Computer Science Foundation Exam

May 20, 2017

Section I A

DATA STRUCTURES

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	DSN	7	
2	10	DSN	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)

Suppose we would like to create an array to store our Must Watch TV list. Currently our list is stored in a text file with the name of each TV show on a line by itself. The name of each show consists of only letters and underscores and doesn't exceed 127 characters. Write a function called makeTVList that reads these names from a file, allocates memory dynamically to store the names, stores them in a two-dimensional character array and returns a pointer to that array. Your function should take 2 parameters: a pointer to the file and an integer indicating the number of TV shows in the file. It should return a pointer to the array of shows. Be sure to allocate memory for the array dynamically and only allocate as much space as is needed. Namely, do not allocate 128 characters to store each show name. Instead dynamically allocate an appropriate number of characters as necessary. Use any necessary functions from string.h.

```
char ** makeTVList (FILE *ifp, int numShows) {

    char buffer[128];
    char **TVList = NULL;
    int i;

    TVList = malloc(numShows * sizeof(char *));          //2 pts

    for(i=0; i<numShows; i++) {                          //1 pt
        fscanf(ifp, "%s", buffer);                       //1 pt
        TVList[i] = malloc((strlen(buffer)+1)*(sizeof(char)));//3 pts
        strcpy(TVList[i], buffer);                       //2 pts
    }

    return TVList;                                       //1 pt
}
```

2) (10 pts) DSN (Linked Lists)

Suppose we have a stack implemented as a linked list. The stack is considered "full" if it has 20 nodes and empty if the head pointer is NULL. The nodes of the stack have the following structure:

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

Write a function to determine if the stack is full.

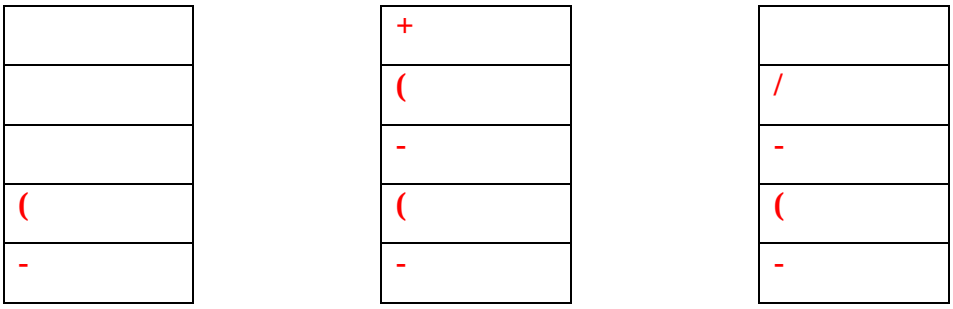
```
int isFull(node *stack) {

    int count = 0;                  //1 pt initializing a counter
    node *helper = stack;

    if (stack == NULL)              //2 pts checking if stack is null
        return 0;

    while(helper != NULL) {         //2 pts iter linked list
        count++;                    //1 pt incrementing counter
        helper = helper->next;      //1 pt advancing node
    }                               // Note: can stop at 20…

    if(count >= 20)             //2 pts returning true iff 20 or more
        return 1;

    return 0;                   //1 pt returning false if no

    // Note: return count >= 20; takes care of both…
}

// Alternate solution.

int isFull(node* stack) {

    int i;                                  // 1 pt
    for (i=0; i<20; i++) {                  // 2 pts
        if (stack == NULL) return 0;        // 3 pts
        stack = stack->next;                // 2 pts
    }
    return 1;                               // 2 pts
}
```

3) (5 pts) ALG (Stacks)

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (1, 2, and 3) in the infix expression.

1 2 3

(A + B) – (C - (D + E) / F) * G



1 2 3

Resulting postfix expression:

A	**B**	**+**	**C**	**D**	**E**	**+**	**F**	**/**	**-**	**G**	*****	**-**						

Grading: 1 point for each stack, 2 points for the whole expression (partial credit allowed.)

Computer Science Foundation Exam

May 20, 2017

Section I B

DATA STRUCTURES

SOLUTION

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

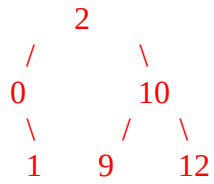
Question #	Max Pts	Category	Passing	Score
1	10	BST	7	
2	5	Heaps	3	
3	10	AVL Trees	7	
TOTAL	25		17	

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) DSN (Binary Search Trees)

(a) (3 pts) Given the following traversals, draw the Binary Search Tree they represent.
Pre-Order: 2, 0, 1, 10, 9, 12
Post-Order: 1, 0, 9, 12, 10, 2
In-Order: 0, 1, 2, 9, 10, 12



// 1 pt placing 2 at the root, 1 pt left subtree, 1 pt right subtree

(b) (5 pts) If the nodes of the BST have the following structure, construct a recursive function to count the number of nodes in the tree.

```
typedef struct bstNode {
   struct bstNode *left, *right;
   char word[20];
} bstNode;

int count(bstNode *root){

    if (root == NULL) return 0;              // 2 pts

    //1 pt 1, 1 pt left, 1 pt right
    return 1 + count(root->left) + count(root->right);
}
```

(c) (2 pts) Write a single line of code calling the count function that assigns the number of nodes in the left subtree of the tree pointed to by a pointer `myTreePtr` to the integer variable `leftCount`. You may assume that `myTreePtr` is not pointing to NULL and points to an actual `bstNode`.

```
int leftCount = count(myTreePtr->left);          // 2 pts
```

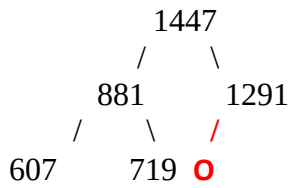
2) (5 pts) ALG (Heaps)

(a) (1 pts) In her computer science courses, Maria has learned some interesting things about prime numbers and data structures. She has decided to store some prime numbers in a Max-Heap using the tree representation of heaps. If Maria has stored 125 prime numbers, how tall would the Heap be?

Height = 6

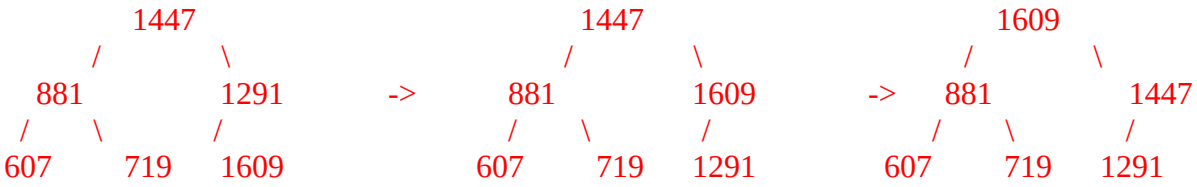
Heaps are complete binary trees, so the height is determined by $\lfloor \log_2 125 \rfloor$. You can also add the "levels" of the tree: 1 + 2 + 4 + 8 + 16 + 32 + 62. 1 pt all or nothing.

(b) (2 pts) Here is the Max-Heap after 5 insertions. Where will the next prime be inserted?



The next node must be added as the left child of 1291. 2 pts all or nothing.

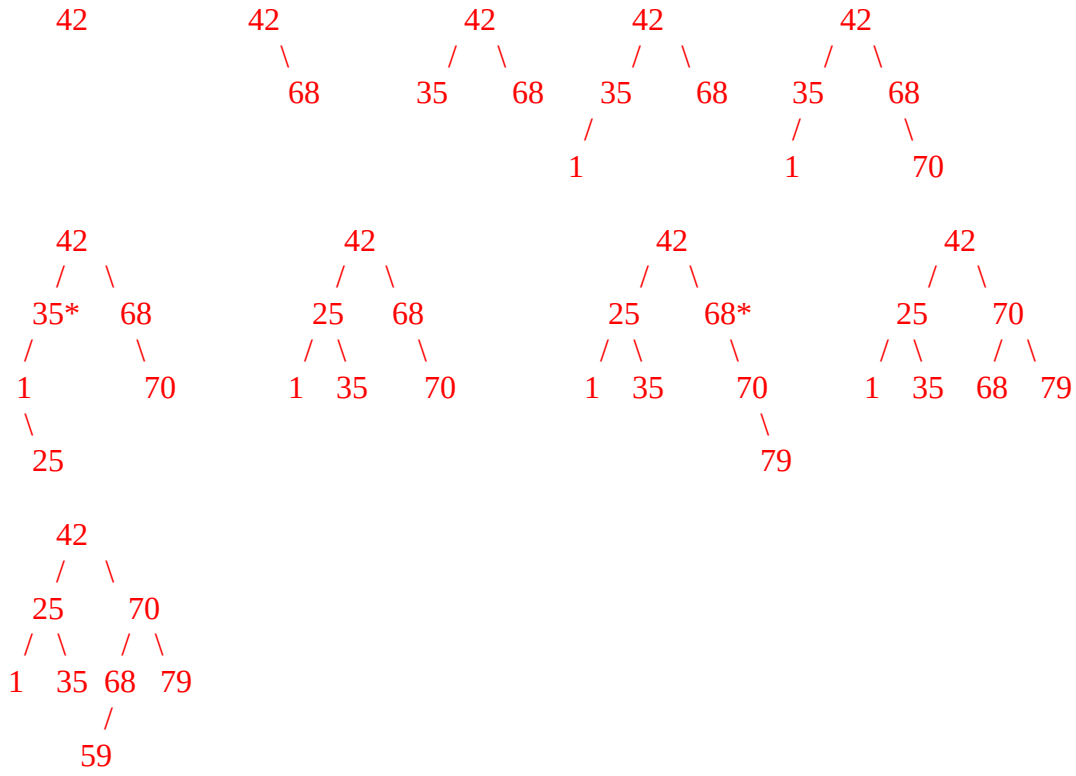
(c) (2 pts) Show each step of inserting 1609 into the Max-Heap.



Students should show both percolate up steps for 1609. 1 point per step. If they don't show the middle step and just have the final picture with 1609 as the root, award 1 point out of 2.

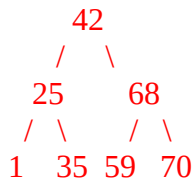
3) (10 pts) DSN (AVL Trees)

(a) (8 pts) Create an AVL tree by inserting the following values in the order given: 42, 68, 35, 1, 70, 25, 79, and 59. Show the state of the tree after each insertion.



Students should show each insertion step for 1 pt each. Imbalances should be detected and corrected for after inserting 25 and after inserting 79; detected at 35 and 68 respectively.

(b) (2 pts) Draw the state of the tree after the deletion of the node containing the value 79 from the tree at the end of part (a).



Deleting 79 creates an imbalance at 70 that must be corrected.

Computer Science Foundation Exam

May 20, 2017

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)

Determine the average case and worst case run-times, using Big-Oh notation, for the following algorithms or data structure operations. In order to earn credit, your answers must be in terms of the appropriate variables given in the question.

Algortithm/Operation	Average Case	Worst Case
Push operation onto a stack implemented with a linked list storing n elements.	O(1)	O(1)
Printing out each permutation of the integers 1, 2, 3, …, n. (Note: printing a single integer takes O(1) time.)	O(n*n!)	O(n*n!)
Insertion of a single node into a binary search tree with n nodes.	O(lg n)	O(n)
Deletion of a single node of an AVL tree with n nodes.	O(lg n)	O(lg n)
Merging a sorted array of size P with another sorted array of size Q, producing a newly allocated sorted array of P+Q elements.	O(P+Q)	O(P+Q)

Grading: 1 pt for each part, to earn the point the answer has to be perfectly correct. Please also accept $O(n^2n!)$ for the second part, as reasonable implementations could have this run time.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm to process an two dimensional array of size n x m takes O(nmlgn) time. If the algorithm takes 1 second to process an array of size n = 2^{20} by m = 2^5, how long will it take to process an array of size n = 2^{25} by m = 2^9. Please express your answer in minutes and seconds, with the number of seconds in between 0 and 59, inclusive.

Let the algorithm with input array size n x m have runtime of $T(n,m) = cnmlgn$, for some constant c. Using the given information we have:

$$T(2^{20}, 2^5) = c(2^{20})(2^5)lg2^{20} = 1\ sec$$
$$c(2^{25})(20) = 1\ sec$$
$$c = \frac{1}{20 \times 2^{25}}\ sec$$

Now, let's solve for T(2^{25}, 2^9)

$$T(2^{25}, 2^9) = c(2^{25})(2^9)lg2^{25}$$
$$T(2^{25}, 2^9) = \frac{1}{20 \times 2^{25}}(2^{25})(2^9)lg2^{25}\ sec$$
$$T(2^{25}, 2^9) = \frac{2^9}{20}(25)\ sec$$
$$T(2^{25}, 2^{10}) = \frac{2^9}{4}(5)\ sec$$
$$\underline{T(2^{25}, 2^{10}) = 2^7 \times (5) sec = 640 \text{ sec} = \mathbf{10\ minutes, 40\ seconds}}$$

Grading: 2 pts solving for c, 1 pt plugging into solve for new dimensions, 1 pt for some simplification, 1 pt for the final answer in minutes and seconds

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

Find the Big-Oh solution to the following recurrence relation using the iteration technique. Please show all of your work, including 3 iterations, followed by guessing the general form of an iteration and completing the solution. Full credit will only be given if all of the work is accurate (and not just for arriving at the correct answer.)

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

First, iterate three times:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$
$$= 2\left[2T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right] + n^2$$
$$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right] + n^2$$
$$= 4T\left(\frac{n}{4}\right) + \frac{n^2}{2} + n^2$$
$$= 4T\left(\frac{n}{4}\right) + \frac{3n^2}{2}$$
$$= 4[2T\left(\frac{n}{8}\right) + (\frac{n}{4})^2] + \frac{3n^2}{2}$$
$$= 4[2T\left(\frac{n}{8}\right) + \frac{n^2}{16}] + \frac{3n^2}{2}$$
$$= 8T\left(\frac{n}{8}\right) + \frac{n^2}{4} + \frac{3n^2}{2}$$
$$= 8T\left(\frac{n}{8}\right) + \frac{7n^2}{4}$$

In general, after k iterations we will have $T(n) = 2^k T\left(\frac{n}{2^k}\right) + \frac{(2^k-1)n^2}{2^{k-1}}$. We want to plug in the value of k for which $\frac{n}{2^k} = 1$,which is when $n = 2^k$. Note that for this value of k, 2^{k-1} = n/2, since 2 x 2^{k-1} = 2^k:

$$T(n) = nT(1) + \frac{(n-1)n^2}{\frac{n}{2}} = n(1) + 2n(n-1) = 2n^2 - n = O(n^2)$$

Grading: 2 pts for second iteration ($4T(n/4)+3n^2/2$), 2 pts for third iteration ($8T(n/8)+7n^2/4$), 2 pts for general form, 2 pts for value to plug into general form, 2 pts for final solution.

Computer Science Foundation Exam

May 20, 2017

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 <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>.

1) (5 pts) DSN (Recursive Coding)

Write a recursive function that returns the product of the digits of its integer input parameter, n. You may assume that n is non-negative. For example, productDigits(274) should return 56, since 2 x 7 x 4 = 56.

```
int productDigits(int n) {


    if (n < 10)                                  // 1 pt
        return n;                                // 1 pt

    return (n%10)*productDigits(n/10);           // 1 pt return
                                                 // 1 pts n%10
                                                 // 1 pt *pD(n/10)
}
```

2) (10 pts) ALG (Sorting)

(a) (5 pts) Show the contents of the following array after each iteration of Bubble Sort. The result after both the first and last iteration have been included for convenience. (Note: due to the nature of this question, relatively little partial credit will be awarded for incorrect answers.)

index	0	1	2	3	4	5	6	7
Initial	12	27	6	1	33	19	4	15
1st iter	12	6	1	27	19	4	15	33
2nd iter	6	1	12	19	4	15	27	33
3rd iter	1	6	12	4	15	19	27	33
4th iter	1	6	4	12	15	19	27	33
5th iter	1	4	6	12	15	19	27	33
6th iter	1	4	6	12	15	19	27	33
7th iter	1	4	6	12	15	19	27	33

Grading: 1 pt per each row, to earn the point all values in the row have to be correct.

(b) (5 pts) The array shown below has been partitioned exactly once (first function call in a Quick Sort of an array.) Which element was the partition element? Why?

index	0	1	2	3	4	5	6	7
Initial	16	19	13	12	9	27	49	33

Partition Element Index: **5 (Grading: 1 pt)**

Partition Element Value: **27 (Grading: 1 pt)**

Reason it was the Partition Element:

It is the only value for each everything to the left of it is less than it and everything to the right of it is greater than it. (For 16, 19, 13 and 12, we have 9 to its right, so these four can't be the partition. 9 can't be the partition since 12 is to its left, 49 can't be the partition since 33 to is to its right, and 33 can't be the partition because 49 is to its left.)

Grading: 3 pts for the reason, the details in the ()'s need not be in student responses, but a clear understanding that the partition element is the one where everything to its left is less than it and everything to its right is greater than it is necessary for full credit. Award partial credit as necessary.

3) (10 pts) DSN (Backtracking)

For the purposes of this question we define a Top Left Knight's Path to be a sequence of jumps taken by a single knight piece, starting at the top left corner of a R by C board, which visits each square exactly once. The knight may end on any square on the board. Recall that the way a knight jumps is by either moving 1 square left or right, followed by 2 squares up or down, OR by moving 2 squares left or right, followed by 1 square up or down. In this question you'll complete a recursive function that counts the total number of Top Left Knight's Paths for a particular R and C (which will be constants in the code.) Your code should use the constants R and C and should still work if the values of these constants were changed. Complete the recursive function below so that it correctly solves this task. Just fill out the blanks given to you in the recursive function.

```
#include <stdio.h>                          // Grading: 1 pt per slot, must be
#include <stdlib.h>                         // perfectly correct to earn the point.
#define R 6
#define C 6
#define NUMDIR 8
const int DR[] = {-2,-2,-1,-1,1,1,2,2};
const int DC[] = {-1,1,-2,2,-2,2,-1,1};

int main() {
    printf("There were %d Top Left Knight Paths.\n", countTours());
    return 0;
}
int countTours() {
    int used[R][C], i, j;
    for (i=0; i<R; i++)
        for (j=0; j<C; j++)
            used[i][j] = 0;
    return countToursRec(used, 0, 0, 0);
}
int countToursRec(int used[][C], int numMarked, int curR, int curC) {

    if (numMarked == R*C )

        return  1;

    int i, res = 0;
    for (i=0; i<NUMDIR; i++) {

        int nextR = curR + DR[i];
        int nextC = curC + DC[i];

        if (inbounds(nextR, nextC) && !used[nextR][nextC] ) {
            used[nextR][nextC] = 1 ;
            res += countToursRec(used, numMarked + 1, nextR , nextC );
            used[nextR][nextC] = 0 ;
        }
    }
    return res;
}

int inbounds(int nextR, int nextC) {
    return nextR >= 0 && nextR < R && nextC >= 0 && nextC < C;
}
```