Computer Science Foundation Exam

May 20, 2023

Section A

BASIC DATA STRUCTURES

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

SOLUTION

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 <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 (Dynamic Memory Management in C)

The struct below is used to define a node in a linked list. Below it is a function, ***freeList***, that is given a pointer to the front/head of a linked list. The function is supposed to free all the dynamically allocated memory associated with the linked list that the given pointer is pointing to. Unfortunately, the function does not work. Explain why the function doesn't work and propose a fix to the function, **in words only, conceptually explaining the order in which the key steps have to be executed.**

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

void freeList(node* front) {
    if (front != NULL)
        free(front);
    if (front->next != NULL)
        freeList(front->next);
}
```

Problem(s) with current code

If front isn't NULL, the memory it's pointing to gets freed. When this occurs, the link to the memory for future nodes, if they exist, is lost and can no longer be freed.

Grading: 2 pts or 0 pts, student must clearly indicate that pointer to potential list memory is inaccessible for credit.

Proposed fixes (conceptually, in words)

If front is already NULL, no action should be taken. This must be checked and taken care of first. Next, the recursive call to free the rest of the list should occur, so long as there is a rest of the list. Finally, front should be freed, last.

Grading: 1 pt for each component, described in the appropriate order.

2) (10 pts) DSN (Linked Lists)

Consider the problem of determining the number of changes in direction of a sequence of integers stored in a linked list. We define a change in direction as any three consecutive values a, b, and c in the sequence where either a > b and b < c, or where a < b and b > c. For example, the sequence 3, 8, 2, 2, 5, 6, 4, 8, 3 has 4 changes in direction: (3, 8, 2), (5, 6, 4), (6, 4, 8) and (4, 8, 3). Notice how consecutive equal values in a sequence affect whether an actual change of direction is detected by this definition. Write an **iterative** function that takes in a pointer to the front/head of a linked list, ***front***, and returns the number of changes of direction, as defined above, in the sequence of integers in the linked list pointed to by ***front***. Note: lists of size 0, 1 or 2, by definition, have no changes of direction.

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

int numDirChange(node *front) {

    // Grading: 3 pts for these initial cases.
    if (front == NULL || front->next == NULL || front->next->next == NULL)
        return 0;

    // Grading: 1 pt to set up some storage before loop.
    node* a = front;
    node* b = a->next;
    node* c = b->next;
    int res = 0;

    // Grading: Loop mechanics - 2 pts
    while (c != NULL) {

        // 2 pts for this check.
        if (a->data > b->data && b->data < c->data) res++;

        // 2 pts for this check.
        if (a->data < b->data && b->data > c->data) res++;

        // This is part of the loop mechanics.
        a = b;
        b = c;
        c = c->next;
    }

    // Grading - 0 pts, being nice here if someone forgets this, since
    // points are all allocated.
    return res;



}
```

3) (10 pts) DSN (Queues)

A regular queue only supports adding an item to the back of the queue and removing the item at the front/head of the queue. A common method of implementing a queue is as a linked list with pointers to both the front/head and back/tail of the list. If this method is used, it's relatively easy to add the functionality of adding an item to the front/head of the queue and removing an item from the back/tail of the queue. Given below is a partial implementation of this data structure (commonly called a deque). Fill in the function that adds an item to the front/head of the queue. You may call the makeNode function, and you may assume that myDeque points to a deque that exists (though it may or may not be empty.)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int data;
    struct node* next;
} node;
typedef struct deque {
    node* front;
    node* back;
} deque;

deque* makeEmptyDeque() {
    deque* tmp = malloc(sizeof(deque));
    tmp->front = tmp->back = NULL;
    return tmp;
}

node* makeNode(int val) {
    node* tmp = malloc(sizeof(node));
    tmp->data = val;
    tmp->next = NULL;
    return tmp;
}

void addFront(deque* myDeque, int val) {


    node* tmp = makeNode(val);              // Grading: 2 pts

    if (myDeque->front == NULL) {           // Grading: 1 pt
        myDeque->front = tmp;               //          1 pt
        myDeque->back = tmp;                //          1 pt
        return;                             //          1 pt
    }

    tmp->next = myDeque->front;             // Grading: 2 pts
    myDeque->front = tmp;                   //          2 pts




}
```

Computer Science Foundation Exam

May 20, 2023

Section B

ADVANCED DATA STRUCTURES

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

SOLUTION

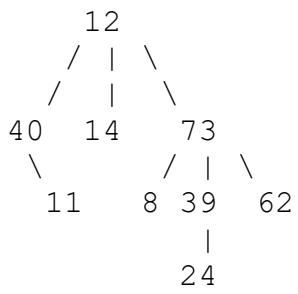
Question #	Max Pts	Category	Score
1	5	ALG	
2	10	ALG	
3	10	ALG	
TOTAL	25		

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 <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)

Consider the following tree, which is a *ternary* tree (i.e., a tree where each node can have up to three children: a *left child*, a *middle child*, and a *right child*):



Give the preorder and postorder traversals of that tree. Follow the same general algorithms you use for giving the preorder and postorder traversals of a binary tree, but extend those ideas to work for a ternary tree without upending the fundamental principles behind those traversal algorithms.

Preorder traversal: 12 40 11 14 73 8 39 24 62

Postorder traversal: 11 40 14 8 24 39 62 73 12

Grading: **2 pts for correct pretraversal, 1 pt if > ½ items are in correct slots, 0 otherwise**
2 pts for correct posttraversal, 1 pt if > ½ items are in correct slots, 0 otherwise
1 pt added if both answers are correct.

2) (10 pts) ALG (Hash Tables)

Suppose we insert nine strings into a hash table and end up with the following:

lime		kiwi	peach		lemon	coconut		mango	orange	apple		banana
0	1	2	3	4	5	6	7	8	9	10	11	12

Furthermore, suppose we no longer know the order in which those strings were inserted or what their hash values were.

In solving this problem, you may assume we used quadratic probing for the insertions. You should also assume that no strings have been deleted from the table and that each of these strings was inserted exactly once (no more, no less). You may also assume the table length was 13 for all insertions (i.e., none of the insertions triggered an expansion of the hash table).

Fill in **the 10 blanks below** to indicate whether the hash value for banana, when modded by the table length (13), might have been the value indicated. For example, it's impossible that hash("banana")%13 could have been 1, because if it were, then banana would be located in position 1 (which is empty now) and not position 12. But, it is possible that that hash("banana")%13 equals 12, because that's where it ended up.

0 NO

1 **NO**

2 NO

3 NO

4 **NO**

5 NO

6 NO

7 NO

8 YES

9 YES

10 NO

11 NO

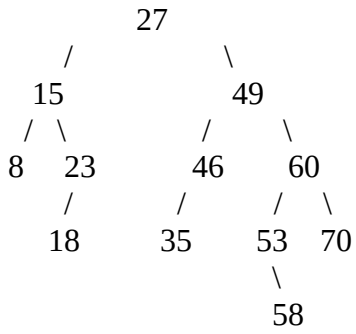
12 **YES**

Fill in each of these blanks with "YES" or "NO." Do not leave any blank.

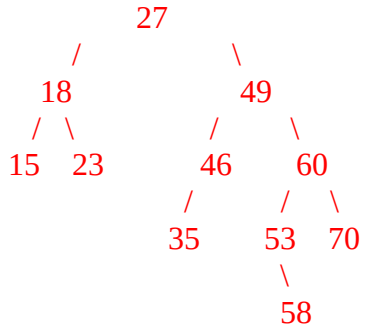
Grading: 2 pts for having all NOs correct (0 if any of these say yes)
3 pts for index 8 saying yes (0 if it says no)
5 pts for index 9 saying yes (0 if it says no)
Possible scores are 0, 2, 3, 5, 7, 8 or 10.

3) (10 pts) ALG (AVL Trees)

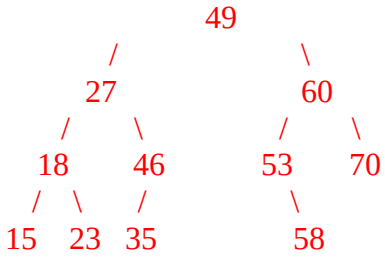
Show the result of deleting the value 8 form the AVL Tree pictured below. In this process, there are two rebalance operations that occur, total. Please show the state of the tree after each rebalance operation. (Note: after the first rebalance operation, the tree, as a whole, will not be balanced.)



Picture after first rebalance



Picture after second(last) rebalance



Grading: **4 pts first picture (3 pts for left side, 1 pt for keep right side untouched)**
6 pts second picture (1 pt – 49 root, 1 pt – 27 left child, 1 pt – 60 right child
3 pts rest of the nodes, decide partial as you see fit)

Computer Science Foundation Exam

May 20, 2023

Section C

ALGORITHM ANALYSIS

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

SOLUTION

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 Big-Oh memory usage for the function call `createNode(N)`? Please provide your answer in terms of the input parameter, N. **Please justify your answer by either evaluating an appropriate recurrence relation or summation.**

```
typedef struct Node Node;
struct Node {
   Node ** children;
   int val;
};
Node * createNode(int N) {
   Node * res = (Node *) malloc(sizeof(Node));
   if (N == 0) return res;
   res->children = (Node **) malloc(sizeof(Node*) * N);
   res->children[0] = createNode(N / 2);
   res->val = 0;
   for (int i = 0; i < N; i++)
      res->val += i;

   return res;
}
```

The amount of memory produced by the function ignoring any recursive call is O(N). Taking into account the recursive call we find that the memory created fits the following recurrence relation

T(N) = T(N/2) + O(N)
T(1) = O(1)

Solving the recurrence relation we get the following,

T(N) = T(N/2) + N
T(N/2) = T(N/4) + N/2
T(N) = T(N/4) + N/2 + N
T(N/4) = T(N/8) + N/4
T(N) = T(N/8) + N/4 + N/2 + N

after *k* iterations

$$T(N) = T\left(\frac{N}{2^k}\right) + \sum_{i=0}^{k-1} \frac{N}{2^i} \leq T(1) + N\left(\frac{1}{1-\frac{1}{2}}\right) = 1 + 2N = O(N)$$

Alternatively, a student could recognize that at each level half as much memory will be allocated, so that ultimately, the amount of memory allocated will be roughly N + N/2 + N/4 + …, and solve the sum.

Grading: 8 pts to get to the correct sum (any way), 2 pts to evaluate it and give the correct answer. If the sum is incorrect, then max credit is 7 pts. Give partial based on work and breakdown of # of nodes created. If recurrence relation method is followed, 4 pts for stating the recurrence, 2 pts for iteration, 2 pts for its solution.

2) (5 pts) ANL (Algorithm Analysis)

An $O(N^2)$ sorting algorithm took 500ms to sort an array of size 5,000. How many values can the same algorithm sort in 8 seconds?

Let T(n) represent the run time of the sorting algorithm. Then there exists a value c such that $T(n) = cn^2$.

$$T(5000) = c \times (5000)^2 = 500ms$$

$$c = \frac{500ms}{(5000)^2}$$ **Grading 3pts**

Let n be the answer to the query, then we have, remembering to convert 8 seconds to milliseconds:

$$T(n) = \frac{500ms}{(5000)^2} \times n^2 = 8000ms$$
$$n^2 = \frac{8000ms}{500ms} \times (5000)^2$$
$$n^2 = 16 \times (5000)^2$$
$$n^2 = 4^2 \times (5000)^2$$
$$n = 4 \times 5000 = \mathbf{20000}$$

Grading 2pts

3) (10 pts) ANL (Recurrence Relations)

Use the iteration technique to determine an **exact closed-form solution** for the recurrence relation, T(N), described below. **(Note: Be very careful with what occurs towards the end of the iteration, in the general case.)**

$$T(N) = (N + 1)T(N - 1) \text{ (for N > 1)}$$
$$T(1) = 1$$

$T(N) = (N + 1)T(N - 1)$ **Grading: 1pt**
$T(N - 1) = (N - 1 + 1)T(N - 1 - 1)$
$T(N - 1) = NT(N - 2)$

$T(N) = (N + 1)(N)T(N - 2)$ **Grading: 2pts**

$T(N - 2) = (N - 2 + 1)T(N - 2 - 1)$
$T(N - 2) = (N - 1)T(N - 3)$
$T(N) = (N + 1)(N)(N - 1)T(N - 3)$ **Grading: 2pts**

General Form after *k* iterations
$T(N) = T(N - k) \prod_{i=1}^{k}(N + 2 - i)$ **Grading: 2pts**
The recursion stops when N-k = 1; k = N-1

Plugging in *N* for *k* we get
$T(N) = T(1) \prod_{i=1}^{N-1}(N + 2 - i)$ **Grading: 1pt**

$T(N) = 1\ (N + 1)(N)(N - 1)\ldots(3)$
$T(N) = (\boldsymbol{N + 1})!/\boldsymbol{2}$ **Grading: 2 pts**

Computer Science Foundation Exam

May 20, 2023

Section D

ALGORITHMS

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

SOLUTION

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**, **`countDown`**, which takes in a single positive integer, ***n***, and prints each of the integers, from ***n*** down to 1, with one integer per line. For example, the function call `countdown(4)` should produce the following output:

```
4
3
2
1
```

Complete the function below:

```
void countDown(int n) {

    if (n > 0) {                    // Grading 1 pt
        printf("%d\n", n);          // Grading 2 pts
        countDown(n-1);             // Grading 2 pts
    }

}
```

2) (10 pts) ALG (Sorting)

(a) (6 pts) Consider the array below passed to merge sort. After dividing the full array into the smallest pieces, it will start calling the merge operation. In total, how many times will the merge function will be called for the following array? Show the content of the array right before the last merge operation.

17	12	11	14	15	16	10	13

Number of merge function calls: **7** **(Grading 2 pts, 1 pt for 6 or 8, 0 otherwise)**

Contents of the array right before the last merge function call:

index	0	1	2	3	4	5	6	7
data	**11**	**12**	**14**	**17**	**10**	**13**	**15**	**16**

Grading: 2 pts left half, 2 pts right half, only give the 2 pts if the half is completely correct.

(b) (4 pts) List the Big-Oh run-times requested for the following sorting algorithms, in terms of ***n***, the number of items being sorted:

Worst case run-time of quick sort : **$O(n^2)$**

Average case run-time of quick sort : **O(nlgn)**

Worst case run-time of merge sort : **O(nlgn)**

Average case run-time of merge sort : **O(nlgn)**

Grading: 1 pt for each

3) (10 pts) DSN (Bitwise Operators)

An organization has 30 groups of employees, labeled as group 0, 1, 2, …, 29. Each individual employee is assigned to some subset of those groups. The set of groups to which an employee belongs can be stored in a single integer, called the employee's ACCESS CODE, based on the bits of that integer. For example, an employee in groups 0, 3, 13 and 18 would have ACCESS CODE $2^0 + 2^3 + 2^{13} + 2^{18}$ (this is equal to 270345.) There are several shared drives at the organization. Each shared drive is accessible by any employees in a specified set of employee groups. The ACCESS CODE of a drive is specified exactly as that of an employee. If all employees who belong to either groups 2, 3 or 6 should have access to a shared drive, then that drive's access code is $2^2 + 2^3 + 2^6$ (76). An employee with ACCESS CODE 270345 would have access to a drive with ACCESS CODE 76, since the employee is part of group 3, and all employees in group 3 get access to the drive. Write a function that takes in an employee's access code, `empCode`, (as a single integer), an array of integers (`driveCodes`) storing the access codes of every shared drive in the organization, and the length of that array (`numDrives`), and returns the number of the shared drives that the employee with the given access code has access to.

```
int numDrivesAccess(int empCode, int* driveCodes, int numDrives) {


    int res = 0;                                        // Grading: 1 pt
    for (int i=0; i<numDrives; i++)                     // Grading: 2 pts
        if ( (empCode & driveCodes[i]) != 0)            // Grading: 5 pts
            res++;                                      // Grading: 1 pt

    return res;                                         // Grading: 1 pt

}
```