Computer Science Foundation Exam

January 17, 2026

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)

In the Mario Kart video game series, drivers collect items (such as shells, bananas, and mushrooms) during a race. Each driver maintains a list of collected items that may grow as new items are obtained. Given the following typedef structure definitions:

```
//struct representing an item
typedef struct {
    char itemName[15];
} item_t;

//struct representing a driver
typedef struct {
    char driverName[20];
    item_t *items;
    int itemCount;
} driver_t;
```

complete the following function `addItem`. This function will store a new item that a driver collects during a race. In particular, the string inside `newItem` needs to be copied into the newly allocated memory within `driver`. The function returns 1 if the item was successfully added. Otherwise, 0 is returned. The following assumptions can be made:

- The driver can potentially have no items initially, which would be represented by the value 0 in the component `itemCount`.
- You may assume that all items inserted fit within the array size of 15 elements. There is no need to do a conditional check.
- A driver can hold at most 3 items. (addItem should return 0 if the driver already has 3 items.)

```
int addItem(driver_t *driver, const item_t *newItem) {

    // 2 pts 1 for if, 1 for return.
    if(driver->itemCount == 3)
        return 0;

    // 3 pts, 1 pt LHS and realloc, 2 pts parameters
    driver->items = realloc(driver->items, sizeof(item_t) * (driver->itemCount+1));

    // 3 pts, 1 pt strcpy, 1 pt 1st parameter, 1 pt 2nd parameter
    strcpy(driver->items[driver->itemCount].itemName, newItem->itemName);

    // 1 pt
    driver->itemCount++;

    // 1 pt
    return 1;
}
```

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 val;
     struct node_s * next;
}node_t;

node_t *mystery(node_t *head) {

    node_t *second, *rest, *tail;

    if(head == NULL || head->next == NULL)
        return head;

    second = head->next;
    rest = mystery(second->next);
    head->next = rest;
    tail = head;

    while(tail->next != NULL)
        tail = tail->next;

    tail->next = second;
    second->next = NULL;

    return head;
}
```

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

head → 1 → 2 → 3 → 4 → 5? Please fill in the designated slots below. (Note: The list does have five items in it after the function call executes.)

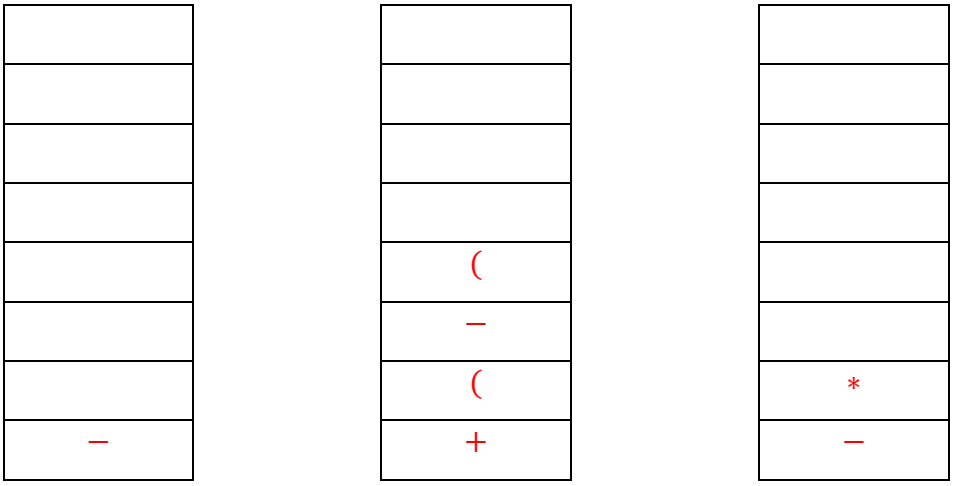
head → 1 → 3 → 5 → 4 → 2

Grading: 1 pt for each slot. Correct number must be in the correct slot to get the point.

3) (10 pts) ALG (Stack)

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
7 * (3 + 9) - 4 / 6 + (8 * 2 - (5 + 7)) / 3 - 9 * 4 + 6
```



A B C

Note: A indicates the location in the expression **AFTER** the minus operator and before the value 4. B indicates the location in the expression **AFTER** the value 5 and before the plus operator. C indicates the location in the expression **AFTER** the value 4 and before the plus operator.

Resulting postfix expression:

7	**3**	**9**	**+**	*****	**4**	**6**	**/**	**-**	**8**	**2**	*****	**5**	**7**	**+**	**-**	**3**	**/**	**+**	**9**	**4**	*****	**-**	**6**	**+**

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

Grading: 1 pt for the first stack, all or nothing
2 pts for the second stack, can give 1 pt
2 pts for the third stack, can give 1 pt
5 pts for the expression – automatic 0 if the numbers aren't in the correct relative order, if they are, then take off 1 pt for each operator you have to move to a different location to make the expression correct, cap at 0.

Computer Science Foundation Exam

January 17, 2026

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	ALG	
2	5	DSN	
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 <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) ALG (Binary Trees)

Consider the following binary search tree and the `printMystery` function shown below. What would be printed by the function if we pass the root of the following tree? Please place each of the ten numbers that get printed in the order they get printed in the blanks provided at the bottom of the page.

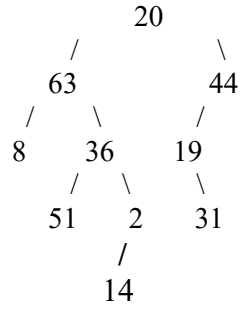
```
typedef struct treenode {
    int data;
    struct treenode *left;
    struct treenode *right;
} treenode;

void printMystery(treenode* root) {

    if (root == NULL)
        return;

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

Tree:



44 31 19 20 2 14 36 51 63 8

Grading: 1 pt per correct number in the correct slot, no exceptions

2) (5 pts) DSN (Hash Tables)

Consider the following hash function for a string, s, of lowercase letters, where value('a') = 1, value('b') = 2, ..., value('z') = 26, and the length of the string is n.

f(s, m) = (value(s[0]) x 27^0 + value(s[1]) x 27^1 + value(s[2]) x 27^2 + ... value(s[n-1]) x 27^{n-1}) mod m.

Complete the function below so that it computes this hash function. **Do not call the pow function.** (Any solution with a call to the pow function will get an automatic 0.) Remember all computations must occur "under mod." You may assume that the value of m is small enough that if coded appropriately no overflow errors will occur.

```
int f(char* s, int m) {

    int res = 0;
    int pow27 = 1;
    int len = strlen(s);

    for (int i=0; i<len; i++) {

        // Update res to equal the running value of the hash function
        // so far.

        res = (res + pow27*(s[i]-'a'+1))%m;

        // Update pow27 to be the current power of 27 under mod.

        pow27 = (pow27*27)%m ;

    }

    return res;
}
```

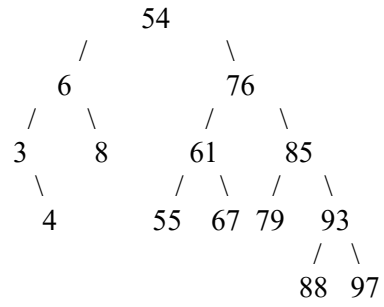
Grading: first line = 3 pts, 1 pt pow27 times, and add to res,
1 pt for (s[i] – 'a' + 1), can give pt if forgot +1
1 pt for mod

second line = 2 pts (1 pt mult, 1 pt mod)

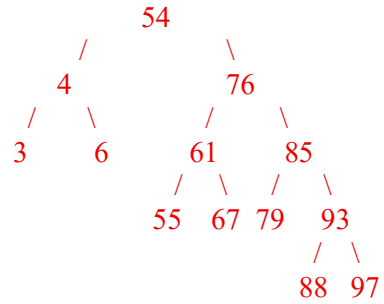
3) (10 pts) ALG (AVL Trees)

Consider the following AVL tree. Delete 8 from the tree and show the final resulting AVL tree. In the process of the delete, the tree gets restructured twice. Draw a box around the full tree at the following stages of the process:

(a) (5 pts) Right after the first restructuring takes place.
(b) (5 pts) At the end of the process, right after the second restructuring takes place.

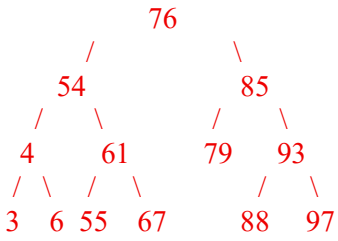


(a) Picture after first restructure



Grading: 1 pt for location of 4, 1 pt location 3, 1 pt location 6, 1 pt location 54, 1 pt rest intact

(b) Picture after second restructure



Grading: 1 pt for location of 54, 1 pt location 76, 1 pt location 85, 1 pt everything below 54, 1 pt everything below 85

Computer Science Foundation Exam

January 17, 2026

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 <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) ANL (Algorithm Analysis)

Let T(n, m) be the run-time of the following function, in terms of input parameters n and m. **Write down a recurrence relation that T(n, m) satisfies.**

```
int f(int* values, int n, int m) {

    if (n == 1) return values[0];

    int left = f(values, n/2, m-1);
    int right = f(values+n/2, n/2, m-1);

    while (m > 0) {
        if (left > right)
            left--;
        else
            right++;
        m--;
    }

    return 2*(left+right);
}
```

Both recursive values take in a second parameter equal to n/2 and a third parameter equal to m-1. Thus, if we let T(n, m) be the run-time of the function above, it follows that each of the recursive calls take T(n/2, m-1) time. Following the recursive calls there is a loop. The body of the loop takes O(1) time and executes exactly the m times. (Notice that no matter what, m is decremented by 1 each time and we exit the loop when m = 0. Thus, the run-time of the loop is O(m). It follows that the desired recurrence relation is:

$T(n, m) = 2T(n/2, m-1) + O(m)$

Grading:

All credit is just based on the recurrence relation written.

+1 for writing T(n, m) on the LHS
+3 for writing 2T(n/2, m-1) on the RHS (+1 for 2, +1 for n/2, +1 for m-1)
+1 for writing + O(m)
Take off points as necessary if components are incorrectly combined.

2) (10 pts) ANL (Algorithm Analysis)

An algorithm which has a run time of $O(n^2\sqrt{n})$ takes 3 seconds to run on an input with size n = 10,000. (Note: the function in the Big-Oh is read out loud as, "n squared times square root n.") If there are 86,400 seconds in a day, how many days would the algorithm take to complete on an input size of $n = 10^6$? **Express your answer as a fraction in lowest terms.** Put a box around your final answer.

Let $T(n) = c(n^2)(\sqrt{n})$ be the amount of time the algorithm takes on input size n. Then, we have

$$T(10^4) = c(10^4)^2\sqrt{10^4} = c(10^8 \times 10^2) = 3\ sec$$

$$c = \frac{3\ sec}{10^{10}}$$

Now, we must determine $T(10^6)$.

$$T(10^6) = c(10^6)^2\sqrt{10^6} = \frac{3\ sec}{10^{10}}(10^{12} \times 10^3) = (3\ sec) \times \frac{10^{15}}{10^{10}} = 300{,}000\ sec.$$

We must convert this to days. Use the factor given to determine:

$$300{,}000\ sec \times \frac{1\ day}{86400\ sec} = \frac{3000}{864}\ days = \frac{375}{108}\ days = \frac{125}{36}\ days$$

Grading: 1 pt set up equation for c
3 pts get to c = $3/10^{10}$ sec or equivalent
1 pt plug in 10^6 into equation with known c.
3 pts to get to 300,000 sec
2 pts to get to a fraction in lowest terms for days. (1 pt for any correct fraction that isn't fully reduced.)

3) (10 pts) ANL (Recurrence Relations)

Determine a closed form solution to the following recurrence relation, in terms of n. (Your solution must be an exact function in terms of n, not a Big-Oh bound.)

$$T(n) = 3T(n-1) + 3^n, \text{ for integers n > 1}$$
$$T(1) = 12$$

Use the iteration technique:

$$\begin{aligned} T(n) &= 3T(n-1) + 3^n \\ &= 3(3T(n-2) + 3^{n-1}) + 3^n \\ &= 9T(n-2) + 3^n + 3^n \\ &= 9T(n-2) + 2(3^n) \\ &= 9(3T(n-3) + 3^{n-2}) + 2(3^n) \\ &= 27T(n-3) + 3^n + 2(3^n) \\ &= 27T(n-3) + 3(3^n) \end{aligned}$$

In general, after k iterations, we have:

$$T(n) = 3^k T(n-k) + k(3^n)$$

Since T(1) is known, substitute k = n – 1 in the general form above to yield:

$$\begin{aligned} T(n) &= 3^{n-1}T\big(n-(n-1)\big) + (n-1)(3^n) \\ T(n) &= 3^{n-1}T(1) + (n-1)(3^n) \\ T(n) &= 12(3^{n-1}) + (n-1)(3^n) \\ T(n) &= 4(3^n) + (n-1)(3^n) \\ T(n) &= (n-1+4)(3^n) \\ \boldsymbol{T(n)} &\boldsymbol{= (n+3)(3^n)} \end{aligned}$$

Grading: 1 pt writing recurrence
1 pt second iteration
1 pt third iteration
2 pts general form
1 pt plug in k = n – 1
1 pt plugging in T(1) = 12
3 pts to get to final answer (also accept $n3^n + 3^{n+1}$), give partial as needed

Computer Science Foundation Exam

January 17, 2026

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	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 (Recursive Coding)

Consider the problem of cutting a chocolate bar that is **length** inches long and **width** inches wide into multiple 1 inch by 1 inch squares. You may either cut the bar horizontally or vertically to create two rectangular bars that are an integer number of inches in length and width and are as close to equal size as possible. For example, a bar that is 8 inches long and 3 inches wide can be cut into two bars that are both 4 inches long and 3 inches wide, OR into two bars where one is 8 inches long and 1 inch wide and the other bar is 8 inches long and 2 inches wide. From there, both bars must be recursively cut. Note that if one dimension is a single inch, only a single cut is possible (reducing the larger dimension.) The cost of a cut that leaves the length unchanged is equal to **length** and a cut that keeps the width unchanged is equal to **2*width**. Write a recursive function that takes in the parameters **length** and **width** (both positive integers), and returns the minimum cost of cutting a chocolate bar with those dimensions into 1inch by 1 inch squares.

```
int minCutCost(int length, int width) {

    // Grading: 2 pts
    if (length == 1 && width == 1) return 0;

    // Grading: 1 pt but these aren't necessary if you have an if
    // elsewhere
    if (length == 1) return width-1;
    if (width == 1) return 2*(length-1);

    // Grading: 2 pts
    int cutLen = minCutCost(length/2, width) +
                 minCutCost(length-length/2, width) + 2*width;

    // Grading: 2 pts
    int cutWid = minCutCost(length, width/2) +
                 minCutCost(length, width-width/2) + length;

    // Grading: 3 pts total 1 pt comparison, 1 pt for each return
    if (cutWid < cutLen)
        return cutWid;
    return cutLen;
}
```

Note: For this cost function, it turns out that no matter how you make your cuts, the cost is the same. But the purpose of this question was to test the idea of recursion, test the idea of trying out both cuts and taking the answer that is smaller, so for that reason, the grading criteria shown above was applied instead of giving full credit to other answers that might turn out to work. Here is an equivalent function, mathematically:

```
int minCutCostAlt(int length, int width) {
    return 2*(length-1)*width + (width-1)*length;
}
```

2) (10 pts) DSN (Sorting)

Although the example code traditionally shown for the Bubble Sort is iterative, the algorithm itself lends itself easily to recursion. (After one pass of Bubble Sort on an array of size n, the work that remains is a problem of the exact same nature.) Write a **recursive** implementation of the Bubble Sort algorithm that sorts elements in ascending order in the function shown below.

```
void bubbleSortRec(int* array, int n) {

    if (n == 1) return;                  // Grading: 1 pt

    for (int i=0; i<n-1; i++) {          // Grading: 1 pt, must be n-1
        if (array[i]>array[i+1]) {       // Grading: 2 pts
            int tmp = array[i];          // Grading: 1 pt
            array[i] = array[i+1];       // Grading: 1 pt
            array[i+1] = tmp;            // Grading: 1 pt
        }
    }

    bubbleSortRec(array, n-1);           // Grading: 3 pts
}
```

Grading Note: There are probably a few valid interpretations of the question and a few different ways to implement one pass of the Bubble Sort followed by a recursive call. Use discretion as necessary.

3) (5 pts) ALG (Base Conversion)

Convert each of the following binary numbers into base 16 (Hexadecimal). No need to show your work, credit will be based solely on the answers.

(a) `10010110`

For each of these, break the bits into blocks of size four from right to left. If there are fewer than 4 bits in the last block, pad to the left with 0s as necessary. Then, convert each block of 4 bits to hexadecimal. (These should just be memorized...)

`1001 0110`

`96`

(b) `11001111`

`1100 1111`

`CF`

(c) `101101`

`0010 1101`

`2D`

(d) `10111101011`

`0101 1110 1011`

`5EB`

(e) `1001101101000010000`

`10 0110 1101 0001 0000`

`26D10`

Grading: 1 pt for each one, must be completely correct to get the point.