store what it is supposed to store.) Once we make those changes, we can finish fixing the sort completely by replacing the line

```
array[i] = best;
```

with three lines of code (where one more variable is declared). Show the three line fix, assuming that best stored the conceptually correct value.

```
int tmp = array[i];
array[i] = array[best];
array[best] = tmp;
```

Grading: 1 pt per line, there's two standard ways to do this and some other ways, give any valid method to swap the two variables full credit.

3) (10 pts) DSN (Bitwise Operators)

In the game of NIM, there are several piles with stones and two players alternate taking 1 or more stones from a single pile, until there are no more stones left. The person who takes the last stone wins. It turns out that if it's someone's turn, if they play optimally, they can win as long as the bitwise XOR of all of the number of stones in each pile is not equal to 0. Write a function that takes in an array of values representing the number of stones in the piles of NIM and the length of that array, and returns 1, if the current player can win, and 0 otherwise, assuming both players play optimally.

```
int canWinNIM(int piles[], int numPiles) {

    // Grading: 1 pt to initialize.
    int res = 0;

    // Grading: 3 pts loop, 4 pts XOR
    for (int i=0; i<numPiles; i++)
        res ^= piles[i];

    // Grading: 2 pts, give 1 pt if it says return res, since
    // prompt asks to specifically return 1 in winning case.
    return res != 0;


}
```