Computer Science Foundation Exam

January 12, 2019

Section I A

DATA STRUCTURES

SOLUTIONS

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

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	10	ALG	7	
3	5	ALG	3	
TOTAL	25		17	

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 struct definition:

```
typedef struct Employee
{
  char *first; // Employee's first name.
  char *last;  // Employee's last name.
  int ID;      // Employee ID.
} Employee;
```

Consider the following function, which takes three arrays – each of length *n* – containing the first names, last names, and ID numbers of *n* employees for some company. The function dynamically allocates an array of *n* Employee structs, copies the information from the array arguments into the corresponding array of structs, and returns the dynamically allocated array.

```
Employee *makeArray(char **firstNames, char **lastNames, int *IDs, int n)
{
  int i;
  Employee *array = malloc(sizeof(Employee) * n);      // 1 point

  for (i = 0; i < n; i++)     // The following blanks are worth 2 points EACH.
  {                           // Award 1 point each if close. Note that
                              // (strlen(…) + 1) must be parenthesized in order
                              // to earn full credit.

    array[i].first = malloc(sizeof(char) * (strlen(firstNames[i]) + 1));
    array[i].last = malloc(sizeof(char) * (strlen(lastNames[i]) + 1));

    strcpy(array[i].first, firstNames[i]);
    strcpy(array[i].last, lastNames[i]);
    array[i].ID = IDs[i];
  }
  return array;
}
```

a) Fill in the blanks above with the appropriate arguments for each *malloc()* statement.

b) Next, write a function that takes a pointer to the array created by the *makeArray()* function, along with the number of employee records in that array (*n*) and frees all the dynamically allocated memory associated with that array. The function signature is as follows:

```
void freeEmployeeArray(Employee *array, int n)
{
  int i;    // 1 point for declaring i and having no syntax errors below

  for (i = 0; i < n; i++)  // 1 point for looping correctly
  {
    free(array[i].first);  // 1 point for this free() statement
    free(array[i].last);   // 1 point for this free() statement
  }

  free(array);             // 1 point for this free() statement
}
```

2) (10 pts) ALG (Linked Lists)

Consider the following code:

```
void doTheThing(node *head, node *current)
{
  if (current == NULL)
    return;

  else if (current == head->next)
  {
    if (current->data == head->next->next->data)
      doTheThing(head, head->next->next->next);
    else if (current->data == head->next->next->data + 1)
      doTheThing(head, head->next->next->next->next);
    else if (current->data == head->next->next->data + 5)
      doTheThing(head, current->next->next->next);
    else if (current->data == head->next->next->data + 10)
      doTheThing(head, head->next);
    else
      doTheThing(head, current->next);
  }

  else
    doTheThing(head, current->next);
}
```

Draw a linked list that simultaneously satisfies **both** of the following properties:

1. The linked list has **exactly four nodes**. Be sure to indicate the integer value contained in each node.

2. If the linked list were passed to the function above, the program would either crash with a segmentation fault, get stuck in an infinite loop, or crash as a result of a stack overflow (infinite recursion).

Note: When this function is first called, the head of your linked list will be passed as *both* arguments to the function, like so:

```
doTheThing(head, head);
```

Hint: Notice that all the recursive calls always pass *head* as the first parameter. So, within this function, *head* will always refer to the actual head of the linked list. The second parameter is the only one that ever changes.

Solution:

The only way to get wrecked with this code is to trigger the `doTheThing(head, head->next)` call. Since we only trigger that call when `current == head->next`, then making that recursive call results in infinite recursion. (Continued on the following page.)

That specific recursive call is only executed when `current == head->next` (i.e., when `current` is the second node in the linked list) and when that second node has a value that is 10 greater than the value in the third node. For example:

```
[1]->[18]->[8]->[3]->

* The first and last values can be anything, but the second value needs to be
exactly 10 greater than the third value.
```

Note that none of the excessive `->next->next->next` accesses would ever cause segfaults here, since we always have four nodes in the linked list, and we never get to those accesses unless *current* is the second node in the linked list. Even the `head->next->next->next->next` access wouldn't cause a segfault; it would just pass NULL to the function recursively, which would hit a base case and return gracefully.

Grading:

10 points for a correct answer.

Note: If the 2nd node has a value 10 less than the 3rd node, still award 10/10.

5 points if the 2nd node has a value 1 greater than the 3rd node, thereby triggering the `head->next->next->next->next` access. That doesn't cause a segfault, but it's an understandable mistake and is the next best thing.

Note: If the 2nd node has a value 1 less than the 3rd node, still award 5/10.

2 points otherwise, as long as they draw a linked list with exactly four nodes, and each node contains an integer. (Also, any circular list with 4 nodes gets 2 points maximum.)

0 points otherwise.

FURTHER GRADING NOTES: A circular list should get at most 2 points. It's clear from the question that the intent is for the list not to be circular but a regular linked list. The reason this is clear is that the way the code is written, we look for the base case with a NULL pointer, but a circular linked list doesn't have one of those. So, ANY circular linked list of size 4 will cause an infinite loop, so one can put any four values down and the second item would be satisfied, which means the second item would be irrelevant. This should help a student realize that the intent was for the answer to be a regular linked list that isn't circular and what's being graded are the specific values they pick for the four nodes. When people refer to a regular linked list, they just say "linked list", they don't say "a linked list that isn't circular and doesn't have links." Instead, the assumption is that unless specified otherwise, a linked list has a head and a single pointer to the next node.

3) (5 pts) ALG (Stacks and Queues)

Consider the following function:

```
void doTheThing(void)
{
  int i, n = 5;  // Note: There are 9 elements in the following array.
  int array[] = {3, 18, 58, 23, 12, 31, 19, 26, 3};

  Stack *s1 = createStack();
  Stack *s2 = createStack();
  Queue *q = createQueue();

  for (i = 0; i < n; i++)
    push(s1, array[i]);

  while (!isEmptyStack(s1))
  {
     while (!isEmptyStack(s1))
       enqueue(q, pop(s1));   // pop element from s1 and enqueue it in q
     while (!isEmptyQueue(q))
       push(s2, dequeue(q));  // dequeue from q and push onto s2

     printf("%d ", pop(s2));  // pop from s2 and print element

     while (!isEmptyStack(s2))
       push(s1, pop(s2));     // pop from s2 and push onto s1
  }
  printf("Tada!\n");

  freeStack(s1);
  freeStack(s2);
  freeQueue(q);
}
```

What will be the exact output of the function above? (You may assume the existence of all functions written in the code, such as *createStack()*, *createQueue()*, *push()*, *pop()*, and so on.)

Solution: `3 18 58 23 12 31 19 26 3 Tada!`

(This function just ends up printing the contents of the array in order.)

Grading:

5 points for the correct output
4 points if their output was simply missing the "Tada!" or if their output was off by one value
2 points if they printed the array in reverse order.
0 points otherwise.

Feel free to award partial credit if you encounter something else that seems reasonable.

Computer Science Foundation Exam

January 12, 2019

Section I B

DATA STRUCTURES

SOLUTIONS

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

Question #	Max Pts	Category	Passing	Score
1	5	DSN	3	
2	10	ALG	7	
3	10	ALG	7	
TOTAL	25		17	

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 (Binary Trees)

Write a **recursive** function to print a postorder traversal of all the integers in a binary tree. The node struct and function signature are as follows:

```
typedef struct node
{
  struct node *left;
  struct node *right;
  int data;
} node;

void print_postorder(node *root)
{
  if (root == NULL)
    return;

  postorder(root->left);
  postorder(root->right);
  printf("%d ", root->data);
}
```

Grading:

+1 point for correct base case

+1 point for making both recursive calls (regardless of order)

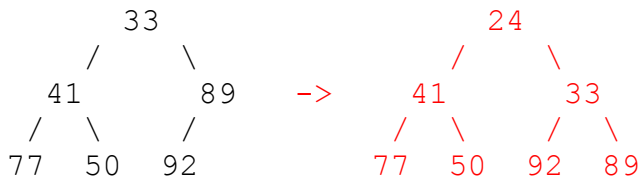
+1 point for printing root->data (regardless of order)

+1 point for having both recursive calls, printing root->data, **and** doing all those things in the correct order.

+1 point for getting all the syntax correct. (So, for example, if they called *postorder(root.left)* and *postorder(root.right)*, they can get the 1 point for making both recursive calls, but they lose this 1 point for using the dot instead of the arrow.)

2) (10 pts) ALG (Minheaps)

a) Show the result of inserting the value 24 into the following minheap.



Grading (4 pts for part a):

4/4 if correct

2/4 if not correct, but they satisfy at least one of the following: (1) 24 ends up at the root, (2) the structure of the tree is the same as above (despite where the values ended up).

0/4 otherwise

b) Show the result of deleting the root of the following minheap.

Grading (4 pts for part b):

4/4 if correct

2/4 if not correct, but they satisfy at least one of the following: (1) 41 ends up at the root, (2) the structure of the tree is the same as above (despite where the values ended up).

0/4 otherwise

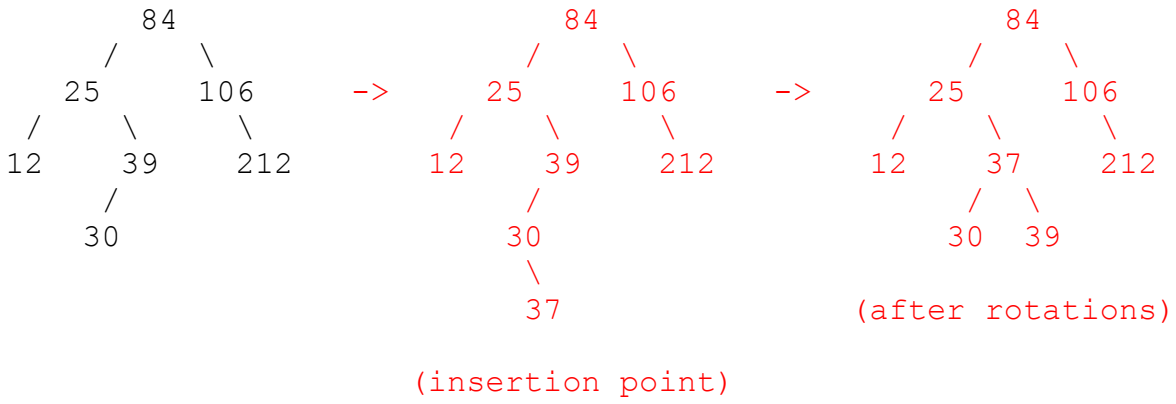
c) Using big-oh notation, what is the **worst-case** runtime for deleting the minimum element from a minheap that has *n* nodes?

Solution: O(log n)

Grading: 2 pts, all or nothing.

3) (10 pts) ALG (AVL Trees)

a) Show the result of inserting 37 into the following AVL tree:



Grading (6 points for part a):

6/6 for correct answer.

3/6 for something reasonably close. (Use your judgment. However, 84 must be the root in order for them to earn these points.)

0/6 otherwise.

b) Using big-oh notation, give the **best-case** runtime for inserting a new element into an AVL tree with *n* nodes:

Solution: O(log n) **Grading:** 1 point if correct, 0 otherwise

c) Using big-oh notation, give the **worst-case** runtime for inserting a new element into an AVL tree with *n* nodes:

Solution: O(log n) **Grading:** 1 point if correct, 0 otherwise

d) Using big-oh notation, give the **best-case** runtime for inserting a new element into a binary search tree with *n* nodes:

Solution: O(1) **Grading:** 1 point if correct, 0 otherwise

e) Using big-oh notation, give the **worst-case** runtime for inserting a new element into a binary search tree with *n* nodes:

Solution: O(n) **Grading:** 1 point if correct, 0 otherwise

Computer Science Foundation Exam

January 12, 2019

Section II A

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Passing	Score
1	10	ANL	7	
2	5	ANL	3	
3	10	ANL	7	
TOTAL	25		17	

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) ANL (Algorithm Analysis)

With proof, determine the Big-Oh run time of the function, f, below, in terms of the input parameter n. (Note: You may use results from algorithms studied previously in COP 3502 without restating the full proof of run time.)

```
int f(int array[], int n) {
    return frec(array, 0, n-1);
}

int frec(int array[], int lo, int hi) {

    if (lo == hi) return array[lo];

    int left = frec(array, lo, (lo+hi)/2);
    int right = frec(array, (lo+hi)/2+1, hi);

    int i, lCnt = 0, rCnt = 0;
    for (i=lo; i<=hi; i++) {
        if (abs(array[i]-left) < abs(array[i]-right))
            lCnt++;
        else
            rCnt++;
    }
    if (lCnt > rCnt) return lCnt;
    return rCnt;
}
```

The function f is a wrapper function that calls the recursive function frec. f takes in an array of size n while frec takes in a subsection of an array of size hi-lo+1. Let T(n) be the run time of the function frec where hi-lo+1 = n.

To determine what T(n) equals, first note that two recursive calls are made, each to arrays of size n/2. Each of these recursive calls, by definition, takes T(n/2) time. This is follows by a for loop that runs n times, inside of which there are only a few O(1) operations. Thus, we add O(n) to the runtime of the function for the second portion of the code. Thus, our total tally is:

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

If one recognizes this as the recurrence of Merge Sort solved in COP 3502, one can state that the result of solving this recurrence relation is **T(n) = O(nlgn).** Alternatively, either the Master Theorem or Iteration Technique can be used to arrive at the final solution for the recurrence.

Grading: 2 pts for setting up any sort of recurrence relation, 2 pts for recognizing that there are two recursive calls, 2 pts for recognizing that the input size for the recursive calls is n/2, 2 pts for recognizing that there is O(n) extra work in the function and 2 pts for the final solution.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm processing a two dimensional array with R rows and C columns runs in $O(RC^2)$ time. For an array with 100 rows and 200 columns, the algorithm processes the array in 120 ms. How long would it be expected for the algorithm to take when processing an array with 200 rows and 500 columns? Please express your answer in ***seconds.***

Let the algorithm with input size n have a runtime of $T(R,C) = kRC^2$, for some constant k. Using the given information we have:

$$T(100,200) = k(100)(200)^2 = 120ms$$
$$c = \frac{120}{4 \times 10^6} ms$$

Now, we must find T(200,500):

$$T(200,500) = k(200)(500)^2 = \frac{120ms}{4 \times 10^6} \times (200)(500)^2 = \frac{120 \times 50 \times 10^6}{4 \times 10^6} ms = 1500ms = 1.5s$$

Thus, our final answer is **1.5 seconds.**

Grading: 1 pt to set up the initial equation for k, 1 pt to solve for k, 2 pts to get answer in ms, 1 pt to convert to seconds. Give partial credit for the 2 pts if the setup is correct but some algebra issue occurred. Also, give full credit if the ratio method (which is also valid) is used instead of this method. Map points accordingly if some error is made using that method.

FURTHER GRADING NOTE: If the student used an algorithm run time of $O(R^2C^2)$ for any reason and correctly solved the problem for this different run time, 4 points out of 5 were awarded.

3) (10 pts) ANL (Summations and Recurrence Relations)

Determine the following summation in terms of n (assume n is a positive integer 2 or greater), expressing your answer in the form $an^3 + bn^2 + cn$, where a, b and c are rational numbers. (Hint: Try rewriting the summation into an equivalent form that generates less algebra when solving.)

$$\sum_{i=n^2-3}^{n^2+n-4} (i+4)$$

To simplify the algebra, do an index shift. Notice that the terms getting added are actually $n^2 + 1$, $n^2 + 2$, …, $n^2 + n$:

$$\sum_{i=n^2-3}^{n^2+n-4} (i+4) = \sum_{i=1}^{n} (n^2+i)$$

$$= \left(\sum_{i=1}^{n} n^2\right) + \left(\sum_{i=1}^{n} i\right)$$

$$= n(n^2) + \frac{n(n+1)}{2}$$

$$= n^3 + \frac{1}{2}n^2 + \frac{1}{2}n$$

Grading: 4 pts for a correct index shift (give partial as necessary), 1 pt for splitting the sum correctly, 2 pts for the sum of the constant, 3 pts for the sum of i.

If they don't do the index shift, then they are likely to be subtracting two sums. 3 pts for each of the two sums, 4 pts for the algebra of subtracting those sums.

If they try something else, try your best to map points to one of these two schemes.

Computer Science Foundation Exam

January 12, 2019

Section II B

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Passing	Score
1	5	DSN	3	
2	10	ALG	7	
3	10	DSN	7	
TOTAL	25		17	

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.

1) (5 pts) DSN (Recursive Coding)

Mathematically, given a function f, we recursively define $f^k(n)$ as follows: if k = 1, $f^1(n) = f(n)$. Otherwise, for k > 1, $f^k(n) = f(f^{k-1}(n))$. Assume that a function, f, which takes in a single integer and returns an integer already exists. Write a ***recursive*** function fcomp, which takes in both n and k (k > 0), and returns $f^k(n)$.

```
int f(int n);
```

Solution #1

```
int fcomp(int n, int k) {
    if (k == 1) return f(n);
    return f(fcomp(n, k-1));
}
```

Solution #2

```
int fcomp(int n, int k) {
    if (k == 1) return f(n);
    return fcomp(f(n), k-1);
}
```

Grading: 2 pts for the base case.
3 pts for the recursive case.

2) (10 pts) ALG (Sorting)

(a) (5 pts) Consider using Merge Sort to sort the array shown below. What would the state of the array be right before the ***last*** call to the Merge function occurs?

index	0	1	2	3	4	5	6	7	8	9
value	20	15	98	45	13	83	66	51	88	32

Answer:

index	0	1	2	3	4	5	6	7	8	9
value	**13**	**15**	**20**	**45**	**98**	**32**	**51**	**66**	**83**	**88**

Grading: 5 pts for the correct answer, 2 pts if each pair is sorted, 1 pt if the whole array is sorted, otherwise, count how many of the blanks are correct and divide by 2, rounding down.

(b) (5 pts) An inversion in an array, *arr*, is a distinct pair of values *i* and *j*, such that *i* < *j* and *arr[i]* > *arr[j]*. The function below is attempting to count the number of inversions in its input array, *arr*, of size *n*. Unfortunately, there is a bug in the program. Given that the array passed to the function has all distinct values, what will the function always return (no matter the order of values in the input array), in terms of n? Also, suggest a quick fix so that the function runs properly. (Note: analyzing inversions is important to studying sorting algorithm run times.)

```
int countInversions(int arr[], int n) {    // line 1
    int i, j, res = 0;                     // line 2
    for (i = 0; i < n; i++) {              // line 3
        for (j = 0; j < n; j++) {          // line 4
            if (arr[i] > arr[j])           // line 5
                res++;                     // line 6
        }                                  // line 7
    }                                      // line 8
    return res;                            // line 9
}                                          // line 10
```

Return value of the function in terms of n: $\frac{n(n-1)}{2}$, the sum of the first n-1 non-negative integers.

Line number to change to fix the function: 4

Line of code to replace that line: `for (j = i+1; j < n; j++) {`

As it's currently written, the code compares each value arr[i] to all other values in the array and adds 1 to res each time arr[i] is bigger. So, for the largest value in the array n-1 is added to res, for the second largest value in the array n-2 is added, and so forth, so res will simply be the sum of the integers from 0 to n-1. The issue with the code is that it doesn't enforce the restriction i < j given in the definition of an inversion. To enforce this, we force j > i by making j's starting value i+1, the smallest integer greater than j.

Grading: 2 pts return value, 1 pt line to change, 2 pts for the change.

3) (10 pts) DSN (Bitwise Operators)

In this problem we will consider buying a collection of 20 figurines, labeled 0 through 19, inclusive. The figurines come in packages. Each package has some non-empty subset of figurines. We can represent the contents of a single package using an integer in between 1 and $2^{20} - 1$, inclusive, where the bits that are on represent which figurines are in the package. For example, the integer $22 = 2^4 + 2^2 + 2^1$, would represent a package with figurines 1, 2 and 4. Each month, one package comes out. You greedily buy every package until you have all 20 figurines. Write a function that takes in an array of integers, *packages*, and its length, *n*, where *packages[i]* stores an integer representing the contents of the package on sale during month i, and returns the number of months you will have to buy packages to complete the set. It is guaranteed that each figurine belongs to at least one of the packages and that each value in the array packages is in between 1 and 2^{20}-1, inclusive. **<u>For full credit, you must use bitwise operators.</u>**

```
int monthsTillComplete(int packages[], int n) {

    int i = 0, mask = 0;

    while (mask != ((1<<20)-1) ) {
        mask |= packages[i];
        i++;
    }

    return i;

}
```

Grading: 1 pt using an integer to keep track of current items, 4 pts looping until all items are collected (bitshift not necessary, but easier), 4 pts to update current collection (3 pts if no bitwise operator is used), 1 pt increment i, 1 pt return appropriate value.

Max grade for solving correctly with no bitwise operators is 8 out of 10.