Computer Science Foundation Exam

May 21, 2022

Section A

BASIC DATA STRUCTURES

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 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 (Dynamic Memory Management in C)

The following function has 5 memory management issues, each one occurring on a different one of the 17 labeled lines of code. Please clearly list which five lines of code have the errors on the slots provided below. **Please list exactly five unique line numbers in between 1 and 17, inclusive. An automatic grade of 0 will be given to anyone who lists MORE than 5 line numbers.**

```
int n = 10;                                    // line 1
int *p1, *p2, *p3, **p4;                       // line 2
char str1[100] = "test string";                // line 3
char *str2;                                    // line 4
strcpy(str2, str1);                            // line 5
p1 = (int *)malloc(n * sizeof(int));           // line 6
p2 = (int *)malloc(n * sizeof(int));           // line 7
for(int i=0; i<n; i++)                         // line 8
     p1[i] = rand()%100;                       // line 9
p2 = p1;                                       // line 10
*p3 = 50;                                      // line 11
p4 = (int **) malloc(n * sizeof(int*));        // line 12
for(int i=0; i<n; i++)                         // line 13
     p4[i] = -5;                               // line 14
free(p1);                                      // line 15
free(p2);                                      // line 16
free(p4);                                      // line 17
```

Lines with Memory Management Errors: 5, 10, 11, 14, 16

Grading: 1 pt for each correct answer on a slot. 0 automatically if more than 5 items are listed.

2) (10 pts) DSN (Linked Lists)

The structure of each node of a singly linked list is shown below.

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

Write a function insertAfterN, that takes the head of a linked list, and two integers M and N (M ≠ N) and inserts M after all the nodes containing N.

For example, if M = 200 and N = 6, the linked list 3, 6, 4, 6, 6, 5 will be changed to 3, 6, 200, 4, 6, 200, 6, 200, 5.

```
void insertAfterN(node* head, int M, int N) {

    if (head == NULL) return;

    if (head->data == N) {
        node* tmp = malloc(sizeof(node));
        tmp->data = M;
        tmp->next = head->next;
        head->next = tmp;
        head = tmp;
    }

    insertAfterN(head->next, M, N);

}
```

Grading:

4 pts for "looping" mechanism. (In this solution, that means both the base case, the recursive call, and making sure head->next is the next node. It's also okay if head=tmp; is not included.) An iterative solution would include a for or while loop with head advancing.

1 pt for checking if the data value in a node equals N

2 pts malloc new node in the appropriate case (1 pt malloc, 1 pt parameter and assignment)
1 pt store M in data field of new node
1 pt linking new node to the old next of the list
1 pt linking node storing N to node storing M

3) (10 pts) DSN (Stacks)

A word is considered a palindrome if the reverse of the word is the same as the original word. For example: the word “test” is not a palindrome as its reverse “tset” is not the same as “test”. On the other hand, the word “racecar” is a palindrome as its reverse is the same as “racecar”. Some other examples of palindromes are “hannah”, “level”, “madam”, and “yay.”

Write a function that will take a string in the parameter and returns 1, if the string is a palindrome, otherwise returns 0. **You have to use stack operations during this process.** (Credit isn’t awarded for correctly solving the problem, but for utilizing the stack in doing so.)

Assume the following stack definition and the functions already available to you. The stack will be extended automatically if it gets full (so you, don’t have to worry about it). The top of the stack is controlled by your push and pop operation as usual stack operations.

```
void initialize(stack* s); // initializes an empty stack.
int push(stack* s, char value); //pushes the char value to the stack
int isEmpty(stack* s); // Returns 1 if the stack is empty, 0 otherwise.
char pop(stack* s); // pops and returns character at the top of the stack.
char peek(stack* s); // returns character at the top of the stack.
```

Note: pop and peek return 'I' if the stack s is empty.

```
int isPalindrome(char *str) {

    struct stack s;
    initialize(&s);
    int len = strlen(str);

    for (int i=0; i<len/2; i++)             // Loop can go to len
        push(&s, str[i]);

    for (int i=len-len/2; i<len; i++)       // if first goes to len
        if (pop(&s) != str[i])              // this one must start
            return 0;                       // at 0.

    return 1;                               // ok to check for empty

}
```

Note: The second for-loop could be written like so: `for (int i=(len+1)/2; i<len; i++)`

Grading: 2 pts push first half,

2 pts correct # of pushes (note: if string length is odd, they can push the middle element or not; that doesn’t affect credit here, but it could affect the next 2 points below)

2 pts pop off second half (give 1 pt if off by 1) (if string length is odd and they pushed the middle element, they must also pop it here in order to get all 2 points)

1 pt return 0 as soon as error is spotted

2 pts indexing and # of pops is accurate (give 1 pt if off by 1)

1 pt return 1 at end (note stack should be empty via # of pushes/pops so no need to check)

Computer Science Foundation Exam

May 21, 2022

Section B

ADVANCED DATA STRUCTURES

SOLUTION

Question #	Max Pts	Category	Score
1	10	DSN	
2	10	ANL	
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 (Binary Trees)

Demonstrate your understanding of recursion by rewriting the following recursive function, which operates on a binary tree, as an **iterative** function. In doing so, you must abide by the following restrictions:

1. Do not use *break* statements in your solution.
2. Do not write any helper functions in your solution.
3. Do not make any recursive calls in your solution.

```
int foo(node *root) {
    if (root == NULL) return 1;
    if (root->left == NULL && root->right == NULL) return 2;
    if (root->left == NULL) return 3 * foo(root->right);
    if (root->right == NULL) return 4 * foo(root->left);
    if (root->right->data > root->left->data) return 5 * foo(root->right);
    return 6 * foo(root->left);
}

int iterative_foo(node *root) {

      int result = 1;
      node *temp = root;

      while (temp != NULL) {
            if (temp->left == NULL && temp->right == NULL) {
               result *= 2;
               temp = NULL;
            }
            else if (temp->left == NULL) {
               result *= 3;
               temp = temp->right;
            }
            else if (temp->right == NULL) {
               result *= 4;
               temp = temp->left;
            }
            else if (temp->right->data > temp->left->data) {
               result *= 5;
               temp = temp->right;
            }
            else {
               result *= 6;
               temp = temp->left;
            }
      }

      return result;
}
```

See grading notes on following page.

Grading for Question #1:

Note: The *temp* variable is not strictly necessary. We could instead use *root* to traverse the tree. Since it's just a local copy of the original *root* pointer at the start of the function, modifying it has no adverse impact here.

Note: We cannot check whether *temp == NULL* in the while loop, since it's gated by the condition that *temp != NULL*.

+1 pt for creating a result variable of some sort.
+1 pt for initializing the result variable to 1. (Initializing to 0 would wipe out results of multiplication. If they found a way around that, though, award this point.)
+1 pt for looping condition. They might instead use *while(1)* and then check whether *temp == NULL* inside the loop, returning *result* if so. That alternative approach is fine. If they forget to declare temp or initialize *temp* to *root*, or if the data type is wrong for that variable, do not award this 1 pt.
+2 pts for moving pointer left and right as appropriate in all cases.
+2 pts for multiplying result variable by appropriate integer in all cases.
+2 for overall structure and general correctness of solution (i.e., a while loop with if statements that multiply the return value by some integer and move a pointer left or right). Note that if they use a temp variable, they must be careful to actually use that in their *if* conditions (rather than *root*), as well as their assignment statements (e.g., temp = temp->right, not temp = root->right). Note also that they **must** use *else* statements.to prevent multiple *if* statements from triggering. The recursive version uses *return* statements to prevent that, but in an iterative version, we can't just slap an *else* statement on the last condition. Otherwise, we could check multiple conditions and possibly encounter a segmentation fault.
+1 for finding a way to terminate when root->left and root->right are both NULL. This could involve a return statement within the loop, or they could set their pointer to NULL.

2) (10 pts) ANL (Hash Tables)

This question asks you to explore the **best-** and **worst-case** runtimes for adding *r* new elements to a hash table that already contains *q* elements. In answering the questions below, assume the following:

1. Generating the initial hash value for any given key takes O(1) time.
2. We are using quadratic probing.
3. Our hash table is at least half empty, and the length of the table is prime.
4. There is enough space in the hash table to allow for all *r* new elements to be inserted without triggering a table expansion.
5. If the specific placement of the *q* elements that are already in the hash table is relevant, you may assume that they are placed in a way that would facilitate the situation you are describing that leads to the best- or worst-case runtime for the *r* new elements being added to the table.

Note that this question is not just asking for the runtime for adding a single element. We want the runtime for adding **all** *r* elements to the hash table. While the *q* elements already in the table may impact the runtime for adding the *r* new elements, you do not have to account for the runtime it took to add those *q* elements. Focus only on the cost of adding the *r* **additional** elements.

a. (2 pts) In big-oh notation, what is the **best-case** runtime for adding *r* new elements to the table?

O(r)

b. (1 pt) What situation leads to the best-case runtime you listed in part (a)?

We encounter no collisions.

c. (5 pts) In big-oh notation, what is the **worst-case** runtime for adding *r* new elements to the table?

O(qr + r^2) --or-- O(r(q + r))

d. (2 pts) What situation leads to the worst-case runtime you listed in part (c)?

All *r* elements map to the same initial index in the hash table, **and** the first of those elements collides with all *q* elements already in the hash table before finding an open position (meaning that all subsequent elements will collide with all *q* elements in the table and any of the *r* elements that have already been inserted). The overall number of collisions is:

$$rq + \sum_{i=0}^{r-1} i = rq + \frac{(r-1)r}{2} = rq + \frac{r^2 - r}{2} = O(rq + r^2)$$

See grading notes on following page.

Grading for Question #2:

a. +2 points for correct answer. This part is all or nothing (no partial credit).
b. +1 point for correct answer.
+5 points for $O(qr + r^2)$, $O(r(q + r))$, $O(r*\max(r,q))$
+3 points for $O(qr)$ or $O(r^2)$
+2 points for $O(qr^2)$, $O(q^2)$
+1 point for $O(r\lg r)$, or any variation with r or q with a linear and log factor, or for $O(r)$ or $O(q)$.
c. This part is worth two points total, as follows:
- +1 point for mentioning the collision with all *q* elements
- +1 for saying all *r* elements map to the same initial position in the table --or-- that each of the *r* new elements collides with all of the other *r* elements that were inserting before it.

3) (5 pts) ALG (AVL Trees)

Suppose we randomly shuffle the six words in the list below and insert them into an AVL tree. (In other words, we insert them in random order – not necessarily the order given – with each of those words ending up in the AVL tree exactly once.)

Fill in the blank next to each word to indicate whether it could **ever** end up at the root of the resulting AVL tree ("yes") or not ("no"). (If you answer "no" for a given word, you are saying it could **never** end up at the root of the resulting AVL tree.)

You may assume the AVL tree is ordered alphabetically. So, all the words in the left subtree of "apple" would have to come before "apple" in alphabetical order, and all the words in its right subtree would have to come after "apple" in alphabetical order.

```
__no____ apple

__yes___ mango

__no____ papaya

__no____ banana

__no___ mulberry

__yes_ blueberry


^
Fill in each blank with "yes" or "no" to indicate whether the word could
serve as the root of an AVL tree that results from shuffling these six words
and inserting them into an AVL tree in random order.
```

See notes on following page for guidance on how to solve this problem.

Grading for Question #3:

Subtract 1 point for each incorrect answer (including any spaces that were left blank), for a minimum score of 0 points.

(So, if they got 5 blanks right and 1 blank wrong, that's -1, for a total of 4/5 on this question. If they got 1 blank right and 5 blanks wrong, that's -5, for a final score of 0/5. If they got all 6 blanks wrong, that's also 0/5.)

<u>Notes for Solving Question #3:</u>

Probably the first thing to do is to list the words in alpha order:

```
      apple
      banana
      blueberry
      mango
      mulberry
      papaya
```

If we try to construct an AVL tree with “apple” at the root, all other words go into its right subtree. The smallest height that right subtree can have is 2, meaning we have an unbalanced root. For example:

```
      apple
          \
           mango              |
         /       \            | There are other subtrees we can make with these
      blueberry   papaya      | five words, but none of them are shorter than this.
       /          /           |
    banana     mulberry       |
```

Therefore, “apple” cannot be at the root of the AVL tree no matter what order we use to insert these words. Rotations would displace “apple” from the root. A symmetric argument can be used to show that “papaya” cannot ever be the root, since every other word would have to go into the left subtree of “papaya.”

Similarly, if we try to place “banana” at the root, its left subtree must contain “apple,” and its right subtree must contain all other words. There would be 4 words in its right subtree, with a minimum height of 2. For example:

```
      banana
      /     \
   apple     mango            |
            /    \            | There are other subtrees we can make with these
      blueberry  papaya       | four words in the right subtree of "banana," but
                  /           | none of them are shorter than this.
              mulberry        |
```

That means we necessarily have a bad balance factor at “banana” if it’s the root, and so it cannot serve as the root of an AVL tree with these words. A symmetric argument can be used to demonstrate that “mulberry” cannot be at the root, since “papaya” would go to the right of “mulberry” and all remaining words would go in its left subtree.

That leaves just “mango” and “blueberry” for us to consider. Either of them can serve as the root. E.g.:

```
            blueberry                                      mango
            /        \                                    /     \
        apple        mulberry              banana    mulberry
           \         /    \                /    \            \
         banana   mango    papaya      apple   blueberry   papaya
```

Computer Science Foundation Exam

May 21, 2022

Section C

ALGORITHM ANALYSIS

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 <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)

What is the worst case Big-Oh runtime for the function **f,** in terms of its input parameter n? You may assume that the array pointed to by arr is of length n. (Grading note: 2 pts will be awarded for the answer, 8 pts for the proof of the answer. Your proof must include either summations or recurrence relations related to the code below.)

```
int f(int* arr, int n, int minVal) {
    return fHelp(arr, 0, n-1, minVal);
}

int fHelp(int* arr, int low, int high, int minVal) {
    if (low > high) return 0;
    if (low == high) return arr[low] >= minVal;

    int mid = (low+high)/2;
    int left = fHelp(arr, low, mid, minVal);
    int right = fHelp(arr, mid+1, high, minVal);
    int res = left;
    if (right > left)
        res = right;

    int alt = 0, i;
    for (i=mid; i>=low; i--) {
        if (arr[i] < minVal) break;
        alt++;
    }
    for (i=mid+1; i<=high; i++) {
        if (arr[i] < minVal) break;
        alt++;
    }

    if (alt > res) res = alt;
    return res;
}
```

There are two recursive calls in fHelp, both to arrays of half the size of the original array. Let T(n) be the run time of function f. Effectively, T(n) breaks down into two function calls, each of which take time T(n/2), plus the work after the recursive calls. There are two loops, each which run n/2 times at most, so in total the loops run n times with constant time operations inside the loops. Thus, the total amount of work beyond the recursive calls is O(n). It follows that T(n) satisfies the following recurrence relation:

$T(n) = 2T(n/2) + O(n)$.

We can solve this recurrence relation via the Master Theorem, getting a solution of **O(nlgn).** (A = 2, B = 2 and k = 1. Since $B^k = 2$ and A = 2, the solution follows.)

Grading: 2 pts for correct answer (give pts even if no work)
2 pts for ANY recurrence relation
2 pts (additional) if recurrence has T(n/2) on RHS
2 pt (additional) if term is 2T(n/2)
2 pt for O(n) added extra work

2) (5 pts) ANL (Algorithm Analysis)

A program that runs an O(Nlog(N)) algorithm to sort an array of N polygons takes 10 seconds to sort 1,000,000 polygons. How long, **in milliseconds**, would it be expected for the program to take when sorting 1,000 polygons?

Let T(N) = cNlog(N) be the run time of the program for sorting N polygons. Using the given information, we have:

$$T(10^6) = c(10^6)\log(10^6) = 10{,}000ms$$
$$c(10^6)\,6\log(10) = 10{,}000ms$$
$$c = \frac{10^4 ms}{6(10^6)\log(10)} = \frac{1ms}{600\log(10)}$$

Now, let's solve for T(1000):

$$T(10^3) = \frac{1ms}{600\log(10)} \times 10^3 \times \log(10^3)$$
$$= \frac{1ms}{600\log(10)} \times 10^3 \times 3 \times \log(10)$$
$$= \frac{3000ms}{600} = \mathbf{5ms}$$

Grading: 1 pt set up equation with constant c
2 pts solve for c (no simplification required)
1 pt plug in N = 1000
1 pt arrive at the final answer

Full credit if the ratio method is used properly, but if it isn't max 1 point.

3) (10 pts) ANL (Recurrence Relations)

Using the iteration technique, determine a closed-form solution for the following recurrence relation in terms of n. Note: Your answer should be **EXACT** and not a Big-Oh bound.

$$T(0) = 1$$
$$T(n) = 4T(n-1) + 2^n, for\ integers\ n > 0$$

Using the iteration technique for three iterations, we get:

$T(n) = 4T(n-1) + 2^n$ **Grading: 1 pt**
$T(n) = 4(4T(n-2) + 2^{n-1}) + 2^n$
$T(n) = 16T(n-2) + (2^2)2^{n-1} + 2^n$
$T(n) = 16T(n-2) + [2^{n+1} + 2^n]$ **Grading: 2 pts**
$T(n) = 16(4T(n-3) + 2^{n-2}) + [2^{n+1} + 2^n]$
$T(n) = 64T(n-3) + (2^4)2^{n-2} + [2^{n+1} + 2^n]$
$T(n) = 64T(n-3) + [2^{n+2} + 2^{n+1} + 2^n]$ **Grading: 2 pts**

From here we see that after k iterations, the recurrence is:

$T(n) = 4^kT(n-k) + \sum_{i=0}^{k-1} 2^{n+i}$ **Grading: 2 pts**

Since we know T(0), let's plug in k = n into the formula above to yield:

$T(n) = 4^nT(n-n) + \sum_{i=0}^{n-1} 2^{n+i}$ **Grading: 1 pt**
$T(n) = 4^nT(0) + \sum_{i=0}^{n-1} 2^n2^i$
$T(n) = 2^{2n} + 2^n \sum_{i=0}^{n-1} 2^i$
$T(n) = 2^n(2^n) + 2^n \sum_{i=0}^{n-1} 2^i$
$T(n) = 2^n\left[2^n + \sum_{i=0}^{n-1} 2^i\right]$
$T(n) = 2^n\left[\sum_{i=0}^{n} 2^i\right]$
$T(n) = 2^n(\frac{2^{n+1}-1}{2-1})$
$T(n) = 2^n(2^{n+1} - 1)$ **Grading: 2 pts**

Grading Note: Please accept any reasonable final form (there are quite a few that are reasonably simplified). For the last two points, award 1 pt if there is some algebra but not quite enough to get to a reasonable final form.

Computer Science Foundation Exam

May 21, 2022

Section D

ALGORITHMS

SOLUTION

Question #	Max Pts	Category	Score
1	10	DSN	
2	10	ALG	
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 (Recursive Coding)

The Towers of Hanoi Problem involves starting with a tower of ***n*** disks (of all different radii) on a single pole and transferring those disks to a different pole. There are only 3 poles that can hold the disks. The disks can only be moved one at a time, from one pole to another, and a disk must always be the only disk on the pole or be placed on top of a disk that is larger. The initial configuration of disks starts with all of the disks sorted in order on a single pole, with the smallest disk on top. For the purposes of this problem, let the radii of the disks be 1 cm, 2 cm, 3 cm, …, ***n*** cm. Let the cost of a single move simply equal the radius of the disk being moved. Define the cost of a solution to the puzzle to be the sum of the costs of the moves to solve the puzzle. Complete the function below, ***using recursion***, so that it takes in a single integer, ***n,*** the number of disks for the puzzle, and returns the minimal cost for transferring the ***n*** disks from the starting pole to either of the other poles. (Note: the actual value of the starting and ending poles don't affect the answer, so long as they are different poles. The code is relatively short and one can do some math to get an O(1) solution, but what is being tested here is the ability to take a problem, break it down recursively, and implement that solution. Thus, the grading criteria described below is focused on rewarding the skill that's being tested, though an alternate, more efficient solution exists.)

Grading Note: Any iterative or non-recursive solution will get a maximum of 3 points, a recursive solution that takes θ(2^n) time will get a maximum of 7 points. To receive full credit, your solution must be recursive and run in O(n) time, where n represents the input value to the function.

```
int towersCost(int n) {

    // Grading: 3 pts, 1 pt if, 1 pt 1, 1 pt ret could also be
    // if (n == 0) return 0;
    if (n == 1) return 1;

    // Grading: 1 pt ret, 2 pts 2*, 2 pts rec call, 1 pt +, 1 pt n
    return 2*towersCost(n-1) + n;

}
```

2) (10 pts) ALG (Sorting)

(a) (5 pts) Consider running an Insertion 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.

35	25	15	29	39	22	19	6	21

Reasoning:

Each move is a swap, since the only moves the algorithm does are swaps.

25 moves once (since 35 is larger)
15 moves twice (since both 25, 35 are larger)
29 moves once (35 is larger)
39 doesn't move (largest)
22 moves 4 times (25, 29, 35, 39 larger)
19 moves 5 times (22, 25, 29, 35, 39 larger)
6 moves 7 times (all values larger)
21 moves 5 times (22, 25, 29, 35, 39 larger)

Total swaps = 1 + 2 + 1 + 0 + 4 + 5 + 7 + 5 = 25

Number of Swaps: **25**

Grading: 2 pts for answer, 3 pts for reasoning
If answer is off due to arithmetic error, 4/5
If answer is correct, but reasoning doesn't make sense 2/5
Incorrect answer with incorrect reasoning is either 0/5 or 1/5

(b) (5 pts) List the **average case** run time of each of the following sorting algorithms, in terms of n, the number of items being sorted. (Please provide Big-Oh bounds.)

(i) Insertion Sort	$O(n^2)$
(ii) Selection Sort	$O(n^2)$
(iii) Heap Sort	O(nlgn)
(iv) Merge Sort	O(nlgn)
(v) Quick Sort	O(nlgn)

Grading: 1 pt each all or nothing

3) (5 pts) ALG (Bitwise Operators)

Determine the value of each of these arithmetic expressions in C. Please use the space below for your scratch work.

(i) 43 | 96	107
(ii) 117 & 74	64
(iii) 76 ^ 49	125
(iv) 11 << 2	44
(v) 330 >> 8	1

Grading: 1 pt each all or nothing, if answer is in binary, divide score by 2.