Computer Science Foundation Exam

January 13, 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	10	DSN	
2	5	ALG	
3	10	ALG	
TOTAL	25	----	

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

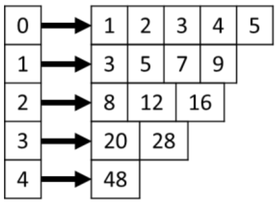
Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib,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

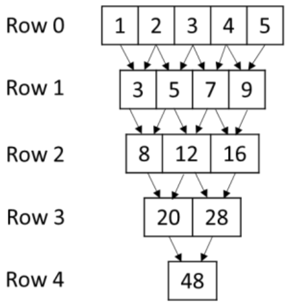
1) (10 pts) DSN (Dynamic Memory Management in C)

Starting with a 0-indexed dynamic integer 1D array called `base`, compute the triangular sum with only the **EXACT** proper amount of allocated space needed (no more or no less) by completing the user defined function definition. The triangular sum is the value of the elements present in the current dynamic array (based on the `row`) after the following process. For each `row`, the current index `i` will result in the value sum of the previous `row` (`row - 1`) at index `i` and `i + 1`. If the 2D array is named `trisum`, then the process to populate the values properly of the triangular sum will be as follows:

`trisum[row][i] = trisum[row – 1][i] + trisum[row – 1][i + 1]`

The below picture shows a nice visual representation of the triangular sum. Note that `base` is row 0.





This function will return an address to an array of arrays (dynamic 2D array) that visually represents the triangular sum. The second parameter `n` represents the number of elements in the `base` array `row` 0.

```
int ** triangularSum(int * base, int n) {

    int ** trisum = malloc(sizeof(int *) * n); //2pts
    trisum[0] = malloc(sizeof(int) * n); //1pt
    for (int i=0; i<n; i++)
        trisum[0][i] = base[i];           // 1 pt for assignment.

    for(int row = 1; row < n; ++row) { //1pt

        trisum[row] = malloc(sizeof(int) * (n - row)); //2pts

        for(int i = 0; i < n - row; ++i) //1pt
            trisum[row][i] = trisum[row-1][i] + trisum[row-1][i+1]; //1pt

     }

     return trisum; //1pt
}
```

Grading Notes: Take an integer number of points. For two small errors that you believe are each worth less than a point, take off 1 pt total. If there's only one tiny error (say arrow instead of one dot) correct it and give full credit. There are other ways to structure this.

2) (5 pts) ALG (Linked Lists)

Suppose we have a singly linked list implemented with the structure below and a function that takes in the head of the list.

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

void whatDoYouDo(node_t * head){
     node_t * temp = head;
     node_t * temp2 = head->nextptr;
     int a;

     while(temp->nextptr != NULL){
          a = temp->data;
          temp->data = temp2->data;
          temp2->data = a;

          temp = temp->nextptr;

          if(temp->nextptr != NULL){
               temp = temp->nextptr;
               temp2 = temp->nextptr;
          }
     }
}
```

If we call whatDoYouDo(head) on the following list, show the list after the function has finished.

head -> 5 -> 2 -> 1 -> 8 -> 7? Please fill in the designated slots below.

head → __2__ → __5__ → __8__ → __1__ → __7__

Grading: Each slot is worth 1 pt. All or nothing. No partial credit.

3) (10 pts) ALG (Stacks)

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (A, B, and C) in the infix expression.

```
           A                          B              C
3 + 1 - 7 * (4 / 2 + 5) * 8 - 7 / (5 - 3 + (5 + 7) / (3 * 2))
```

		/
	-	+
	(	(
*	/	/
-	-	-
A	B	C

Note: A indicates the location in the expression **AFTER** the multiplication and before the open parenthesis. B indicates the location in the expression **AFTER** the subtraction and before the value 3. C indicates the location in the expression **AFTER** the division and before the open parenthesis.

Resulting postfix expression:

3	**1**	**+**	**7**	**4**	**2**	**/**	**5**	**+**	*****	**8**	*****	**-**	**7**	**5**	**3**	**-**	**5**	**7**	**+**	**3**	**2**	*****	**/**	**+**	**/**	**-**

Note: There are exactly the correct number of boxes above. These should be filled with 14 numbers and 13 operators.

Grading: 2 pts for each stack, 4 pts for the total expression. Give partial as necessary.

Computer Science Foundation Exam

January 13, 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	10	DSN	
2	10	ALG	
3	5	ALG	
TOTAL	25		

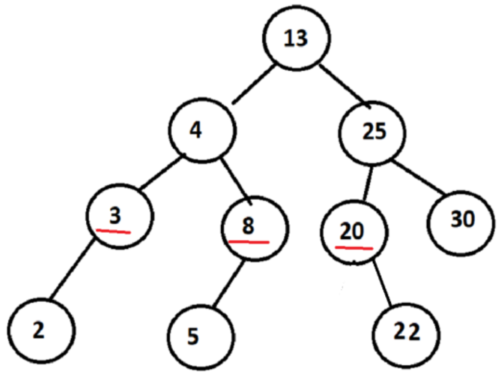
You must do all 3 problems in this section of the exam.

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 (Binary Trees)

Write a function named *sumSingleParents*() that takes a pointer to the root of a binary tree (*root*) and returns the sum of all the values in the nodes with a single child.

For example, if you pass the root of the following binary tree, the function should return 31 (=3+8+20) as the nodes containing 3, 8, and 20 have only one child:



You must write your solution in a **single** function. You cannot write any helper functions.

The function signature and node struct are given below.

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

int sumSingleParents(node *root) {

    if(root == NULL)                          // 1 pt
        return 0;

    int sum = 0;                              // 1 pt

    if((root->left == NULL && root->right != NULL) ||   // 2 pts
       (root->left != NULL && root->right == NULL))
        sum+= root->data;                               // 1 pt

    sum +=  sumSingleParents(root->left);               // 2 pts
    sum +=  sumSingleParents(root->right);              // 2 pts
    return sum;                                         // 1 pt
}
```

Note: Many ways to do this. Map points accordingly. Watch out for NULL ptr errors.

2) (10 pts) ALG (Heaps)

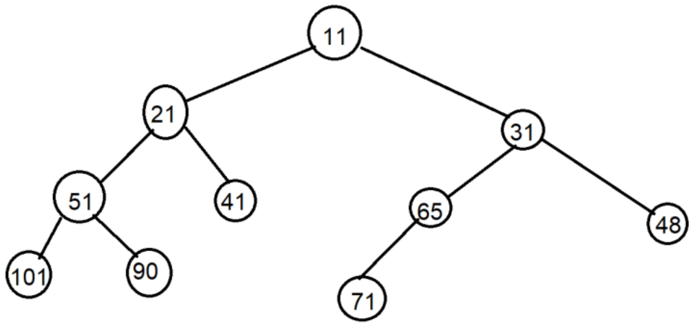
(a) (3 pts) A heap is represented by the array below. The first item is stored at index 1. Answer the following questions (**please answer the data not the index where it's stored.)**

index	1	2	3	4	5	6	7	8	9	10
data	7	11	13	16	18	19	24	21	20	35

i). Who is the left child of 13: _**19**____, ii). Right child of 16: _**20**__iii) parent of 24:__**13**____

Gade: 3 pts (1 pt for each)

(b) (2 pts) Consider the following tree. Is this a valid minheap? Justify your answer. *Just saying yes/no has no credit without justification.*



It is not a valid minheap because it's not a complete binary tree. Specifically, 41 has no left child but 65 has a child, so the nodes on the bottom most level aren't filled out left to right.

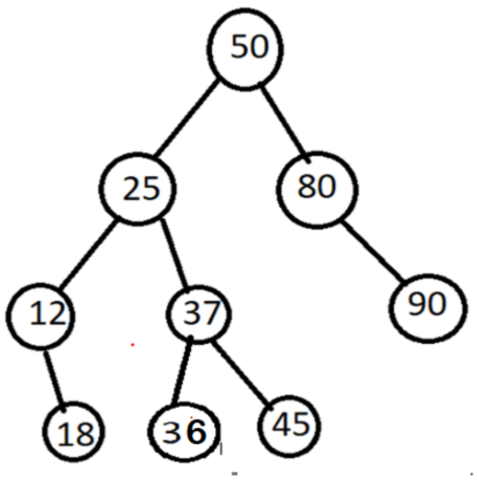
Grade: 2 pts all or nothing.

(c) (5 pts) Consider a **minheap** stored in an integer array *int heaparray[100]*, which is globally declared. Write a percolateUp function that takes an index and perform the full percolate up operation for the item at that index. While writing the code, you can assume that there is a swap function available for you that is described below.

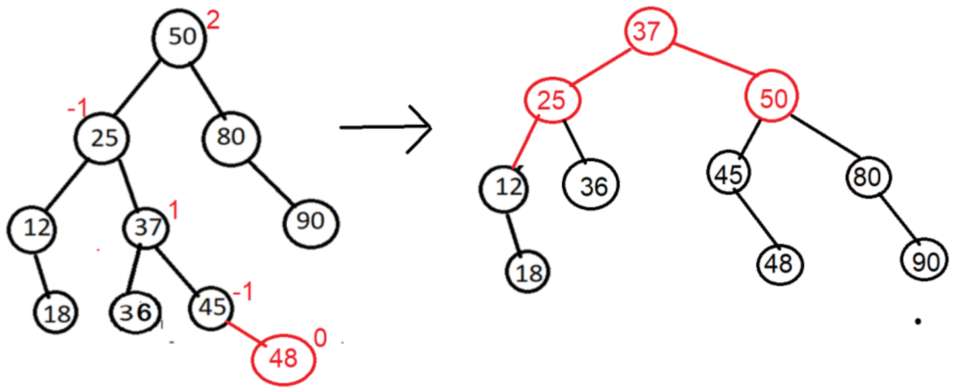
```
// swap(int* ptrA, int* ptrB) - swaps the contents in the variables
//                              pointed to by ptrA and ptrB.
void percolateUp(int idx){
    if (idx > 1) {  // 1 pt
        if (heaparray[idx/2] > heaparray[idx]) { // 1  pt
            swap(&heaparray[idx], &heaparray[idx/2]); // 2 pts
            percolateUp(idx/2);  // 1 pt
        }
    }
}
```

3) (5 pts) ALG (AVL Trees)

Show the final result of inserting 48 into the AVL tree below. Draw a box around your final answer.



Answer:



Grading: <u>Automatic 0 out of 5 if not valid BST or not valid AVL tree.</u>

1 pt valid AVL with right values (0 if this isn't satisfied)
1 pt for 37 at root,
1 pt for 25 left or root
1 pt for 50 right of root
1 pt for the rest being attached properly

Note: no need to show the difference in left and right heights.

Computer Science Foundation Exam

January 13, 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	10	ANL	
2	5	ANL	
3	10	ANL	
TOTAL	25		

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

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

1) (10 pts) ANL (Algorithm Analysis)

What is the worst case run-time of each of the following algorithms/operations? Please give your answers in Big-Oh form, using the appropriate variables in each question.

(a) Inserting 1 item into a binary search tree storing ***n*** items. **O(n)**

(b) Inserting 1 item into an AVL Tree storing ***n*** items. **O(lg n)**

(c) Printing out each number in base ***b*** with exactly ***k*** digits. Assume printing one digit takes O(1) time. **O(k*b^k)**

(d) Creating a heap using the most efficient algorithm out of ***n*** unsorted values. **O(n)**

(e) Deleting the third item in a linked list (of more than 3 items) and returning a pointer to the front of the adjusted list. **O(1)**

(f) Determining the number of integers that are included in **both** of two separate lists of **n sorted integers**, using the most efficient algorithm. **O(n)**

(g) Executing ***p*** consecutive pop operations on a stack that initially had ***n*** elements. (Note: ***p*** < ***n***.) **O(p)**

(h) Sorting ***n*** unsorted items via Heap Sort. **O(nlgn)**

(i) Converting a positive integer ***n*** expressed in decimal into binary. **O(lg n)**

(j) Adding a ***c*** digit integer to a ***d*** digit integer, where the integers are stored in arrays, digit by digit. **O(max(c,d)) or O(c+d)**

Grading: 1 pt for each, all or nothing

2) (5 pts) ANL (Algorithm Analysis)

A O(n^3) image processing algorithm took 125 milliseconds to index ***n*** = 400 images. How long would it be expected for this algorithm to take to index ***640*** images, in milliseconds? **Please show all your work, including algebraic simplification, which is part of what is being tested with this question.**

Let T(n) be the run time of the algorithm. Then there is some constant c such that

$T(n) = cn^3$

$$T(400) = c(400^3) = 125ms \rightarrow c = \frac{125\,ms}{400^3}.$$

$$T(640) = c(640^3) = \frac{125\,ms}{400^3} \times 640^3 = (125ms) \times \left(\frac{640}{400}\right)^3 = (125ms) \times \left(\frac{8}{5}\right)^3 = 125ms \times \frac{512}{125} = \mathbf{512ms}$$

Grading: 1 pt set up to solve for c
1 pt solve for c (no simplify)
1 pt plug in 640
2 pts to get to final answer

3) (10 pts) ANL (Summations)

Determine a closed form solution to the following summation in terms of **n**. Please leave your answer in **factored** form. Specifically, your answer should be of the form $\frac{(n+a)(n+b)(n+c)}{d}$, where a, b, c and d are all integers.

$$\sum_{i=1}^{n}\sum_{j=1}^{i} j$$

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

$$= \frac{1}{2}[(\sum_{i=1}^{n} i^2) + (\sum_{i=1}^{n} i)\,]$$

$$= \frac{1}{2}[\left(\frac{n(n+1)(2n+1)}{6}\right) + \left(\frac{n(n+1)}{2}\right)]$$

$$= \frac{1}{12}[\big(n(n+1)(2n+1)\big) + (3n(n+1))\,]$$

$$= \frac{n(n+1)}{12}[(2n+1)+3\,]$$

$$= \frac{n(n+1)}{12}[2n+4\,]$$

$$= \frac{n(n+1)}{12}[2(n+2)\,]$$

$$= \frac{n(n+1)(n+2)}{6}$$

Grading: 1 pt inner sum, 1 pt split sum, 2 pts plug into i^2 formula, 2 pts plug into I formula
4 pts algebra (decide partial as necessary)

Computer Science Foundation Exam

January 13, 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	5	ALG	
3	10	DSN	
TOTAL	25		

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

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

1) (10 pts) DSN (Recursive Coding)

Finish the function below so that it determines the **number of permutations** of size ***n*** (values 0 through ***n***-1, inclusive) such that each pair of adjacent values is within ***maxgap*** of each other. You can assume any spot in the permutation that has not been filled in yet will be 0. The function should also take in an array, ***perm***, the permutation, a second array, ***used***, which indicates the values that have been used in the permutation, and an integer, ***k***, representing which is the current empty spot (0-indexed) to be filled in. **Functions that fail to utilize recursion will receive 0 points**. (For example, if ***n*** = 4, ***maxgap*** = 2 and ***k*** = 0, the only permutations of size 4 that would not be counted are the ones that have 1 and 4 adjacent, since the difference between these is 3, which is bigger than maxgap.)

```
int kClosePerm(int* perm, int* used, int n, int maxgap, int k) {


    if (n == k)

        return 1; ;                 // 1 pt

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

        if (used[i]) continue;                          // 1 pt

        if (k>0 && abs(perm[k-1]-i) > maxgap) continue; // 2 pts

        used[i] = 1;                                    // 1 pt
        perm[k] = i;                                    // 1 pt
        res += kClosePerm(perm, used, n, maxgap, k+1);  // 3 pts
        used[i] = 0;                                    // 1 pt

    }

    return res;
}
```

2) (5 pts) ALG (Sorting)

Show the result after each iteration of performing Selection Sort, where we select for the **maximum** element in each iteration, on the array show below. For convenience, the result after the first and last iterations are provided. The first row (iteration 0) of the table contains the original values of the array.

Iteration	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7
0	13	11	9	16	12	15	10	5
1	13	11	9	5	12	15	10	16
2	13	11	9	5	12	10	15	16
3	10	11	9	5	12	13	15	16
4	10	11	9	5	12	13	15	16
5	10	5	9	11	12	13	15	16
6	9	5	10	11	12	13	15	16
7	5	9	10	11	12	13	15	16

Grading: 1 pt for each row, row has to be completely correct to get the point.

3) (10 pts) DSN (Bitwise Operators)

It is getting harder and harder to stay green while using computers. You decided that you will reduce your carbon footprint by storing fewer bits for your integers. How you might ask? You will store your integer plus some power of 2. Your goal will be to use as few on-bits as possible in the resulting sum. This way you will have less bits on and just like turning off the lights when you leave a room you will be saving energy and the planet. Complete the function below so that it determines and returns the least number of bits that will be on after adding a positive power of 2 to your number. For full credit your function should take O(***b***) time where ***b*** is the number of bits in an int. You are guaranteed that no positive power of 2 added to the original number will result in an overflow.

For example, the value 76 = $2^2 + 2^3 + 2^6$ can have 4 added to it to result in 80 = $2^4 + 2^6$, which requires only 2 on-bits. No better result can be achieved by adding a different power of 2.

```
int leastBitsOn(int x) {

    int numOn = 0;
    int cur = 0;
    int longest = 0;

    for (int i = 0; i < 8 * sizeof(int); i++) {

        // Determine if the bit is on (3 pts)
        if ((x&(1<<i)) != 0){
            cur++; // Update the bits on in the current run (1 pt)
            numOn++; // Update the bits on the current number(1 pt)
        }

        else {
            cur = 0;  // Reset, 1 pt
        }

        // (+2 pts) Update the longest run of bits.
        if (longest < cur) {
            longest = cur;
        }
    }

    // (+2 pts) Remove the longest run  AND  (+1 pt) add 1
    return numOn + 1 - longest;

}
```