Computer Science Foundation Exam

August 8, 2020

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* 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	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.

1) (5 pts) DSN (Dynamic Memory Management in C)

Suppose we have a structure to store information about cases of juice. The structure is shown below: the name of the juice in the case is statically allocated. The structure also contains the number of containers of juice in that case. Complete the function below so that will takes 2 parameters: the name of a juice and an integer. Your function should create a new case of juice by allocating space for it, copying in the contents specified by the formal parameters into the struct and returning a pointer to the new case. You may assume that the pointer new_name is pointing to a valid string storing the name of a juice for which memory has already been allocated and is 127 or fewer characters.

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

struct juice_case {
     char name[128];
     int num_bottles;
};

struct juice_case* create_case(char *new_name, int new_number) {

     // allocate space for a new case - 2 points
     struct juice_case *new_case = malloc(sizeof(struct juice_case));

     // add the name for the case - 1 point
     strcpy(new_case->name, new_name);

     // add the number for the case - 1 point
     new_case->num_bottles = new_number;

     // return the case - 1 point
     return new_case;









}
```

2) (10 pts) ALG (Linked Lists)

Suppose we have a linked list implemented with the structure below. The function below takes in a pointer, **head**, to a linked list which is guaranteed to store data in strictly ascending order. If the list doesn't contain the value 3, the function should create a struct node storing 3 in its data component, insert the node so that the listed pointed to by head stores its data, including 3, in strictly ascending order, and returns a pointer to the front of the resulting list. If a node already exists storing 3 in the list pointed to by head, then return head and make no changes to the list.

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

node* addValue3(node* head) {

    if ( head == NULL || head->data > 3 ) {
        node* tmp = malloc(sizeof(node));
        tmp->data = 3;
        tmp->next = head;
        return tmp;
    }

    if ( head->data == 3 )
        return head;

    node* iter = head;
    while (iter->next != NULL && iter->next->data < 3 )

        iter = iter->next;

    if ( iter->next != NULL && iter->next->data == 3 )
        return head;

    node* tmp = malloc(sizeof(node));
    tmp->data = 3;
    tmp->next = iter->next;
    iter->next = tmp ;
    return head;
}
```

Grading: 1 pt per slot, record an integer grade. If two slots are partially correct, you may just take 1 point off.

3) (10 pts) DSN (Stacks)

Suppose we have implemented a stack using a linked list. The structure of each node of the linked list is shown below. The stack structure contains a pointer to the head of a linked list and an integer, size, to indicate how many items are on the stack.

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

typedef struct stack {
    struct node *top;
    int size;
} stack;
```

The generalized Towers of Hanoi game can be represented by **`numTowers`** stacks of integers, where the values in each stack represent the radii of the disks from the game for the corresponding tower. Recall that a valid move involves taking a disk at the top of one stack and placing it on the top of another stack, so long as that other stack is either empty or the disk currently at the top of the other stack is bigger than the disk about to be placed on it. Complete the function below so that it takes in an array of stacks representing the contents of the towers in Towers of Hanoi and prints out all of the valid moves that could be made from that state, but doesn't move anything. You may assume that the array of stacks passed into the function represent a valid state in a Towers of Hanoi game, where the value stored in the stack is the corresponding disk radius and the disk radii range from 1 to n, for some positive integer n. Assume that you have access to the following functions that involve a stack and that they work as described:

```
// Returns the value stored at the top of the stack pointed to by s. If stack pointed to by s is empty, a
// random value is returned.
int peek(stack *s);

// Returns 1 if the stack pointed to by s is empty, and 0 otherwise.
int isEmpty(stack *s);

void printValidMoves(stack towers[], int numTowers) {

    for (int i=0; i<numTowers; i++) {
        for (int j=0; j<numTowers; j++) {

            if ( isEmpty(&towers[i]) ) continue;

            if ( isEmpty(&towers[j]) || peek(&towers[i]) < peek(&towers[j]) )

                printf("Valid Move from tower %d to tower %d.\n", i, j);
        }
    }
}
```

Grading: 3 pts first slot, 3 pts second slot, 4 pts last slot

Computer Science Foundation Exam

August 8, 2020

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	5	ALG	
2	10	ALG	
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.

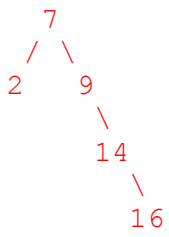
1) (5 pts) ALG (Binary Search Trees)

Consider the following tree traversals:

Pre-order: 7 2 9 14 16
Post-order: 2 16 14 9 7
In-order: 2 7 9 14 16

Is it possible for a **single** binary **search** tree to give rise to all three of those traversals? If so, draw the tree. If not, clearly explain why it's not possible.

Solution:



Grading:

5/5 for correct answer.
3/5 if they're very close (e.g., just one value is out of place)
0/5 otherwise (including if they try to explain that this isn't possible)

2) (10 pts) ALG (Heaps/Hash Tables)

Consider the following hash table and the strings it already contains, along with the hash function being used to insert strings into the table. Assume the table only stores alphabetic strings.

Note: The length of the hash table is **11**.

llama	xenon	want	yurt		mop	nook			uvula	
0	1	2	3	4	5	6	7	8	9	10

```
// This function (which is pretty bad for hashing strings in the real
// world, by the way) assumes str is non-NULL and non-empty.
int hash(char *str)
{
   // Note: This converts letters on the range 'a' through 'z' or
   // 'A' through 'Z' to integers on the range 0 through 25.
   // For example: 'a' -> 0, 'b' -> 1, ..., 'z' -> 25.
   return (tolower(str[0]) - 'a')%11;
}
```

For each of the following questions, refer to the original hash table above. For example, in part (b), refer to the original table – **not** the table that contains the string you come up with in part (a).

a. (2 pts) Give a string that, if **inserted** into the table above using **quadratic probing**, would cause us to encounter the **minimum** number of collisions possible.

Any string starting with: e, h, i, k, p, s, t, or v (… which lead to cells 4, 7, 8, and 10)
Grading: all or nothing

b. (2 pts) Give a string that, if **inserted** into the table above using **quadratic probing**, would cause us to encounter the **maximum** number of collisions possible.

Any string starting with: c, f, n, q, or y (… which lead to cells 2 and 5) **Grading:** all or nothing

c. (5 pts) Give all the alphabetic letters someone could have used to start their string in order to give a correct answer for part (b) of this problem.

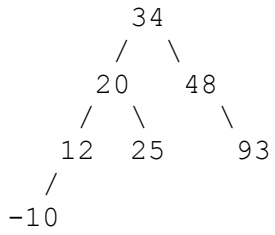
c, f, n, q, y (no need to also list uppercase letters) **Grading:** 1 pt each

d. (1 pt) Give a string that, if inserted into the table above using **linear probing**, would cause us to encounter the **maximum** number of collisions possible.

Any string starting with: a, l, or w (… which lead to cell 0) **Grading:** all or nothing

3) (10 pts) ALG (AVL Trees)

List the ranges of all the integer values that would cause a **double** rotation to occur if inserted into the following AVL tree (as opposed to a single rotation or no rotation at all). (For example: "-10 through -5 and any value greater than 93.") You may assume we do not allow the insertion of duplicate values into the tree. **Note: A double rotation can alternately be described as a restructuring where, out of the three nodes that need to move structurally, the new root node was previously two levels below the node that needs to be restructured.) These cases are also called the C-A-B and A-C-B cases.**



Solution:

All values from -9 through 11
All values from 49 through 92

Grading:

First, give +5 for **each** of the two ranges above. (**Note:** We can accept "-10 through 12" and "48 through 93," since the problem specifies that duplicates would not be inserted.)

If a range is given but incomplete by more than an off by one error, award 2 out of 5 points. (Something like -3 to 7.)

If a range is given with an off by one error, take off 1 pt per off by one error on an essentially correct range out of the 5 pts.

If a range is given in addition to correct ranges, subtract 2 pts for an extraneous range being given, capping any score at zero.

So, if no valid ranges are given, then automatically 0 of 10. If one valid range is given in full and two invalid ranges are given, this would be 5 - 2 - 2 = 1 point. If just one valid range is given, that would be 5 points, If one valid range is given with one off by one error, that's 4 points, etc.

Computer Science Foundation Exam

August 8, 2020

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)

There is a very long corridor of rooms, labeled 1 through n, from left to right. It is reputed that in the very last room, room *n*, there is the Treasure of the Golden Knight. Unfortunately, you don't know what *n* is equal to. Whenever you are in a particular room, you are allowed to ask questions of the form, "Is there a room 2^k slots to the right of my current location?", where k is a non-negative integer. For a fee, Knightro, an omnipresent, omnipotent, omniscient knight, will answer your question correctly, with either "yes" or "no." After you ask 1 or more questions from a single room, Knightro will move you, for free, to any of the rooms you asked a question about for which he replied "yes." Your goal is to get to room n by asking as few questions as possible, to reduce the fee that you pay Knightro. Devise a strategy to find the value of n and clearly outline this strategy. How many questions, in terms of n, will your strategy use, in the worst case? Answer, with proof, this last question with a Big-Oh bound in terms of n. **(Note: Any strategy that works will be given some credit. The amount of credit given will be based on how efficient your strategy is, in relation to the intended solution.)**

One strategy that is fairly efficient, is as follows:

1. When in room 1, ask the questions for each value of k successively, until receiving the first no answer. Thus, if there is a room 2^m rooms to the right, but NOT 2^{m+1} rooms to the right, stop asking questions and ask to be moved to the room 2^m rooms to the right, room $1+ 2^m$.

2. If there was no "yes" response at all during step 1, then the answer is n = 1. If this isn't the case, go onto the next steps.

3. Create a variable called cJump and set it equal to m.
4. Let cur equal the current room number you are in.
5. While cJump is greater than equal to 0, do the following:

    a. Ask the question, "Can I move to the right by $2^{curJump}$ number of rooms?
    b. While curJump is non-negative and the answer to question in step a is no, subtract 1 from curJump.
    c. If curJump is non-negative, update cur by adding $2^{curJump}$ to it (asking Knightro to move you)

Step one takes O(log n) steps, since we know that $2^m <= n$ and $2^{m+1} > n$. Solving the former inequality shows that $m <= \log_2 n$, and we ask m+1 questions, we've asked O(log n) questions in this step.

Steps 2, 3 and 4 take O(1) time. Even though the loop for #5 has a loop in it, since curJump never gets incremented, the total number of times Steps 5a and 5b run is m+1. (It's always guaranteed to decrement once every time it runs due to addition with powers of two.) Thus, the total run time of step #5 is O(log n) as well.

Adding, we get a run-time of O(log n).

Grading: 10 pts max O(lg n) strategy, 9 pts max $O(\lg^2 n)$ strategy, 4 pts max O(n) strategy. Within a strategy, award points for the explanation and run-time proof as follows (7/3, 6/3, 2/2). If there is a run time better than O(n) but not as good as $O(\lg^2 n)$, give something in between 4 and 9 points.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm to find a particular value takes O(log(n)) time where n is the total number of values. On a data set of $n = 2^{30}$ it took 1.2 seconds to find the desired value. How many **milliseconds** will it take to find a value in a data set with $n = 2^{20}$? (Note: for ease of computation, you may use a logarithm with base 2.)

The runtime is in seconds can be expressed as $c\log_2(n)$ where c is some constant. We can find the c by plugging in $n = 2^{30}$ with the answer as 1.2 seconds. We find that

$$1.2s = c\log_2(2^{30})$$
$$1.2s = c\log_2(2^{30})$$
$$1.2s = 30c$$
$$\frac{1.2s}{30} = c$$

To solve for the question we plug 2^{20} for n.

$$answer = \frac{1.2s}{30}\log_2(2^{20})$$
$$= \frac{1.2s}{30} \times 20$$
$$= \frac{2\ (1.2s)}{3}$$
$$= .8s$$

Convert to milliseconds

$$\boldsymbol{\underline{answer = 800ms}}$$

Grading:
Find c, 2 pts.
Plugging in 2^{20}, 2 pts.
Correct answer by converting, 1 pts.

3) (10 pts) ANL (Summations)

Using the fact that if $x \neq 1$, then $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$, for positive integers n, determine the following summation, in terms of n (assume n is a positive integer):

$$\sum_{i=2n+1}^{3n} 4^i$$

First, notice that we can factor out 4^{2n+1} from each term of our summation. Next, we can re-index the summation by noticing that inside the new sum, the terms are $4^0 + 4^1 + \ldots + 4^{n-1}$. Formally, we set j = i - (2n+1).

$$\sum_{i=2n+1}^{3n} 4^i = 4^{2n+1} \sum_{i=2n+1}^{3n} 4^{i-(2n+1)}$$

$$= 4^{2n+1} \sum_{j=0}^{n-1} 4^j$$

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

$$= \frac{4^{3n+1} - 4^{2n+1}}{3}$$

Another way to solve the sum is to take the sum from i=1 to 3n, and subtract from it the same sum form i=1 to 2n. If we proceed in this way, we'll get $\frac{4^{3n+1}-1}{4-1} - \frac{4^{2n+1}-1}{4-1}$, after evaluating both sums. Notice that both terms equal to one-third (first -, second +) cancel out and that we arrive at the same answer as above.

Grading Method #1: 3 pts factor out, 3 pts rewrite sum, 3 pts apply formula, 1 pt final answer (note, final answer can be factored form instead.)

Grading Method #2: 2 pts split sum, 3 pts apply formula first sum, 3 pts apply formula second sum, 2 pts algebra to get to final answer

Computer Science Foundation Exam

August 8, 2020

Section II B

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 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	5	DSN	
2	10	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.

1) (5 pts) DSN (Recursive Coding)

Write a ***recursive*** function that returns the sum of all of the even elements in an integer array ***vals***, in between the indexes ***low*** and ***high***, inclusive. For example, for the function call sumEven(vals, 3, 8) with the array vals shown below, the function should return 24 + 8 + 10 = 42, since these three numbers are the only even numbers stored in the array in between index 3 and index 8, inclusive.

index	0	1	2	3	4	5	6	7	8	9
vals[i]	15	13	28	19	24	8	7	99	10	14

```
int sumEven(int vals[], int low, int high) {

    if (low > high) return 0;        // 1 pt base case.

    int res = 0;                     // 2 pts assign variable based
    if (vals[low]%2 == 0)            // on current term.
        res = vals[low];

     return res + sumEven(vals, low+1, high); // 2 pts return and
                                              // recursive call

}
```

2) (10 pts) ALG/DSN (Sorting)

(a) (5 pts) Consider running a Bubble Sort on the array shown below. How many swaps will execute for the duration of the algorithm running on the array shown below? Explain how you got your answer.

97	16	45	63	13	22	7	58	72

Reasoning:

On iteration #1, 97 will swap with all items, for a total of 8 swaps.
On iteration #2, 63 will swap with 13, 22, 7 and 58, for a total of 4 swaps.
On iteration #3, 45 will swap with 13, 22 and 7 for a total of 3 swaps.
On iteration #4, 16 swaps with 13, and 22 will swap with 7 for a total of 2 swaps.
On iteration #5, 16 will swap with 7, for a total of one swap.
On iteration 36, 13 will swap with 7, for a total of one swap.

Adding up, we get 8 + 4 + 3 + 2 + 1 + 1 = 19 swaps.

A perhaps, easier way to solve this is to realize that all swaps are between inverted elements, namely, pairs of items that are out of place in the original array, meaning that in the pair the larger item appears first. Thus, we can count all the inversions:

(97, 16), (97, 45), (97, 63), (97, 13), (97, 22), (97, 7), (97, 58), (97, 72),
(16, 13), (16, 7),
(45, 13), (45, 22), (45, 7),
(63, 13), (63, 22), (63, 7), (63, 58)
(13, 7)
(22, 7)

which is 19 inverted pairs.

Grading: 4 pts for correct answer, 1 pt for reason. If answer is 18 or 20 and reason is valid, take off 1 pt, if answer is 17 or 21 and reason is valid, take off 2 pts.

(b) (5 pts) List the **best case** run time of each of the following sorting algorithms, in terms of n, the number of items being sorted. Assume all items being sorted are distinct.

(i) Insertion Sort O(n) **Grading: 1 pt each**

(ii) Selection Sort $O(n^2)$

(iii) Heap Sort O(nlgn)

(iv) Merge Sort O(nlgn)

(v) Quick Sort O(nlgn)

3) (10 pts) DSN (Backtracking)

A "unique" positive integer of n digits is such that no two adjacent digits differ by less than 2. Specifically, given an n digit number, $d_0d_1\ldots d_{n-1}$, where d_0 is the most significant digit, (and thus, this one digit can't be 0), $|d_i - d_{i+1}| \geq 2$ for all i ($0 \leq i \leq n-2$). Consider the problem of printing out all "unique" positive integers of n digits via backtracking, in numerical order. Fill in the code below to complete the task. (To run the code, one would have to call printWrapper with their desired parameter.)

```
#include <stdio.h>
#include <math.h>
void print(int number[], int n);
void printWrapper(int n);
void printRec(int number[], int k, int n);

void printWrapper(int n) {
    int* array = malloc(sizeof(int)*n);
    printRec(array, 0, n);
    free(array);
}

void printRec(int number[], int k, int n) {
    if (k == n) {
        print(number, n) ;                    // Grading: 1 pt

        return ;                              // Grading: 1 pt
    }

    int start = 0;

    if ( k==0 )                               // Grading: 1 pt

        start = 1 ;                           // Grading: 1 pt
    for (int i=start; i < 10 ; i++) {         // Grading: 1 pt

        if (k > 0 && abs(number[k-1]-i)<2 )   // Grading: 2 pts
            continue;

        number[ k ] = i ;                     // Grading: 2 pts

        printRec(number, k+1, n) ;            // Grading: 1 pt
    }
}

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