Computer Science Foundation Exam

May 22, 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	10	DSN	
2	10	DSN	
3	5	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) (10 pts) DSN (Dynamic Memory Management in C)

Suppose we have a stack implemented with an array as shown in the structure below. Write a function called grow_stack that will increase the stack's capacity while preserving the exact values currently in the stack and their current locations. Your function should take 2 parameters: a pointer to the current stack and an integer representing the amount to increase the stack's capacity by. **You may not use the realloc function.** You may assume s isn't NULL and pts to a valid struct stack. You may assume that capacity stores the current size of the array that the pointer array is pointing to and that top represents the number of items currently in the stack (items are stored in indexes 0 through top-1).

```
struct Stack {
     int *array;
     int top;
     int capacity;
};

void grow_stack(struct Stack *s, int increase) {

     // Calculates new size as an increase to current capacity
     // 1 point
     int new_size = s->capacity + increase;

     // Has a mechanism to prevent "losing" the current array pointer
     // 2 points
     int *hold = s->array;
     int i;

     // Allocates space for the increased stack array
     // 2 points
     s->array = malloc(sizeof(int) * new_size);

     // Copies values from old array to new array
     // 3 points
     for(i = 0; i < s->top; i++)
               s->array[i] = hold[i];

     // Cleans up old memory space
     // 1 point
     free(hold);

     // Updates capacity
     // 1 point
     s->capacity = new_size;



}
```

2) (10 pts) DSN (Linked Lists)

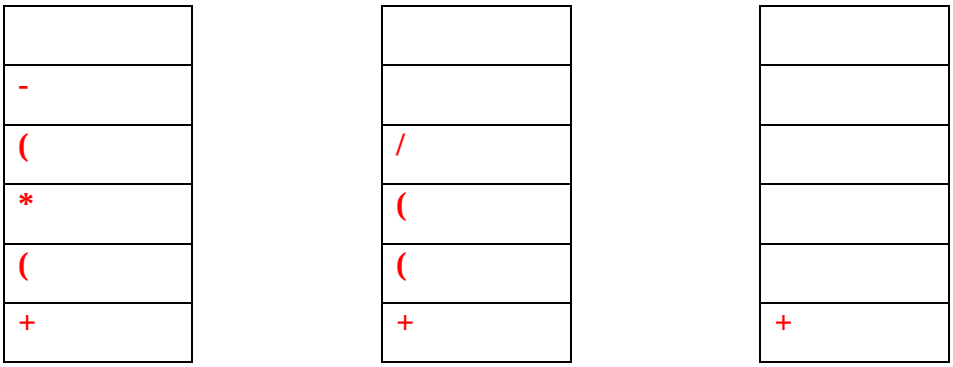
Suppose we have a singly linked list implemented with the structure below. Write a function that will convert it into a circular linked list and return the pointer to the beginning of the circle.

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

struct node * make_circle(struct node * head) {

     // checks to see if head is null
     // 2 points
     if (head == NULL)
        return NULL;

     // create a new node to traverse list
     // 1 point
     struct node *helper = head;

     // advance through list until we find the end
     // 4 points
     while (helper->next != NULL)
        helper = helper->next;

     // connect the end of the list to the front
     // 2 points
     helper->next = head;

     // return the head of the list
     // 1 point
     return head;




}
```

3) (5 pts) ALG (Stacks)

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (1, 2, and 3) in the infix expression.

A + (B * (C – D 1 **)) + ((E /** 2 **F) + G)** 3 **+ H**



1 2 3

Resulting postfix expression:

A	**B**	**C**	**D**	**-**	*****	**+**	**E**	**F**	**/**	**G**	**+**	**+**	**H**	**+**					

Grading: 1 point for each stack, 2 points for the whole expression (partial credit allowed.)

Computer Science Foundation Exam

May 22, 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	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.

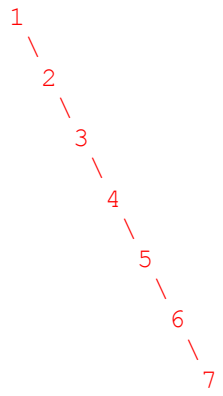
1) (5 pts) DSN/ALG (Binary Trees)

Draw a single **binary search tree** that meets all the following conditions:

- The tree contains 7 nodes.
- The tree's pre-order traversal is the same as its in-order traversal.
- The tree does not contain any duplicate values.

If it is not possible to draw such a tree, say so and explain why not.

Here's one such tree. Basically, it has to devolve into a linked list like this:



Pre-order: 1 2 3 4 5 6 7

In-order: 1 2 3 4 5 6 7

Grading: 0 if answered not possible,
If a tree is given 1 pt for a valid binary search tree
4 pts if it goes all right (correct)
Take off 1 pt for each left node (cap at 4 pts off)

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

Consider the following strings and their corresponding hash values, which have been generated by some hash function:

hash("squiggle") = 301

hash("giggle") = 174

hash("haggle") = 431

hash("gaggle") = 263

hash("straggle") = 361

a) (7 pts) Insert the strings above into the following hash table using **quadratic probing**. In doing so, insert them in the order given above (i.e., starting with "squiggle", then "giggle", and so on). Note that the hash table's length is **11** (not 10).

		haggle		squiggle			straggle		giggle	gaggle
0	1	2	3	4	5	6	7	8	9	10

Grading:

+3 pts for getting "straggle" in index 7
+4 pts for all the rest being correct (1 pt per item)

b) (3 pts) What is one hash value, *h*, between 100 and 500 (inclusively) that would cause a collision to occur in your final table from part (a) of this problem, but which also satisfies all of the following additional restrictions:

h % *table_length* != hash("squiggle") % *table_length*

h % *table_length* != hash("giggle") % *table_length*

h % *table_length* != hash("haggle") % *table_length*

h % *table_length* != hash("gaggle") % *table_length*

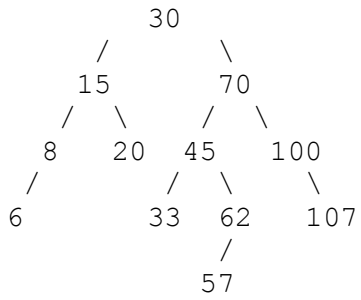
h % *table_length* != hash("straggle") % *table_length*

Any value *h* such that *h* % 11 = 7 will do the trick (e.g., 106, 117, 128, and so on).

Grading: +3 pts for a correct value. 0 otherwise.

3) (10 pts) ALG (AVL Trees)

This question deals with the AVL Tree shown below:



(a) (7 pts) How many restructure operations (a single restructure operation is either a single or double rotation) would occur if each of the following items was deleted? Consider each item separately as being the only item being deleted from the tree shown above. (Note: It's possible that the answer to some parts is 0.)

Item to Delete	**Number of Restructure Operations**
6	**1 (at 30)**
20	**2 (at 15, then 30)**
33	**1 (at 45)**
57	**0**
62	**0**
100	**1 (at 70)**
107	**1 (at 70)**

Grading: 1 pt per answer, no exceptions (node of restructuring isn't necessary)

(b) (3 pts) What is the fewest number of consecutive insertion operations that would need to occur to force a rebalance at the root node of the given tree in the picture? (Hint: In order for this to occur, there has to be the requisite height imbalance at the root node 30, and no other imbalances on the path from the last inserted node to the root.)

5 **Grading: 3 pts for the correct answer, 2 pts for 4 or 6, 1 pt for 3 or 7, 0 pts otherwise**

Note: In order for this to occur, the right side would have to become a height 2 more than the left of 30. But since the node 70 is already not perfectly balanced, we must first insert 2 items (one example that works is 90 followed by 101) so that 70 is perfectly balanced. We must then get 45 perfectly balanced. One way to do this is to insert 31. Finally, either 33 or 62 must be perfectly balanced. One way to do this is inserting 67. Finally, from this point, inserting 65 will ultimately trigger a rebalance operation all the way up at 30, which, for this example, would make 45 the new root of the tree. More generally, to force a rebalance at a particular node, both trees underneath it must be valid AVL trees, and in general, for a height of n, the minimum number of nodes to create a valid AVL tree is $F_{n+3} - 1$, where F_{n+3} is the $(n+3)^{rd}$ Fibonacci number. This dictates that we need the right subtree of 30 to have $F_{4+3} - 1 = 13 - 1 = 12$ nodes. Since it currently has 7, the minimum number of insertions theoretically possible is 12 - 7 = 5. The construction given shows that it's possible to achieve this theoretical minimum.

Computer Science Foundation Exam

May 22, 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	5	ANL	
2	10	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) (5 pts) ANL (Algorithm Analysis)

Given an array, `vals`, of size n, one can determine the sum of the elements in the array from index i through index j (i ≤ j), inclusive, simply by running a for loop through the elements:

```
int sum = 0;
for (int z=i; z<=j; z++)
    sum += vals[z];
```

This type of sum is known as a contiguous subsequence sum.

Note: There are more efficient ways to do this if many sums of this format need to be determined, but for the purposes of this problem, assume that this is how such a sum is determined.

(a) (3 pts) What is the worst case run time of answering **q** questions about contiguous subsequence sums on an array of size **n**? Express your answer in Big-Oh notation, in terms of both **n** and **q**. Give a brief justification for your answer.

The worst case query is when i = 0 and j = n-1. Answering this one query takes O(n) time, since we loop through each element of the array. If we were to answer q of these queries, it would take **O(nq)** time.

Grading: 1 pt for the answer, 2 pts for the reason. A specific worst case scenario doesn't need to be given, but simply referencing that a large query may take time proportional to the array size is good enough.

(b) (2 pts) What is the best case run time of answering **q** questions about contiguous subsequence sums on an array of size **n**? Express your answer in Big-Oh notation, in terms of both **n** and **q**. Give a brief justification for your answer.

The best case query is where i = j. Running such a query takes O(1) time. Running q of these best case queries takes **O(q)** time. Note that in this case, the answer does not depend on n at all.

Grading: 1 pt for the answer, 1 pt for the reason

2) (10 pts) ANL (Algorithm Analysis)

Querying a user in our data base of 10^4 users take about 10 milliseconds. The runtime of the query is logarithmic with respect to the number of users. Namely, if there are n users, a query takes O(lg n) time. About how many users can we support while taking no more than 20 milliseconds per query?

Let t(n) represent the run time of a single query. Then, for some constant c, we have

t(n) = c(lg n)

Let n be the answer to the question. Using the given information, we can set up a ratio between the two run-times as follows:

$$\frac{c(\lg n)}{c(\lg 10^4)} = \frac{20\ ms}{10\ ms}$$

$$\frac{\lg n}{4lg10} = 2$$

$$\lg n = 8lg10$$

$$\lg n = lg10^8$$

$$\boldsymbol{n = 10^8}$$

Note: There are quite a few other ways to arrive at the result.

Grading: 2 pts for setting up the run time as a constant times log.
2 pts for setting up a variable to store the answer to the question.
2 pts for writing down the appropriate equations.
4 pts for solving the equations for the correct value of n. (Give partial as needed.)

3) (10 pts) ANL (Summations)

What is the closed form of the following summation? Your solution should be a function in terms of n. For full credit work must be shown.

$$\sum_{i=0}^{n}\sum_{j=0}^{i} 2^j$$

$$\sum_{i=0}^{n}\sum_{j=0}^{i} 2^j = \sum_{i=0}^{n}(2^{i+1}-1)$$

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

$$= \sum_{i=0}^{n} (2)2^i - (n+1)$$

$$= 2\sum_{i=0}^{n} 2^i - (n+1)$$

$$= 2(2^{n+1}-1) - (n+1)$$

$$= 2^{n+2} - 2 - n - 1$$

$$= \mathbf{2^{n+2} - n - 3}$$

Grading: 3 pts inner sum
1 pt split
1 pt second sum
4 pts first sum (lots of ways to break this down)
1 pt simplification at the end

Computer Science Foundation Exam

May 22, 2021

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	10	DSN	
2	10	ALG	
3	5	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)

We call a list of positive integers **nice** if each pair of consecutive elements shares a common divisor greater than 1. For example, the list 18, 15, 35, 40 is nice because 18 and 15 are both divisible by 3, 15 and 35 are both divisible by 5, and 35 and 40 are both divisible by 5 as well. Given a list of unique positive integers, complete the recursive code below so that it counts the number of permutations of an original list that are nice lists. (For example, for the 4 numbers above, there are 12 nice arrangements: [18,15,35,40], [18,15,40,35], [18,40,15,35], [18,40,35,15], [15,18,40,35], [15,35,40,18], [35,15,18,40], [35,15,40,18], [35,40,18,15], [35,40,15,18], [40,18,15,35], and [40,35,15,18].)

The strategy the code below uses is to go through each permutation of the integers 0, 1, 2, …, n-1, where n is the number of values being considered. For each permutation, the evaluation function returns 1 if the order of the values creates a nice list, and 0 otherwise. Over all permutations, these values are added. Note: the gcd function provided returns the greatest common divisor shared by the 2 positive integer parameters.

```
int numArr(int values[], int perm[], int used[], int k, int n) {

    if (k == n)

        return eval(values, perm, n);

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

        if ( !used[i] ) {

            used[i] = 1;

            perm[k] = i;
            res += numArr(values, perm, used, k+1 , n );

            used[i] = 0;
        }
    }
    return res;
}

int eval(int values[], int perm[], int n) {

    for (int i=0; i< n-1 ; i++)

        if ( gcd( values[perm[i]] , values[perm[i+1]] ) == 1 )
            return 0;
    return 1;
}

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a%b);
}
```

Grading: 1 pt per slot, gcd slots can be exchanged. Slot must be perfect to get point.

2) (10 pts) ALG (Sorting)

(a) (5 pts) Show the state of the following array below after each iteration of an Insertion Sort. The results after the first iteration and last iteration are included for clarity.

Original	13	6	9	44	18	22	3	11
1st iteration	6	13	9	44	18	22	3	11
2nd iteration	6	9	13	44	18	22	3	11
3rd iteration	6	9	13	44	18	22	3	11
4th iteration	6	9	13	18	44	22	3	11
5th iteration	6	9	13	18	22	44	3	11
6th iteration	3	6	9	13	18	22	44	11
7th iteration	3	6	9	11	13	18	22	44

Grading: 1 pt per row, whole row has to be perfect to get the point

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

(i) Insertion Sort $O(n^2)$

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

(iii) Heap Sort O(nlgn) or O(nlog(n))

(iv) Merge Sort O(nlgn) or O(nlog(n))

(v) Quick Sort $O(n^2)$

Grading: 1 pt for each slot, all or nothing.

3) (5 pts) ALG (Bitwise Operators)

There are a total of 25 cards, numbered 0 through 24. We can represent a set of cards with a single integer by setting the i^{th} bit to 1 if the set contains card i, and setting the bit to 0 otherwise. For example, the set of cards {2, 6, 7} would be stored as the integer 196, since $196 = 2^7 + 2^6 + 2^2$. Two sets of cards are disjoint, if and only if no card appears in both sets. Complete the function below so that it returns 1 if the sets of cards represented by the integers set1 and set2 are disjoint, and returns 0 if they are not disjoint. (For example, disjoint(196, 49) should return 1 because $49 = 2^5 + 2^4 + 2^0$, and there is no common value in the two sets {2, 6, 7} and {0, 4, 5}. On the other hand, disjoint(196, 30) should return 0 because $30 = 2^4 + 2^3 + 2^2 + 2^1$, so that card number 2 is included in both sets 196 and set 30.)

```
// Pre-condition: set1 and set2 are bitmasks in between 0 and
//                (1<<25)-1.
// Post-condition: Returns 1 if the two bitmasks are disjoint,
//                 meaning that the sets they represent don't have
//                 any items in common, and returns 0 otherwise, if
//                 the two represented sets do have common items.
int disjoint(int set1, int set2) {

    return (set1 & set2) == 0;

}
```

Grading: 1 pt return
3 pts and between sets
1 pt checking if that and is 0 or not.

Note: Many other ways to do this that are less succinct. As long as bitwise ops are used, give full credit even if there is a loop (25 times) that looks only at pairs of bits at a time and does the right thing. So something like this would get full credit:

```
int disjoint(int set1, int set2) {

    for (int i=0; i<25; i++) {
        if ( (set1&1) && (set2&1) ) return 0;
        set1 >>= 1;
        set2 >>= 1;
    }

    return 1;

}
```