Computer Science Foundation Exam

May 18, 2024

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	ALG	
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)

Given the following C code.

```
int **arr1 = malloc(3 * sizeof(int *));
for(int i = 0; i < 3; ++i)
     arr1[i] = malloc(2 * sizeof(int));
int *arr2 = malloc(3 * 2 * sizeof(int));
```

Answer the following questions about the above lines of code.

a) Does `arr1` and `arr2` require the same number of total bytes allocated to be stored in the heap space? Please write yes or no. No reason is needed.

No. **Grading: 1 pt All or nothing.**

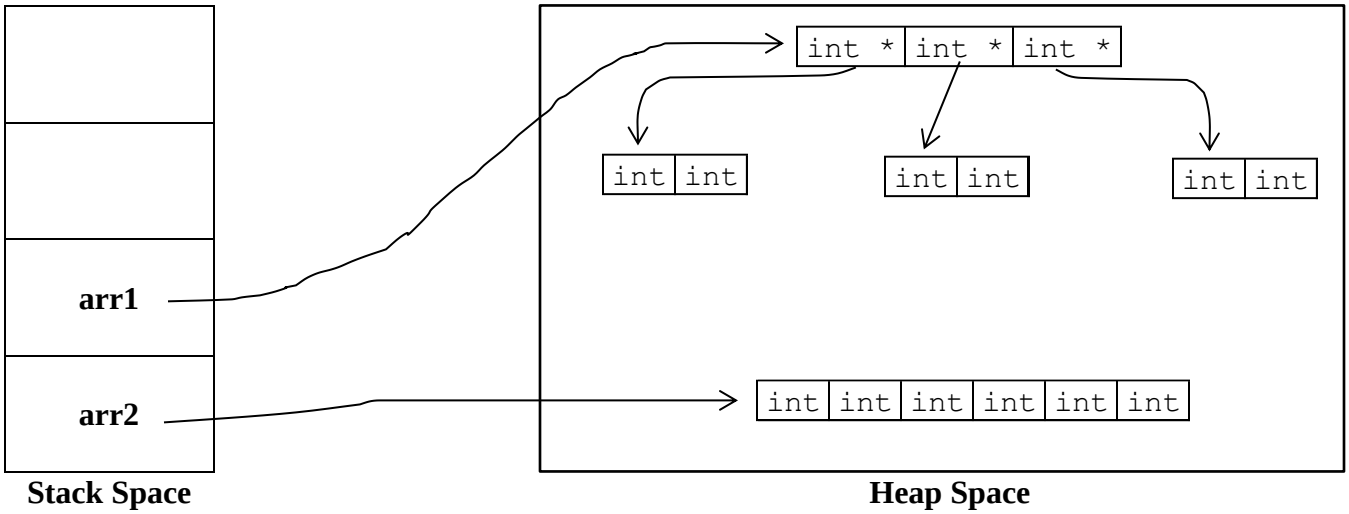
b) Are **all** the addresses associated with `arr1` (excluding `arr1` itself on the stack space) adjacent in memory? Please write yes or no only. No reason is needed.

No. **Grading: 1 pt All or nothing.**

c) Are **all** the addresses associated with `arr2` (excluding `arr2` itself on the stack space) adjacent in memory? Please write yes or no only. No reason is needed.

Yes. **Grading: 1 pt All or nothing.**

d) Complete the following stack and heap space visual below showing how the memory state looks for both `arr1` and `arr2` from the above lines of code (after all lines execute properly).



Grading: 2 pts – 1 pt for Stack (can be either order), 1 pt for heap space

2) (10 pts) DSN (Linked Lists)

Given a singly integer linked list, complete the following user defined function definition `moveHeadNearTail`. The user defined function moves the head node of some singly linked list that is passed to the second last position of the list (the node that comes before the tail node itself). The following figure shows a sample scenario. The function returns the head of the modified linked list. **You may assume the linked list pointed to by head has at least 3 elements in it.**

Before moveHeadNearTail

After moveHeadNearTail

```
typedef struct node_s {
    int data;
    struct node_s* next;
} node_t;

node_t * moveHeadNearTail(node_t * head) {

    node_t* tmp = head;                  // 1 pt
    while (tmp->next->next != NULL)      // 2 pts
        tmp = tmp->next;                 // 1 pt

    node_t* newfront = head->next;       // 1 pt
    head->next = tmp->next;              // 2 pts

    tmp->next = head;                    // 2 pts

    return newfront;                     // 1 pt
}
```

Grading: 4 pts for putting a temp pointer at the second to last node.
2 pts for storing and returning the new front
2 pts for linking first node next to last node
2 pts for linking second to last node to first node

Note: There were quite a few creative (correct) solutions significantly different than this one that were submitted by students during the exam. A couple of these techniques were:

(a) Storing the first value in the list in a temporary variable and copying the 2nd value into the 1st node, 3rd value into the 2nd node, etc, until getting to the second to last place and copying the temp variable into that node.

(b) Recursively moving the first node to the second slot (swapping the position of the nodes) until it's the second to last node in the list. The method has to return the new front of the list to fully work.

3) (10 pts) DSN (Stacks)

You are playing a scoring game that uses a LIFO approach for keeping track of scores. Here are how the scores are managed. You are given a character array (string) of moves where each index represents some rule for managing the score. You must go through the array in index order to properly manage the score.

Here are the rules for managing the score:

- If the move is some character representing an integer (0-9 both inclusive), record the integer itself.
- If the move is the character '+'. You will need to retrieve the last 2 scores recorded and compute the sum. After computing the sum, you will need to add this sum to the recorded list of scores.

Once all moves have been processed, you will need to compute the total sum of all the scores and return this value. For example, if the string passed to the function is "25+3++1", then after processing the first plus sign the corresponding stack of values from bottom to top would be [2, 5, 7]. After processing the second plus sign the corresponding stack of values from bottom to top would be [2, 5, 7, 3, 10]. After processing the string completely, the stack would store [2, 5, 7, 3, 10, 13, 1]. The sum of these values, 41, should be returned. Complete the following function definition that simulates this scoring game. You may assume that the string header file is included. The provided functions and stack structure are here to assist you with completing this function. **Note: There exists a solution that doesn't use the stack and this or any such solution will get full credit, if correctly implemented.** The parameter represents the character array of moves, and is guaranteed to be valid. Namely, the string will consist solely of digits and plus signs, and if the string has any plus signs, they will only appear in an index 2 or greater (meaning that there will be two previous scores to add.)

This code is fairly long, so go ahead and write your code on the following page. The structs and functions you may use are listed on this page, below:

```
typedef struct node_s {
      int data;
      struct node_s * next;
} node_t;
typedef struct {
      node_t * top;
} stack_t;

// Initializes a stack to be empty.
void init(stact_t* s);

// Pushes data onto the stack pointed to by s.
void push(stack_t* s, int data);

// Removes and returns the integer at the top of the stack pointed to by s.
int pop(stack_t* s);

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

```
// Stack solution.
int computeScore(char * moves) {

    int len = strlen(moves);              // 1 pt total for all init
    stack_t mys;
    init(&mys);

    for (int i=0; i<len; i++) {           // 1 pt for loop
        if (moves[i] == '+') {            // 1 pt check +
            int v2 = pop(&mys);           // 1 pt both pops.
            int v1 = pop(&mys);
            push(&mys, v1);               // 1 pt both pushes, don't
            push(&mys, v2);               // worry about order.
            push(&mys, v1+v2);            // 1 pt this must be last.
        }
        else
            push(&mys, moves[i]-'0');     // 2 pts push number
    }

    int res = 0;                          // 2 pts for this whole
    for (int i=0; i<len; i++)             // part. 1 pt if some
        res += pop(&mys);                 // error but right idea
    return res;
}
```

Note: The could also malloc a stack_t pointer in which case the function calls would look different; just make sure all of that is consistent.

```
// Array solution.
int computeScoreAlt(char * moves) {

    int len = strlen(moves);                  // 1 pt
    int* tmp = malloc(sizeof(int)*len);       // 1 pt
    int res = 0;                              // 1 pt
    for (int i=0; i<len; i++) {               // 1 pt
        if (moves[i] == '+')                  // 1 pt
            tmp[i] = tmp[i-1] + tmp[i-2];     // 1 pt
        else
            tmp[i] = moves[i] - '0';          // 1 pt
        res += tmp[i];                        // 1 pt
    }
    free(tmp);                                // 1 pt
    return res;                               // 1 pt
}
```

Note: If you're clever, the array isn't necessary, you just have to keep the two previous values and update them each loop iteration. (The array makes is easier to write bug free code at the cost of more memory.)

Computer Science Foundation Exam

May 18, 2024

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	DSN	
3	10	DSN	
TOTAL	25	----	

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

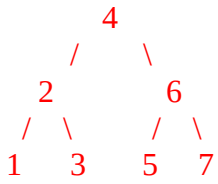
Problems will be graded based on the completeness of the solution steps and <u>not</u> graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all <u>be neat</u>. For each coding question, assume that all of the necessary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

1) (5 pts) ALG (Binary Trees)

a) Draw **a binary search tree** of with 5 nodes (storing positive integers) that has the maximum possible height. (1 pt)

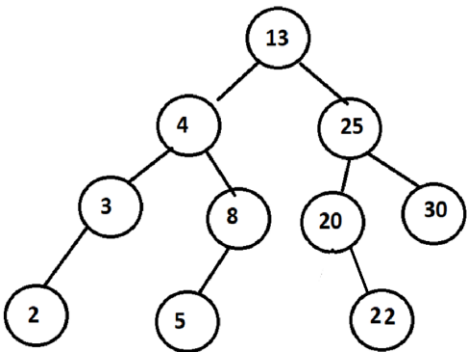
Grading: There are 16 possible structures give 1 point for any correct answer 0 otherwise. (For each item, you may choose the smallest or largest unused item, so you have 2 choices when placing each item but the last.)

b) Draw another binary search tree with 7 nodes (storing positive integers) that has the minimum possible height. (1 pt)

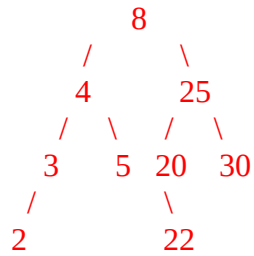


Grading: only one possible structure works here, give 1 point if their answer is a correct one, 0 otherwise.

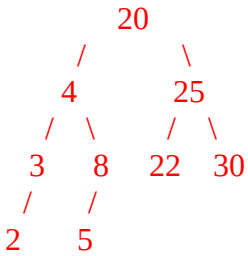
c) Re-draw the following binary search tree after deleting the root node. (3 pts)



There are two possible answers: **(Grading: 1 pt root, 1 pt untouched side, 1 pt adjusted side Automatic 0 out of 3 if not a valid Binary Search Tree.)**



OR



2) (10 pts) DSN (Heaps)

a) (6 pts) Consider the following struct that represents a binary minheap.

```
typedef struct heap {
    int* elements; //points to the array of heap elements
    int capacity; // total size of the array
    int size;     // actual number of elements in the heap
} heapStruct;
```

Also, the following functions are available to you, and you are free to call them as needed:

- `int removeMin(heapStruct *h);//removes the smallest item from the heap pointed to by h.`
- `int size(heapStruct* h); // returns the number of elements in the heap pointed to by h.`

Write a function called heapsort that takes a pointer to a heap, and returns those values in a sorted integer array. At the end of the function, the heap pointed to by h will be empty.

```
int* heapsort(heapStruct* h) { //complete this function

    int n = size(h);                          // 1 pt
    int* res = malloc(n*sizeof(int));         // 1 pt
    for (int i=0; i<n; i++)                   // 1 pt
        res[i] = removeMin(h);                // 2 pts
    return res;                               // 1 pt
}
```

Note: Students can access the size directly via h->size so they could use a while loop (while (h->size > 0)). Since removeMin adjusts the size of the heap, separately changing this variable in addition to the function call is incorrect.

b) (4 pts) Specify the worst run-time when efficiently implemented for the following operations:

Operation	**Run-time**
Building a binary heap from an unordered array of size **n** using heapify	O(**n**)
Inserting an item into a binary heap with **n** items.	O(**lg n**)
Deleting the minimum item from a binary heap with **n** items	O(**lg n**)
Heapsort of **n** items.	O(**nlg n**)

3) (10 pts) DSN (Tries)

As an afficionado of Wordle, you're curious how many five letter words there are in a dictionary stored in a trie. Write a recursive function that takes in a pointer to a trie node and an integer k, representing the depth of the node in the trie, and **returns the number of five letter words** stored within that subtrie. A wrapper function is provided which makes the initial recursive call on the root node of the trie storing the dictionary. Please use the struct shown below. Assume all necessary includes.

```
typedef struct trieNode {
    int isWord;
    struct trieNode* children[26];
} trieNode;

int num5LetterWrapper(trieNode* root) {
    return num5Rec(root, 0);
}

int num5Rec(trieNode* root, int k) {

    if (root == NULL) return 0;                    // 2 pts
    if (k == 5) return root->isWord;               // 2 pts

    int res = 0;                                   // 1 pt
    for (int i=0; i<26; i++)                       // 1 pt
        res += num5Rec(root->children[i], k+1);    // 3 pts
    return res;                                    // 1 pt
}
```

Note: The NULL check can be done in the for loop instead, but either way, it's 2 pts to avoid a NULL ptr error.

Computer Science Foundation Exam

May 18, 2024

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	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 <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) ANL (Algorithm Analysis)

What is the **worst-case** Big O runtime for the following function, in terms of the input parameter, **n**? (You may assume that the array pointed to by list is of length **n**.) In order to receive full credit, you must use words to explain your reasoning AND arrive at the correct answer.

```
int mystery(int* list, int n) {

    int i = 0, j = 1;

    if (n < 2) return 0;

    while (j < n) {

        while (i < j && list[i] < list[j]) i++;
        j++;
    }

    return i;
}
```

REASON:

It's impossible for either i or j to exceed n, since i will never get greater than j and j will never get greater than n. There's O(1) extra work before either i or j are incremented. This means that the inner while loop never runs more than n times **total** and the outer while loop never runs more than n times **total**. In fact, structurally, as long as n >= 2, the outer while loop runs precisely n – 1 times. It follows that the runtime of this code is simply **O(n).**

RUN-TIME: (n)

Grading: **1 pt for the correct answer**
4 pts for the reason: 1 pt to argue that outer loop runs <= n times.
3 pts to argue that the inner loop never runs more than n times.

If the answer is wrong, max score is 1/5 for arguing that the outer loop runs no more than n times.

2) (10 pts) ANL (Algorithm Analysis)

An algorithm that processes a list of size n takes $O(\sqrt{n}lgn)$ time. On Shannon's computer, when she runs the algorithm on a list of size **n = 2^{16},** her computer takes **c** milliseconds. (Shannon is very secretive, so she hasn't told you the value of **c** unfortunately!) **In terms of c,** how long, **in milliseconds,** should we expect the algorithm to take on her computer when she is processing a list of size 2^{20}? (Your answer should be of the form kc, where k is a positive real number.)

Let T(n) = $d\sqrt{n}lgn$, be the run time of the algorithm on a list of size n, where d is some constant, different than the variable c mentioned in the problem. Using the given information we have:

$$c = T(2^{16}) = d\left(\sqrt{2^{16}}\right)(\lg 2^{16}) = (2^8)(16lg2)d = (2^{12}lg2)d$$

We seek the value of T(2^{20}):

$$T(2^{20}) = d\left(\sqrt{2^{20}}\right)(\lg 2^{20}) = (2^{10})(20lg2)d = 5(2^{12}lg2)d = \mathbf{5}\boldsymbol{c}$$

It follows that the desired answer is 5c. (Shannon should expect this list to take about five times longer than the previous list.)

Note: Notice that the value of d cancels out, so we can simply calculate the ratio of T(2^{20})/T(2^{16}) and this will reveal the value of 5 as the multiplicative factor.

Grading: 1 pt for plugging in 2^{16} into the Big-Oh function,
3 pts for simplifying it to $2^{12}lg2$, or another suitable form.
1 pt for plugging in 2^{20} into the Big-Oh function,
3 pts for simplifying it to $(2^{10})(20lg2)$, or another suitable form.
2 pts for doing substitution or division or anything equivalent to arrive at the answer 5c.

3) (10 pts) ANL (Recurrence Relations)

Using the iteration technique, determine the Big-Oh solution to the recurrence relation below, in terms of ***n***.

$$T(n) = 2T\left(\frac{n}{2}\right) + n^3, \text{ for n > 1}$$
$$T(1) = 1$$

Let's work out three iterations:

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

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

$T(n) = 4T\left(\frac{n}{4}\right) + n^3 + \frac{n^3}{4}$ (Iteration 2 – some students might take more steps to get here)

$T(n) = 4(2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^3) + n^3 + \frac{n^3}{4}$

$T(n) = 8T\left(\frac{n}{8}\right) + (n^3 + \frac{n^3}{4} + \frac{n^3}{16})$ (Iteration 3)

After k iterations, we have the following:

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \frac{n^3}{4^i}$

Plug in $\frac{n}{2^k} = 1$, $n = 2^k$ and substitute:

$$T(n) \leq nT(1) + n^3 \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = n + n^3\left(\frac{1}{1-\frac{1}{4}}\right) = n + \frac{4}{3}n^3 = O(n^3)$$

Grading:
1 pt first iteration
2 pts second iteration
2 pts third iteration
2 pts guess
1 pt substitute n = 2^k
2 pts work to final answer

Computer Science Foundation Exam

May 18, 2024

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	10	DSN	
2	10	DSN	
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.

1) (10 pts) DSN (Recursive Coding)

Write a recursive function that determines if player X, who goes first, can win a game of tic-tac-toe (played on a 3 by 3 int board). You can use the following helper functions. You don't have to implement any of the listed helper functions. **Functions that do not use recursion will receive 0 points**. Note: board is a 3 by 3 array of integers, storing either EMPTY(0), X(1), or O(2). The three given functions return the current state of the board, as it is, from player X's perspective, not what "could happen" in the future. myTurn is 1, when it's X's turn and 0 when it's O's turn. You will have to place and unplace X's and O's in your solution. Finally, the intention here is that player 'O' plays pessimistic. Namely, if there is ANY set of moves that players X and O could make from the current state of the board that result in X winning, the function should return 1.

```
#define EMPTY 0
#define X 1
#define O 2

int XWin(int ** board); // returns 1 if I have won and 0 otherwise
int XLose(int ** board); // returns 1 if I have lost and 0 otherwise
int tied(int ** board); // returns 1 if the board is in a tied state

int canXWin(int ** board, int myTurn) {

   // (+1 pt) for returning the correct answer
   // in each terminating case (lose, tie, win)
   if (tied(board) || XLose(board)) return 0;
   if (XWin(board)) return 1;

   // (+1 pts) for trying all spots
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         // (+1 pt) for checking if spot can be used
         if (board[i][j] == EMPTY) {

            // (+1 pt) for setting to (+1 pt) appropriate value
            board[i][j] = myTurn ? X : O;

            // (+1 pt) if check if winning spot recursively
            if (canXWin(board, !myTurn)) return 1;

            // 1 pt for undoing the piece placed.
            board[i][j] = EMPTY;
         }
      }
   }

   return 0; // (+1 pt) if no win found

}
```

2) (10 pts) ALG (Sorting)

(a) (1 pt) Which of the sorting algorithms (listed in part d) could encounter problems if an array can contain duplicates? (Specifically, for four of the algorithms, whether or not there are duplicates in the array don't alter the run-time of the algorithm on individual cases, but one of the algorithms, in its original form, is definitively affected.)

Quick Sort (**Grading: 1 pt all or nothing)**

(b) (2 pts) What problem could be encountered?

If all of the numbers are the same, the partition element will either end up being the very first element or the very last element, always creating a recursive call on an array of size n – 1, if the previous call was on an array of size n. This results in an $O(n^2)$ run-time.

(c) (2 pts) Pick one of the algorithms that aren't affected by duplicates and explain why it runs similarly with or without duplicates.

Bubble: Regardless of what values are in the array, the sort always makes the same number of loop iterations, so duplicates don't affect the run-time at all.

Insertion: If we only continue our inner loop if the value being inserted is less than the value it's compared to, then equal values would not result in the inner loop running longer. Instead, a full array of equal values would take n steps to sort, since the inner loop would never repeat.

Merge: The merge algorithm does the same number of comparisons and copies regardless of the actual values in the two subarrays being merged. The Merge Sort always calls the Merge on arrays of the same sizes, so duplicates do not affect the run-time at all.

Selection: The loop structure of selection sort is similar to Bubble Sort and guaranteed not to change, regardless of the values, thus duplicates won't affect the run time of Selection Sort. (First time, loop runs through n values, then n – 1, then n – 2 and so forth.)

Grading: Give full credit if the choice of algorithm is clear and the reason is accurate. Give 1 point if you feel the answer is worth some credit but has some non-trivial ambiguities or holes.

(d) (5 pts) What is the worst case runtime for the following sorting algorithms on an array with ***n*** distinct values? Please list your answers with Big-Oh notation, using proper conventions.

Quick **$O(n^2)$**

Bubble **$O(n^2)$**

Insertion **$O(n^2)$**

Merge **$O(nlgn)$**

Selection **$O(n^2)$**

Grading: 1 pt per each one, don't give credit if there are leading constants

3) (5 pts) DSN (Base Conversion)

Convert 277 in base 8 to base 16. **Please show your work and put a box around your final answer.**

Convert 277 to binary as follows, substituting each octal value for its corresponding binary value in 3 bits, since $2^3 = 8$.

```
010 111 111
```

Now, regroup these bits by groups of 4, from the right:

```
0 1011 1111
```

Finally, convert each group of 4 bits to Hexadecimal, omitting leading 0s:

BF

Grading: 1 pt - if they tried to convert to decimal or binary
2 pts - if they converted to decimal or binary correctly
2 pts - for the correct answer

Note: Here is the conversion to through base 10:

$277_8 = 2 \times 8^2 + 7 \times 8 + 7 = 128 + 56 + 7 = 191$

16 | 191
16 | 11 R 15

11 = B, 15 = F, so the final answer is **BF.**