Computer Science Foundation Exam

August 23, 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	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 (Dynamic Memory Management in C)

Consider the following typedef struct definition that represents a basic bank checking account.

```
//struct representing basic checking account
typedef struct {
    char * name;
    double amount;
    int id;
} account_t;
```

Complete the following user defined function `mergeAccounts`. The function will combine two accounts (that were already declared with proper information) into one account. Please note that both structs `acct1` and `acct2` were both dynamically allocated in the heap space. You will need to properly allocate a new struct in the heap space that will store the combined information. That means the following is going to happen:

- The names will be merged with `acct1` name followed by `acct2`. Both names will be separated with the word "and". Example let's say `acct1` name is "Sonic" and `acct2` is "Amy". That means the new account name would be "Sonic and Amy". Make sure there's enough space to hold the updated name with the word "and", including the two spaces.
- The amounts will be summed together.
- The id of the merge account will be the original id of `acct2`.
- No changes should be made to `acct1` and `acct2`.

The function returns the heap address of the **new** struct.

```
account_t * mergeAccounts (account_t * acct1, account_t * acct2) {

    // Grading: 2 pts
    account_t * newacct = malloc(sizeof(account_t));

    // Grading: 5 pts (1 pt per line)
    int stringSize = strlen(acct1->name) + strlen(acct2->name) + 6;
    newacct->name = malloc(sizeof(char) * stringSize);
    strcpy(newacct->name, acct1->name);
    strcat(newacct->name, " and ");
    strcat(newacct->name, acct2->name);

    // Grading: 2 pts (1 pt per line)
    newacct->amount = acct1->amount + acct2->amount;
    newacct->id = acct2->id;

    // Grading: 1 pt
    return newacct;
}
```

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 and an integer.

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

node_t* mysterious(node_t * head, int k){
     if (k == 0 || head == NULL)
        return head;

    for (int i = 0; i < k; ++i){
        node_t *curr = head;
        while (curr->next != NULL)
            curr = curr->next;

        curr->next = head;
        curr = curr->next;
        head = head->next;
        curr->next = NULL;
    }

    return head;
}
```

If we call head = mysterious(head, 3); on the following list, show the list after the function has finished.

head -> 6 -> 8 -> 4 -> 7 -> 5? Please fill in the designated slots below.

head → 7 → 5 → 6 → 8 → 4

Grading: Full credit for a correct answer.
3/5 for a cyclic rotation off by one (4,7,5,6,8 or 5,6,8,4,7)
Otherwise just give 1 pt per correct slot

3) (10 pts) DSN (Queues)

We are going to simulate a simple battle game involving players in a queue. The battle game has the following rules.

- The game involves several battles. Each battle will involve the two front players in the queue.
- The winner of the battle is determined by figuring out who has more hp. In the case that both players have the same hp, the **second player** in line is the winner of the round.
- The loser of the game is removed completely and the winner is added to the back of the queue.
- The game is over when there is 1 player remaining in the queue.

Complete the following function `battleGame` that simulates this game. You are provided the following structure definition and helper functions that have been implemented already. You **must** utilize them reasonably for full credit. The function should print the name of the player that won the game. **It is guaranteed that the queue pointed to by gameQueue has at least one player in it.**

```
typedef struct player_s {
    char * name;
    int hp;
    struct player_s* next;
} player_t;

typedef struct {
    player_t* front;
    player_t* back;
    int size ;
} queue_t;

void enqueue(queue_t* gameQueue, player_t* player);//enqueues player to gameQueue
int size(queue_t* gameQueue) ; // returns # of elements in gameQueue
player_t* dequeue(queue_t* gameQueue); // removes front node, returns ptr to it
player_t* front(queue_t * gameQueue); // returns pointer to front node.
void deletePlayer(player_t * player);  // frees memory pointed to by player

void battleGame(queue_t * gameQueue) {

      while(size(gameQueue) > 1) {                    // 1 pt
            player_t * p1 = dequeue(gameQueue);       // 1 pt
            player_t * p2 = dequeue(gameQueue);       // 1 pt

            if(p1->hp > p2->hp) {                     // 1 pt
                  deletePlayer(p2);                   // 1 pt
                  enqueue(gameQueue, p1);             // 1 pt
            }
            else {
                  deletePlayer(p1);                   // 1 pt
                  enqueue(gameQueue, p2);             // 1 pt
            }
      }
      player_t * winner = front(gameQueue);           // 1 pt
      printf("%s is the winner.\n", winner->name);    // 1 pt
}
```

Computer Science Foundation Exam

August 23, 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	5	DSN	
2	10	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 <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) (5 pts) DSN (Binary Trees)

A binary tree is stored using the struct definition below. Write a **recursive** function that returns the sum of the data values stored in the tree pointed to by the parameter root.

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

int sumData(treenode* root) {

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

    // 4 pts - 1 pt each part
    return root->data + sumData(root->left) + sumData(root->right);

}
```

2) (10 pts) ALG (Heaps)

Consider a min-heap represented below in its array representation (root stored in index 1):

Index	1	2	3	4	5	6	7	8
Value	6	33	9	41	36	12	100	50

(a) (3 pts) Insert the value 32 into this heap and show its new array representation. (Note: feel free to jot down the tree version before copying your answer over into the array shown below.)

Index	1	2	3	4	5	6	7	8	9
Value	6	32	9	33	36	12	100	50	41

Grading: 1 pt for keeping 6, 9, 36, 12, 100 and 50 in the same place
1 pt for placement of 32
1 pt for placement of both 33 and 41

(b) (3 pts) Insert the value 3 into the heap **AFTER part (a) is done** and show its new array representation.

Index	1	2	3	4	5	6	7	8	9	10
Value	3	6	9	33	32	12	100	50	41	36

Grading: 1 pt for keeping 9, 33, 12, 100, 50 and 41 in th same place
1 pt for placement of both 3 and 6
1 pt for placement of both 32 and 36

(c) (4 pts) Delete the minimum value in the heap **AFTER part (b) is done** (this is 3) and show its new array representation.

Index	1	2	3	4	5	6	7	8	9
Value	6	32	9	33	36	12	100	50	41

Grading: 1 pt for keeping 9, 33, 12, 100, 50 and 41 in the same place
1 pt for placement of 6
1 pt for placement of 32
1 pt for placement of 36

3) (10 pts) DSN (Tries)

Assume that a dictionary of words is already stored in a trie with the struct shown below. Complete the function below so that it prints out ALL of the words stored in the dictionary of length LENGTH (constant) in alphabetical order. (Note: students often get tripped up about when to use the NULL character, so that code's been provided for you. A correct response won't require ever writing '\0' in the space you're given.) The necessary support code is given below.

```
#define LENGTH 10

typedef struct TrieNode {
    struct TrieNode *children[26];
    int isWord; // 1 if the string is in the trie, 0 otherwise
} TrieNode;

int main() {

    TrieNode* root = init();
    // Fill trie rooted at root here.

    // Set up code before recursive function call.
    char buffer[100];
    buffer[LENGTH+1] = '\0';
    printWordsLenN(root, 0, buffer, LENGTH);

    freeTree(root);
    return 0;
}

// Prints all words in the trie rooted at root of length n.
void printWordsLenN(TrieNode* root, int k, char* buffer, int n) {

    if (root == NULL) return;

    // Fill in code here.
    if (k == n) {

        if (root->isWord)                  // 1 pt
            printf("%s\n", buffer);        // 1 pt
        return;                            // 1 pt

    }

    // And there is some code to write here as well.

    for (int i=0; i<26; i++) {                                  // 1 pt
        buffer[k] = (char)('a'+i);                              // 3 pts
        printWordsLenN(root->children[i], k+1, buffer, n);      // 3 pts
    }


}
```

Computer Science Foundation Exam

August 23, 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	10	ANL	
2	10	ANL	
3	5	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, stdio, math, string) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

What is the Big-Oh run-time of the following function in terms of BOTH **n** and **k**? Carefully explain your reasoning. You may assume that all integers stored in values are positive and that the sum of the values of the form 2^k/values[i] is greater target. **Note: most of the credit will be due to the reasoning, guesses at the correct answer without any justification will only receive 1 point.**

```
int f(int* values, int n, int k, long long target) {

    int lo = 0;
    int hi = (1<<k);

    while (lo < hi) {
        int mid = (lo+hi)/2;
        long long tmp = 0;
        for (int i=0; i<n; i++)
            tmp += mid/values[i];

        if (tmp < target)
            lo = mid+1;
        else
            hi = mid;
    }
    return lo;
}
```

The outer loop is a binary search structure with initial values of lo and hi set to 0 and 2^k, respectively. The number of times this loop structure will run is lg (2^k) since each loop iteration the difference between lo and hi gets divided by 2. Since the log base is 2, lg (2^k) = k.

Luckily, inside the main loop, the code takes the same amount of time to run each time and is dominated by the for loop, which runs n times. There are a fixed number of statements outside of the for loop, so these take O(1) time. Since we repeat this for loop k times, it follows that the run time of this function in terms of n and k is **O(nk).**

Grading:

6 pts for recognizing that the outer loop runs k times. 2 pts for recognizing that the difference between lo and hi initially is 2^k, 2 pts for noticing the binary search type structure, 2 pts for applying the log rule to get to the conclusion that this code runs k times.

2 pts for recognizing that inside the while loop, the set of statements takes O(n) time dominated by the for loop.

1 pt for multiplying and 1 pt for the answer O(nk).

2) (10 pts) ANL (Algorithm Analysis)

An algorithm to process n entries of data runs in $O(n\sqrt[3]{n})$ time. For $n = 40^3$, the algorithm takes 2.5 milliseconds to run. How long will the algorithm take to run for an input size of $n = 80^3$, in milliseconds?

Let $T(n) = cn\sqrt[3]{n}$ be the amount of time the algorithm takes to process n entries of data. It follows that

$$T(40^3) = c40^3\sqrt[3]{40^3} = c(40^3 \times 40) = 2.5ms$$

$$c = \frac{2.5ms}{40^4}$$

Now, we must determine $T(80^3)$.

$$T(80^3) = c80^3\sqrt[3]{80^3} = \frac{2.5ms}{40^4}(80^3 \times 80) = (2.5ms) \times \frac{80^4}{40^4} = (2.5ms) \times \left(\frac{80}{40}\right)^4 = 2.5(16)ms = \mathbf{40ms}$$

Note: Some students interpreted the question with the run time of as $O(n^3\sqrt{n})$. If it's clear that this is how they interpreted, here is the corresponding work (grading criteria is nearly identical):

$$T(40^3) = c(40^3)^3\sqrt{40^3} = c\left(40^9 \times 40\sqrt{40}\right) = 2.5ms$$

$$c = \frac{2.5ms}{40^{10.5}}$$

Now, we must determine $T(80^3)$.

$$T(80^3) = c(80^3)^3\sqrt{80^3} = \frac{2.5ms}{40^{10.5}}(80^{10.5}) = (2.5ms) \times \frac{80^{10.5}}{40^{10.5}} = (2.5ms) \times \left(\frac{80}{40}\right)^{10.5} = 2.5(2^{10})(\sqrt{2})ms$$
$$= \mathbf{2560\sqrt{2}ms}$$

Grading: 1 pt set up equation for c
3 pts get to RHS = $2.5/40^4$ or $2/5/40^{10.5}$ or equivalent
1 pt to get to c
1 pt plug in 80^3
4 pts to simplify to correct answer (give partial as needed)

3) (5 pts) ANL (Summations)

Determine the following summation in terms of n, in factorized form. (Do NOT multiply the answer out into polynomial form. Note: Your answer should NOT have a fraction in it.)

$$\sum_{i=1}^{2n} i^3$$

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

Grading: 2 pts correctly plug into formula
2 pts expand $(2n)^2$
1 pt cancel with 4 and write final answer

Computer Science Foundation Exam

August 23, 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	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 <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 (Recursive Coding)

In a 2D video game, one can spray a mist from a cell from the top of the game board. The mist will spread recursively to up to 3 cells below it: left, center and right, unless there is a mist blocker at that cell. Here is a before and after picture of activating the mist from cell row 0 cell 3. The letter 'M' is used for mist and 'B' for blocker:

			M			
		B				
			B		B	

			M			
		B	M	M		
		M	B	M	B	
	M	M	M	M	M	

Complete the code below so that it **recursively** activates the mist from row r, column c. You may assume the array grid has numR rows and numC columns. (Note: The diagram above has 4 rows and 7 columns.) To activate the mist, the recursive function places the character 'M' in the spot r, c (unless there's already a B there, in which nothing happens), and then recursively mists the three spots below it. For convenience, both DR,CD arrays and an inbounds function have been provided.

```
#define NUMDIR 3
const int DR[3] = {1,1,1};
const int DC[3] = {-1,0,1};

void mistRec(char** grid, int r, int c, int numR, int numC) {

    // Don't do anything if (r,c) is out of bounds or contains 'B'
    if (!inbounds(r,c, numR, numC)) return;     // 2 pts
    if (grid[r][c] == 'B') return;              // 2 pts

    // Put the mist in location r, c
    grid[r][c] = 'M';                           // 1 pt

    // Mist below.

    for (int i=0; i<3; i++)                                 // 1 pt
        mistRec(grid, r+DR[i], c+ DC[i], numR, numC);       // 4 pts



}

int inbounds(int myr, int myc, int numR, int numC) {
    return myr >= 0 && myr < numR && myc >= 0 && myc < numC;
}
```

2) (5 pts) ALG (Sorting)

Show the result after each iteration of performing **Bubble Sort** on the array shown below. Remember that the largest value is guaranteed to be in the correct spot (last index) after the first iteration of the algorithm.

Iteration	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
0	10	22	16	30	13	17	8
1	10	16	22	13	17	8	30
2	10	16	13	17	8	22	30
3	10	13	16	8	17	22	30
4	10	13	8	16	17	22	30
5	10	8	13	16	17	22	30
6	8	10	13	16	17	22	30

Grading: 1 pt per line must get whole line correct to get the point.

3) (10 pts) DSN (Bitwise Operators)

Complete the function below, using the appropriate bitwise operators when necessary, so that it returns 1 if the array, **`numbers`** (first parameter to the function) contains a subset of values that adds up exactly to **`target`** (third parameter to the function). Note that the second parameter, **`n`**, represents the length of the array **`numbers.`** If no such subset exists, then the function should return 0.

```
int subsetSum(int* numbers, int n, int target) {

    for (int i=0; i<(1<<n); i++) {

        int sum = 0;                    // 1 pt
        for (int j=0; j<n; j++)         // 2 pt
            if (i & (1<<j))             // 3 pts
                sum += numbers[j];      // 2 pts

        if (sum == target) return 1;    // 2 pts (1 pt if, 1 pt ret)

    }

    return 0;
}
```