# Computer Science Foundation Exam

## January 11, 2025

## 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 | 10 | DSN | |
| 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 (Dynamic Memory Management in C)

Consider the following typedef struct definition that represents a book.

```
//struct representing a book with content
typedef struct {
    char ** sentences; // actual sentences
    int numSentences; // total number of sentences
    char * title; // book title
    char * author ; // book author
} book_t;
```

Complete the following user defined function definition that properly deallocates all memory associated with the heap space of the struct type `book_t`. The parameters of the function contains the reference to heap space of where the **array** of `book_t` is stored along with the total number of elements as `numBooks`. Note that within each type `book_t`, `sentences` is an array of `numSentences` strings, where each string was dynamically allocated, as were the strings `title` and `author`.

```
void cleanUp(book_t * lib, int numBooks){


    for(int x = 0; x < numBooks; x++)                    // 1 pt
    {
        for(int y = 0; y < lib[x].numSentences; y++)     // 2 pts
            free(lib[x].sentences[y]);                   // 2 pts

        free(lib[x].sentences);                          // 1 pt
        free(lib[x].title);                              // 1 pt
        free(lib[x].author);                             // 1 pt
    }

    free(lib);                                           // 1 pt

     // 1 pt for correctly using . all the time, so just 1 pt
     // off total if -> was used at all.




}
```

**2)** (10 pts) DSN (Linked Lists)

The function below is given two sorted, singly linked lists, where each node contains a 2D coordinate (x, y). The lists are sorted by the x-coordinate (increasing order) first and then by the y-coordinate (increasing order). Complete the following user defined function `merge2DCoordinates` so that it merges the two sorted linked lists into one sorted linked list, preserving the order of the coordinates. In particular, your code should NOT allocate or free any memory. Instead, your code should simply change where some of the existing pointers (`next`) are pointing and return a pointer to the front of the updated list. For full credit, **write your code recursively.** (Note: The recursive solution is also shorter to write.)

```
typedef struct node_s {
    int x;
    int y;
    struct node_s* next;
} node_t;

node_t * merge2DCoordinates (node_t* ptr1, node_t* ptr2) {

    if (ptr1 == NULL && ptr2 == NULL ) return NULL;    // 1 pt
    if (ptr1 == NULL) return ptr2;                     // 1 pt
    if (ptr2 == NULL) return ptr1;                     // 1 pt

    // 3 pts if statement condition to split into 2 cases.
    if (ptr1->x < ptr2->x || (ptr1->x == ptr2->x && ptr1->y < ptr2->y)) {

        // 2 pts for call, connect and return.
        ptr1->next = merge2DCoordinates(ptr1->next, ptr2);
        return ptr1;
    }

    // 2 pts for call, connect and return.
    ptr2->next = merge2DCoordinates(ptr1, ptr2->next);
    return ptr2;
}
```

**3)** (5 pts) ALG (Stack)

Consider the following C code that represents a stack that holds a list of values. Show the contents of the stack **right after** each indicated point commented (A, B, and C), under the assumption that the `followStack` function is called with a pointer to a `stack_t` that is empty.

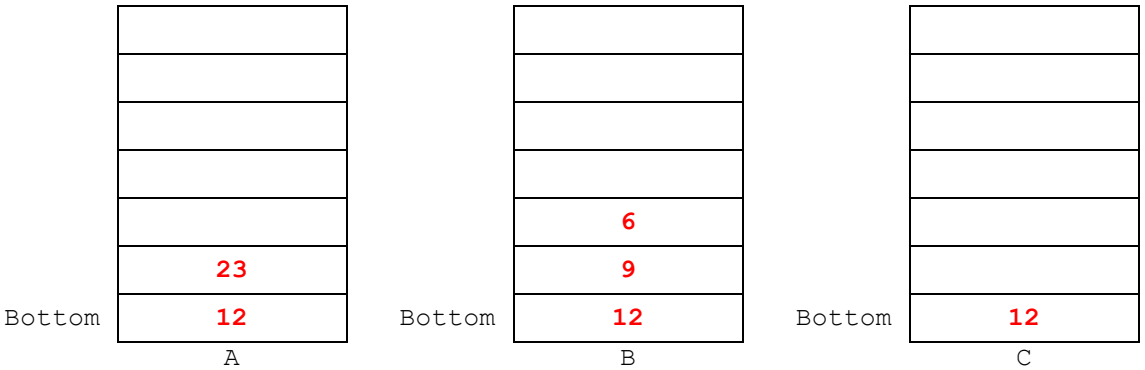
```
typedef struct node_s{
      int data;
      struct node_s * next;
}node_t;

typedef struct{
      node_t * top;
}stack_t;

void push(stack_t * s, int data);
int pop(stack_t * s);

void followStack(stack_t * myStack){

      int x;
      push(myStack, 12);
      push(myStack, 5);
      push(myStack, -8);
      x = pop(myStack);
      x = pop(myStack);
      push(myStack, 23); //A
      x = pop(myStack);
      push(myStack, 17);
      push(myStack, -3);
      x = pop(myStack);
      x = pop(myStack);
      push(myStack, 9);
      push(myStack, 6); //B
      push(myStack, -14);
      x = pop(myStack);
      x = pop(myStack);
      x = pop(myStack);
      push(myStack, 34);
      x = pop(myStack); //C
}
```




**Grading: 1 pt first stack, 2 pts second stack, 2 pts last stack, can only award partial credit for stacks B and C (1 pt if it's close).**

# Computer Science Foundation Exam

## January 11, 2025

## Section B

## ADVANCED DATA STRUCTURES

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

## SOLUTION

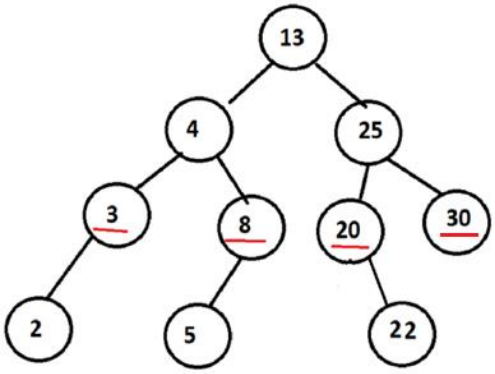
| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 10 | DSN | |
| 2 | 10 | DSN | |
| 3 | 5 | ALG | |
| TOTAL | 25 | | |

**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 <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 Trees)

Write a **recursive** function named `sumAtDepth` that takes a pointer to the root of a binary tree, `root`, and non-negative integer, `depth`, and returns the sum of all the values in the nodes that are at a level `depth` below the root. For example, if you pass the root of the following binary tree and `depth` = 2, the function should return 61 ( = 3 + 8 + 20 + 30) since the each of the nodes storing 3, 8, 20 and 30 are 2 levels below the root node of the tree. You may assume that depth is a non-negative integer.




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 sumAtDepth(node *root, int depth) {

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

  if (depth == 0)                // 1 pt
    return root->data;           // 1 pt

  // 1 pt return, 2 pts each recursive call, 1 pt adding
  return sumAtDepth(root->left, depth-1) + sumAtDepth(root->right, depth-1);
}
```

**2)** (10 pts) DSN (Heaps)

You are given an array, `arr`, of `size` integers. Write a **non-recursive** function that takes in this array and its size as input and returns 1 if the array represents a min-heap, and 0 otherwise. Recall that in an array implementation of a heap, index 0 isn't used and the root is stored at index 1. Since index 0 is unused, the function is checking if the array stores a valid heap in indexes 1 through `size-1`, inclusive, thus the number of nodes in the tree represented is `size-1`. **You may assume `size >= 2`.**

```
int isMinHeap(int arr[], int size) {

    for (int i = 1; 2*i < size; i++) {                  // 2 pts

        if (arr[i] > arr[2 * i])                        // 2 pts
            return 0;                                   // 1 pt

        if (2*i+1 < size && arr[i] > arr[2*i+1])        // 3 pts
            return 0;                                   // 1 pt

    }

    return 1;                                           // 1 pt

}
```

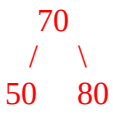
**Grading note: if 1 is returned pre-maturely, take off 4 points (so max score of 6 in this case).**

**3)** (5 pts) ALG (AVL Trees)

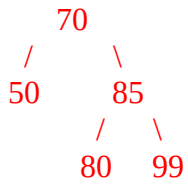
Insert the following integers into an AVL tree in the given order. Whenever an insertion causes the tree to become unbalanced, rebalance it immediately before proceeding with the next insertion. Continue this process, rebalancing as needed, until all elements have been added. Put a box around the state of the tree after each of the bolded-underlined elements are inserted. (Each of these pictures will be worth 1 point.)

50, 80, **70,** 99, **85**, **100,** **95** and **84**

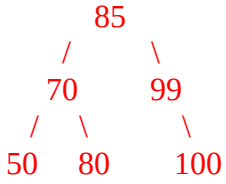
After 70 is inserted and rebalanced, tree is



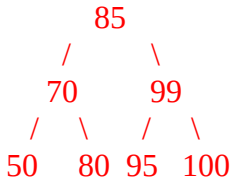

After 85 is inserted and rebalanced, tree is



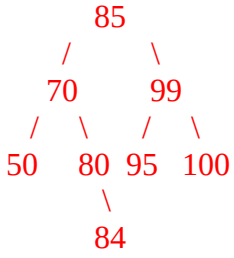

After 100 is inserted and rebalanced, tree is




Here is the tree after 95 is inserted:




Here is the tree after 84 is inserted:




**Grading: 1 pt for each of the trees shown above. Tree has to be perfectly correct to get the point.**

# Computer Science Foundation Exam

## January 11, 2025

## Section C

## ALGORITHM ANALYSIS

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

**SOLUTION**

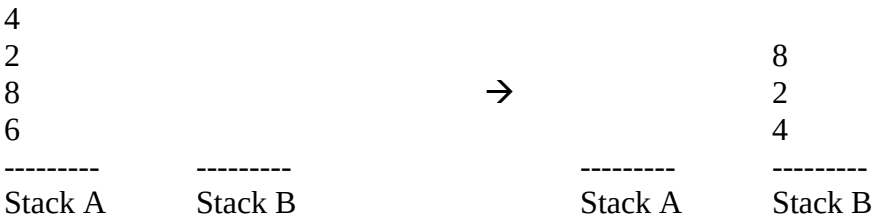
| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 5 | ANL | |
| 2 | 10 | ANL | |
| 3 | 10 | ANL | |
| TOTAL | 25 | | |

**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 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)** (5 pts) ANL (Algorithm Analysis)

Consider an implementation of a queue with two stacks, A and B. If the last operation was an enqueue, all elements will be in stack A and the top of the stack represents the back of the queue. If the last operation was a dequeue, then all of the elements will be on stack B and the top of the stack will represent the front of the queue. For example, if we start with an empty queue and enqueue the items 6, 8, 2 and 4 in succession, the picture on the left is what both stacks look like. If we follow that up with a dequeue, we first must pop off each item from stack A and push each item onto stack B, and then remove the top element of the stack




(a) (2 pts) What would be the total run-time, in terms of n, of n enqueue operations, followed by n dequeue operations, using this implementation. (Assume both stacks are implemented efficiently.) Please give your answer as a simplified Big-Oh answer.

As each enqueue operation adds to the top of Stack A, each of these will run in O(1) time, because they are in succession and we don't touch Stack B at all. There are n of these operations. The first dequeue will take O(n) time because all items are popped off of stack A and subsequently pushed onto stack B. Finally, each of the following n – 1 dequeue operations will each take O(1) time. Adding this all up we get nO(1) + O(n) + (n-1)O(1) = O(n).

**O(n)** **Grading: 2 pts all or nothing, no reasoning necessary.**

(b) (3 pts) What would be the total run-time, in terms of n, of n enqueue operations, followed by n more alternating enqueue and dequeue operations? Please give your answer as a simplified Big-Oh answer.

The first n+1 enqueue operations take O(n) time total, as described above. If dequeues and enqueues are then alternated, each with a queue size of either n+1 or n, then every single one of these operations will take O(n) time because they will involve either transferring n+1 items from Stack A to Stack B, or transferring n items from Stack B to Stack A. Since there are n of these operations, our total time is O(n) + nO(n) = $O(n^2)$.

**$O(n^2)$** **Grading: 3 pts all or nothing, no reasoning necessary.**

**2)** (10 pts) ANL (Algorithm Analysis)

An algorithm that processes a grid of R rows and C columns runs in O(RlgC) time. It turns out that for any R by C grid, we can transpose the grid so that it has C rows and R columns instead and solve the problem on that grid to get the same answer. Fred ran the code with R = $10^6$ and C = $10^2$ and it took **3 hours** to run. Shanille transposed the grid and reran the code with R = $10^2$ and C = $10^6$ to prove to Fred how inefficient his technique was. How long, **in seconds,** would be expect Shanille's execution of the code to take? **Please answer as a decimal to two places.**

Let T(R, C) = kRlgC be the run-time of the algorithm on a grid with R rows and C columns, where k is a constant.

$T(10^6, 10^2) = k10^6 \lg(10^2) = 3\ hours$

$k(2 \times 10^6)\lg(10) = 3\ hours \qquad \rightarrow k = \frac{3\ hours}{2\times 10^6 \lg(10)}$

Now, we must solve for $T(10^2, 10^6)$:

$$T(10^2, 10^6) = (\frac{3\ hours}{2 \times 10^6 \lg(10)})10^2 \lg(10^6)$$

$$= (\frac{3\ hours}{2 \times 10^6 \lg(10)})10^2(6)\lg 10$$

$$= (\frac{18\ hours}{2 \times 10^4}) \times \frac{60\ min}{1\ hour} \times \frac{60\ sec}{1\ min}$$

$$= (\frac{9\ hours}{10^4}) \times 3600 sec/hr$$

$$= (9)(.3600)sec$$

$$= \mathbf{3.24\ seconds}$$

**Grading:**
**1 pt setting up equation for k**
**2 pts solving for k no simplification**
**2 pts setting up equation for Shanille**
**2 pts correctly converting from hours to seconds**
**1 pt log simplification**
**2 pts rest of the simplification to the correct final form**

**3)** (10 pts) ANL (Recurrence Relations)

Use the iteration technique to find **an exact closed-form** solution to the recurrence relation defined below for all positive integers n:

$$T(1) = 1$$
$$T(n) = 2T(n-1) + 5, for\ all\ integers\ n \geq 2$$

Please explicitly show the work for the first three iterations before attempting to find the form for an arbitrary iteration, followed by arriving at the closed form. Hint: Your answer should be of the form $T(n) = a(b^n) + c$, where *a*, *b*, and *c* are all integers.

Here are the first three iterations:

$T(n) = 2T(n-1) + 5$ // Iteration #1 **Grading: 1 pt**
$T(n) = 2(2T(n-2) + 5) + 5$
$T(n) = 4T(n-2) + (10 + 5)$
$T(n) = 4T(n-2) + 15$ // Iteration #2 **Grading: 1 pt**
$T(n) = 4(2T(n-3) + 5) + 15$
$T(n) = 8T(n-3) + (20 + 15)$
$T(n) = 8T(n-3) + 35$ // Iteration #3 **Grading: 2 pts**

After k iterations, we have:

$T(n) = 2^k T(n-k) + 5(2^k - 1)$ **Grading: 2 pts**

Since we know T(1), plug in k = n – 1 into this formula: **Grading: 1 pt**

$T(n) = 2^{n-1}T(n-(n-1)) + 5\left(2^{(n-1)} - 1\right)$
$= 2^{n-1}T(1) + 5(2^{n-1}) - 5$ **Grading: 1 pt**
$= 2^{n-1} + 5(2^{n-1}) - 5$
$= 6(2^{n-1}) - 5$ **Grading: 1 pt**
$= (3)(2)(2^{n-1}) - 5$
$= 3(2^n) - 5$ **Grading: 1 pt**

**Note: 1 pt is allocated to factor out the 2 from the 6 and include it in the exponent.**

# Computer Science Foundation Exam

## January 11, 2025

## 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 | 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)

A regular odometer of 6 digits counts from 000000 to 999999. A lucky odometer setting is one that contains the digit 7 at least twice. Complete the **recursive function** below so that the code below prints out each lucky odometer setting of n = 6 digits. In the recursive function, k represents the number of digits of the odometer already filled in.

```
#include <stdio.h>
#include <stdlib.h>

int numd(int* odometer, int n, int d);
void printlucky(int n);
void printluckyrec(int* odometer, int k, int n);
void print(int* odometer, int n);

int main() {
    printlucky(6);
    return 0;
}

void printlucky(int n) {
    int* odom = malloc(n*sizeof(int));
    printluckyrec(odom, 0, n);
    free(odom);
}

void printluckyrec(int* odometer, int k, int n) {

    if (k == n){                                    // 1 pt
        if (numd(odometer, n, 7) >= 2)              // 2 pts
            print(odometer, n);                     // 1 pt
        return;                                     // 1 pt (or an else...)
    }

    for (int i=0; i<10; i++) {                      // 1 pt
        odometer[k] = i;                            // 1 pt
        printluckyrec(odometer, k+1, n);            // 3 pts
    }
}

int numd(int* odometer, int n, int d) {
    int res = 0;
    for (int i=0; i<n; i++)
        res += (odometer[i] == d);
    return res;
}

void print(int* odometer, int n) {
    for (int i=0; i<n; i++)
        printf("%d ", odometer[i]);
    printf("\n");
}
```

**2)** (10 pts) ALG (Sorting)

Consider tracing through a Merge Sort of the array below. In the process of the code running, 10 merge operations occur. Assume that when a subarray of odd size gets split into two arrays for recursive calls, the left array is smaller than the right. (So, when the very first recursive call occurs, it will call the subarray storing the five leftmost elements.) **Show the contents of the whole array RIGHT AFTER each of the 10 merge operations occurs.** Please show your result in the order that the Merge operations occur. Note: Please think carefully about the actual order each Merge operation occurs. **PLEASE FILL IN ALL SLOTS EVEN IF THEY DON'T CHANGE BETWEEN ITERATIONS!!!** (1 pt is awarded for every row that is 100% perfect, plus one bonus point for getting all of the rows.)

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Orig | 13 | 18 | 19 | 11 | 2 | 7 | 4 | 15 | 6 | 12 | 1 |
| 1st Merge | 13 | 18 | 19 | 11 | 2 | 7 | 4 | 15 | 6 | 12 | 1 |
| 2nd Merge | 13 | 18 | 19 | 2 | 11 | 7 | 4 | 15 | 6 | 12 | 1 |
| 3rd Merge | 13 | 18 | 2 | 11 | 19 | 7 | 4 | 15 | 6 | 12 | 1 |
| 4th Merge | 2 | 11 | 13 | 18 | 19 | 7 | 4 | 15 | 6 | 12 | 1 |
| 5th Merge | 2 | 11 | 13 | 18 | 19 | 7 | 4 | 15 | 6 | 12 | 1 |
| 6th Merge | 2 | 11 | 13 | 18 | 19 | 4 | 7 | 15 | 6 | 12 | 1 |
| 7th Merge | 2 | 11 | 13 | 18 | 19 | 4 | 7 | 15 | 6 | 1 | 12 |
| 8th Merge | 2 | 11 | 13 | 18 | 19 | 4 | 7 | 15 | 1 | 6 | 12 |
| 9th Merge | 2 | 11 | 13 | 18 | 19 | 1 | 4 | 6 | 7 | 12 | 15 |
| 10th Merge | 1 | 2 | 4 | 6 | 7 | 11 | 12 | 13 | 15 | 18 | 19 |

**Grading: 1 pt per row, row has to be completely correct to get the point. Give the last (10th) point if all 9 rows are correct. (Possible scores are 0,1,2,3,4,5,6,7,8 and 10.)**

**3)** (5 pts) ALG (Base Conversion)

Convert $32456_{10}$ to base 16, using A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15. Put a box around your final answer.

16 | 32456
16 | 2028 R 8
16 | 126 R 12
16 | 7 R 14
16 | 0 R 7

Reading the remainders in reverse order and replacing with letters as needed, we get:

$32456_{10}$ = **$7EC8_{16}$**

**Grading:** **0 pts if incorrect process is used.**

**Take 1 pt off per arithmetic error, capping at 5 errors.**