Computer Science Foundation Exam

August 27, 2022

Section A

BASIC DATA STRUCTURES

SOLUTION

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 <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) (10 pts) DSN (Dynamic Memory Management in C)

This problem relies on the following *Player* and the *Team* struct definitions:

```
typedef struct Player
{
  char pname[50];   //player's name
  char country[50]; //player's country
  int age;
} Player;
```

```
typedef struct Team
{
    char tname[50];  // team's name
    Player *players; // all players on the team
    int numPlayers;  // number of players on the team
} Team;
```

We are making a team of players from multiple countries. There is a text file containing the details of a team, where the first line of the file contains the team name, followed by a single space, followed by the number of players on the team, *N*. The next *N* line contains the data for *N* players. Each player line contains three tokens, each separated by a space: the player name, country, and age (as an integer). Each team name, player name, and country will be a single-word string (no spaces) with a maximum length of 49. Here is a sample file:

```
NewKnights 5
Hannan USA 22
Mabel India 21
Samarina Bangladesh 21
Tamsen USA 21
Susan Mexico 22
```

Write a function that takes a file pointer and then returns a pointer to a dynamically allocated Team struct with all the information loaded into it. You can assume that the file is already opened in read mode and ready to read from the beginning of the file. Do not worry about closing the input file with *fclose()* when you finish reading it. Assume the function that opened the file and called *createTeam()* will close the file.

```
Team *createTeam (FILE *fp) {

    Team* res = malloc(sizeof(Team));          // 1 pt

    // 2 pts
    fscanf(fp, "%s%d", res->tname, &(res->numPlayers));

    // 2 pts
    res->players = calloc(res->numPlayers, sizeof(Player));

    // 4 pts - 1 pt loop, 1 pt each read/scanf
    int i;
    for (i=0; i<res->numPlayers; i++)
        fscanf(fp, "%s%s%d", res->players[i].pname,
               res->players[i].country, &(res->players[i].age));

    // 1 pt
    return res;
}
```

2) (10 pts) DSN (Linked Lists)

The structure of a node of a doubly linked list is shown below.

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

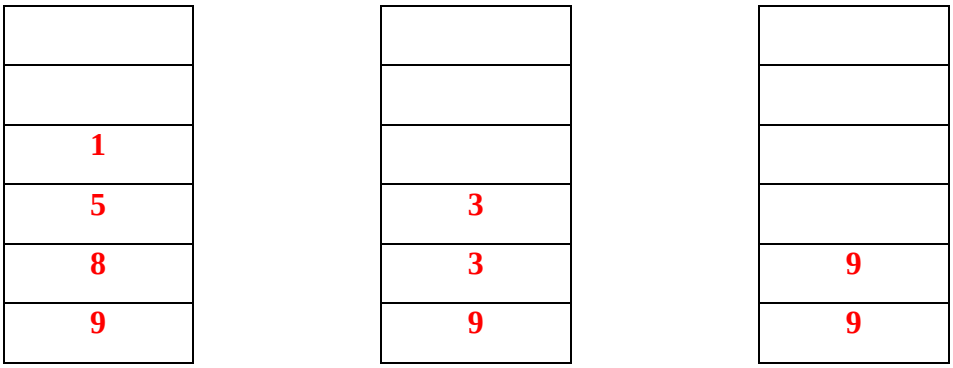
Write a function that takes in a pointer to the head of a doubly linked list (*head*) and a pointer to a node in that list (*me*), removes that node (*me*) from the list, and returns a pointer to the head node of the resulting doubly linked list. You may assume that both *head* and *me* are not NULL, and that *me* points to a node in the list pointed to by *head*.

```
node* deleteMe(node* head, node* me) {

    // Grading 2 pts: updating head for delete front case.
    if(me == head)
        head = head->next;

    // Grading: 3 pts patch next node to previous node if nec.
    if(me->next != NULL)
        me->next->prev = me->prev;

    // Grading: 3 pts patch previous node to next if nec.
    if(me->prev != NULL)
        me->prev->next = me->next;

    // 1 pt to free the designated node.
    free(me);

    // 1 pt to return the new head of the list.
    return head;
}
```

3) (5 pts) ALG (Stacks)

Evaluate the following postfix expression, showing the state of the stack at each of the indicated points.

				A			B			C				
9	**8**	**5**	**1**	*****	**-**	**3**	**2**	*****	**+**	**/**	**9**	**+**	**5**	**/**



Value of the Expression: **2**

Grading: 1 pt for each stack (must be correct to get point), 2 pts for the final answer (no partial)

Computer Science Foundation Exam

August 27, 2022

Section B

ADVANCED DATA STRUCTURES

SOLUTION

Question #	Max Pts	Category	Score
1	5	ALG	
2	10	ALG	
3	10	DSN	
TOTAL	25		

You must do all 3 problems in this section of the exam.

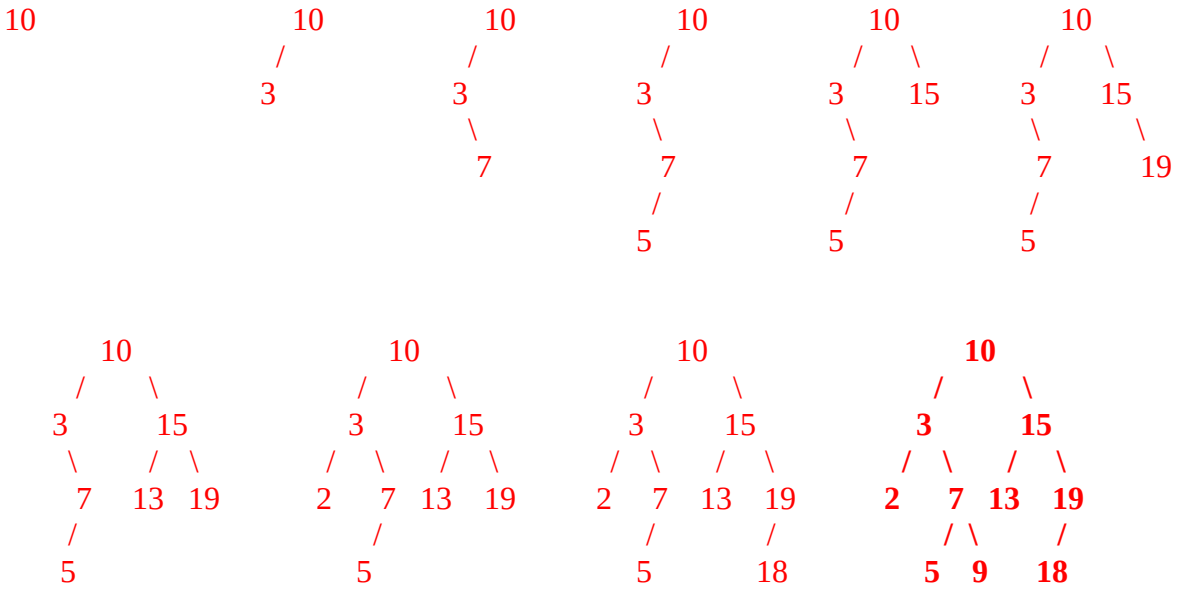
Problems will be graded based on the completeness of the solution steps and <u>not</u> graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all <u>be neat</u>. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (5 pts) ALG (Binary Trees)

Show the result of inserting the following integers into an initially empty **binary search tree.** Please draw a box around your final answer. This is the only portion of your work that will be graded.

Insert 10, 3, 7, 5, 15, 19, 13, 2, 18 and 9.

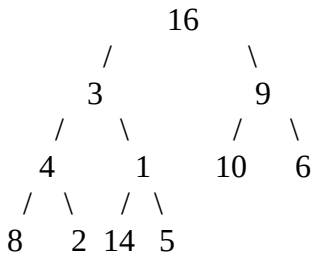
Here is the tree at each stage:



Grading: ½ pt per location of each item in final tree, record score as an integer, rounding down.

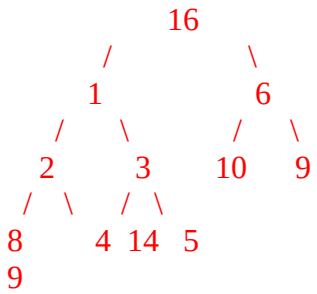
2) (10 pts) ALG (Binary Heaps)

Consider running the Make Heap/Heapify algorithm on the following set of random values (shown stored in a heap structure) to convert it to a min heap. In doing so, exactly six swaps will occur between adjacent values in the heap. When the algorithm concludes, the structure will be valid minheap. Draw the structure of the tree after (a) the first three swaps have completed, and (b) when the Make Heap algorithm completes (the final valid minheap). Here is the initial tree drawing of the values:



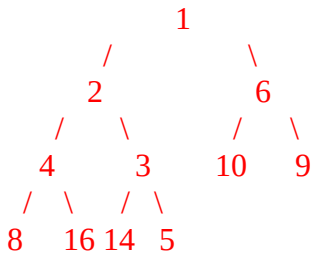
(a) Draw the minheap here after completing the first three swaps.

(a) After we run percolate down on the node initially storing the 1, then the 4, then the 9 and then the 3, we make three swaps: (4, 2), (9, 6) and (3, 1). Here is the picture of the heap after those swaps:



(b) Draw the final minheap.

(c) Then, we run percolate down on the root node and it will successively swap with 1, 2 and 4 producing the following result:



Grading: 2 pts for swap of 4 and 2 in first pic,
2 pts for swap of 9 and 6 in first pic,
2 pts for swap of 3 and 1 in first pic,
1 pt each for placement of 1, 2, 4 and 16 in second pic

3) (10 pts) DSN (Tries)

Given a dictionary of words stored in a trie rooted at `root`, and a string, `str`, consider the problem of determining the length of the longest prefix of `str` that is also a prefix of one of the words stored in the trie. You may assume that if a link in the trie exists, a valid word is stored down that path in the trie. You may use string functions as needed, but please try to do so efficiently. (One point will be deducted for inefficient use of a particular string function.) For example, if str = "capitulate" and the trie stored the set of words {"actor", "bank", "cat", "capitol", and "caption"}, then the function should return 5 since, the first five letters of "capitulate" are the same as the first five letters of "capitol", and no other word stored in the trie shares the first six letters with "capitulate."

Complete the code for the function that solves this problem below. `root` is a pointer to the root of the trie and `str` is a pointer to the string. **You may assume that at least one word is stored in the trie.** The function signature and trie node struct definition are given below. Note that due to the function signature, **you must write your code iteratively.**

```
#include <string.h>
typedef struct TrieNode {
    struct TrieNode *children[26];
    int flag;  // 1 if the string is in our trie, 0 otherwise
} TrieNode;

int maxPrefixMatch(TrieNode* root, char* str) {


    // Grading: 2 pts - give 1 pt if called multiple times.
    int len = strlen(str);

    // Grading: 2 pts loop
    for (int i=0; i<len; i++) {

        // 3 pts - 2 pts for statement, 1 pt for placement (before NULL check)
        root = root->children[str[i]-'a'];

        // 2 pts - 1 pt for check, 1 pt for return
        if (root == NULL) return i;
    }

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

Computer Science Foundation Exam

August 27, 2022

Section C

ALGORITHM ANALYSIS

SOLUTION

Question #	Max Pts	Category	Score
1	10	ANL	
2	10	ANL	
3	5	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 <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.h, stdio.h, math.h, string.h) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

Write down the **worst case run-times** for each of the requested operations. **You may assume that each operation is done with an efficient algorithm.** Please leave your answer in **simplified Big-Oh** form, in terms of the variables given in the problem. Thus, please do NOT include any leading constants or unnecessary terms. Answers such as $O(2n^2)$ or $O(n^2 + \lg n)$ **will receive no credit**, even if they are technically correct. Each part is worth 1 point.

a) Inserting **k** items, each into the front of a linked list which starts with **n** items. — $O(k)$

b) Running a floodfill on a grid with **r** rows and **c** columns. — $O(rc)$

c) Sorting **n** elements via the Quick Sort algorithm. — $O(n^2)$

d) Efficiently forming a heap out of **n** unsorted items. — $O(n)$

e) Removing all of **n** items, one by one, from a Priority Queue that originally has **n** items. — $O(nlgn)$

f) Inserting **n** items, one by one, into a Binary Search Tree. — $O(n^2)$

g) Inserting **n** items, one by one, into a AVL Tree. — $O(nlgn)$

h) Printing out the set of moves to solve the Towers of Hanoi with a tower of **n** disks. — $O(2^n)$

i) Merging two **sorted lists**, one with **r** elements, the other with **s** elements, into a single sorted list. — $O(r+s)$

j) Writing out the first 10 Fibonacci numbers. — $O(1)$

Grading: 1 pt per answer. Answer has to match exactly to get credit. Only exception is that (i) can be listed as O(max(r,s)).

2) (10 pts) ANL (Algorithm Analysis)

A program takes $O(n^3)$ time to process data about a map with **n** points of interest. For n = 100, the program completes in 25 milliseconds. The time limit for running the program has been set at a maximum of 12.8 seconds. What is the largest value of **n** for which the program is expected to complete within the time limit?

Let the run time of the program on a map with n points be $T(n) = cn^3$, for some constant c. Using the given information, we have:

$$T(100) = c(100)^3 = 25ms$$

$$c = \frac{25}{10^6} ms$$

Now, we want to find a value of n, such that $T(n) \leq 12.8$ seconds. Since we want to maximize n, set both sides equal to each other, converting 12.8 seconds to 12800 ms.

$$T(n) = \frac{25ms}{10^6} n^3 = 12800ms$$

$$n^3 = \frac{128 \times 10^2 \times 10^6}{5^2}$$

$$n^3 = \frac{2^7 \times 10^8}{5^2} = \frac{2^7 \times 2^8 \times 5^8}{5^2} = 2^{15}5^6$$

Taking the cube root of both sides, we get:

$$n^3 = 2^5 \times 5^2 = 32 \times 25 = 8 \times (4 \times 25) = \mathbf{800}$$

Grading: 1 pt setting up equation for c, 2 pts solving for c (without simplification)
2 pts setting up equation with n, 1 pt converting 12.8 to 12,800 or equivalent conversion
3 pts for algebra to get to answer
1 pt for final answer
If they get to cube root of 512,000,000, then give 8/10 (so -2 total of the last 4 pts)

Note: Please give full credit to 799, just in case someone used a strictly less than sign.

3) (5 pts) ANL (Recurrence Relations)

Using the iteration technique, just solve for the **next two** iterations of the following recurrence relation:

$$T(n) = 3T(n-1) + n^2, for\ integers\ n > 0$$
$$T(0) = 1$$

Your answers should be of the form

$T(n) = aT(n-2) + bn^2 - cn + d$ and
$T(n) = eT(n-3) + fn^2 - gn + h$, where *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h* are positive integers.

$$T(n) = 3(3T(n-2) + (n-1)^2) + n^2$$

$$T(n) = 9T(n-2) + 3n^2 - 6n + 3 + n^2$$

$$T(n) = 9T(n-2) + 4n^2 - 6n + 3$$

It follows that a = 9, b = 4, c = -6, and d = 3

Now, plug in one more iteration

$$T(n) = 9(3T(n-3) + (n-2)^2) + 4n^2 - 6n + 3$$

$$T(n) = 27T(n-3) + 9n^2 - 36n + 36 + 4n^2 - 6n + 3$$

$$T(n) = 27T(n-3) + 13n^2 - 42n + 39$$

It follows that e = 27, f = 13, g = -42 and d = 39

Grading: 2 pts first iteration, (give 1 pt if some terms correct)
3 pts second iteration (give 1 pt if 1 or 2 terms correct,
2 pts if 3 terms correct)

Take off 1 pt if they leave 1st unsimplified: $9T(n-2) + 3(n-1)^2 + n^2$
Take off 1 pt if they leave 2nd unsimplified: $27T(n-3) + 9(n-2)^2 + 3(n-1)^2 + n^2$

Computer Science Foundation Exam

August 27, 2022

Section D

ALGORITHMS

SOLUTION

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

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 <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) (10 pts) DSN (Recursive Coding)

Ten tiles each have strings of in between 1 and 4 letters on them (hardcoded in the code below). The goal of this problem is to complete the code below so it counts the number of different orders in which **all** of the tiles can be placed such that the string they form creates a palindrome (a word that reads the same forwards and backwards). All of main, as well as the function which determines if a particular ordering of the tiles forms a palindrome, **included on the next page** have been given. You may call this function in the function go. Complete the recursive function (named go) to complete the solution.

```
#include <stdio.h>
#include <string.h>
#define N 10
#define MAXLEN 5

int go(int perm[], int used[], int k, char tiles[N][MAXLEN]);
int eval(int perm[], char tiles[N][MAXLEN]);
char MYTILES[N][MAXLEN] = {"at","ta","g","cc","ccac","ca","cc","gag","cga","gc"};

int main(void) {
    int perm[N];
    int used[N];
    for (int i=0; i<N; i++) used[i] = 0;
    int res = go(perm, used, 0, MYTILES);
    printf("Number of tile orderings that create palindromes is %d\n", res);
    return 0;
}

int go(int perm[], int used[], int k, char tiles[N][MAXLEN]) {

    if (k == N)
        return eval(perm, tiles);                    // 3 pts

    int res = 0;

    for (int i=0; i<N; i++) {

        if (used[i]) continue;

        used[i] = 1;                                 // 1 pt

        perm[k] = i;                                 // 1 pt

        res += go(perm, used, k+1, tiles) ;          // 4 pts

        used[i] = 0;                                 // 1 pt
    }

    return res;
}
```

```
int eval(int perm[], char tiles[N][MAXLEN]) {

    char tmp[N*MAXLEN];
    int idx = 0;
    for (int i=0; i<N; i++) {
        int len = strlen(tiles[perm[i]]);
        for (int j=0; j<len; j++)
            tmp[idx++] = tiles[perm[i]][j];
    }
    tmp[idx] = '\0';

    for (int i=0; i<idx/2; i++)
        if (tmp[i] != tmp[idx-1-i])
            return 0;

    return 1;
}
```

2) (5 pts) ALG (Sorting)

Show the result after each iteration of performing Bubble Sort on the array shown below. For convenience, the result after the first and last iterations are provided. The first row of the table contains the original values of the array.

Iteration	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7
0	12	2	8	19	13	7	1	16
1	2	8	12	13	7	1	16	19
2	2	8	12	7	1	13	16	19
3	2	8	7	1	12	13	16	19
4	2	7	1	8	12	13	16	19
5	2	1	7	8	12	13	16	19
6	1	2	7	8	12	13	16	19
7	1	2	7	8	12	13	16	19

Grading: 1 per row, must get the whole row to get the point.

3) (10 pts) DSN (Bitwise Operators)

There are 20 light switches, numbered 0 to 19, each which control a single light. Initially, all of the lights the switches control are off. There are several buttons. Each button toggles several switches, when pressed. For example, if a button toggles the switches 3, 5 and 9, then pressing the button changes the state of the three switches 3, 5 and 9, leaving the other switches in the same state. (So, if lights 3 and 5 were on and light 9 was off, after the button is pressed, lights 3 and 5 would be off and light 9 would be on.) Each button can be stored in a single integer, where the k^{th} bit is set to 1 if that button toggles the k^{th} switch, and set to 0 if pressing the button doesn't affect the k^{th} switch. For example, the button described would be stored as the integer 552 since $2^3 + 2^5 + 2^9 = 552$. Write a function that takes in an array, ***buttons***, storing the buttons to press and an integer ***len***, representing the length of the array ***buttons*** and returns a single integer storing the state of the **lights** after each of the buttons has been pressed once, assuming that all of the lights were off before any of the button presses. The format for storing the state of the lights should be identical to the format of the buttons.

```
int pressButtons(int buttons[], int len) {


    int res = 0;                          // 1 pt

    for (int i=0; i<len; i++)             // 2 pts
        res ^= buttons[i];                // 6 pts (partial possible)

    return res;                           // 1 pt
}
```

Grading Partial: 1 pt update res, 3 pts XOR, 2 pts access buttons[i]
If they use | instead of ^ (-2), & instead of ^ (-3), + instead of ^ (-4).