Computer Science Foundation Exam

May 16, 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	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 (Dynamic Memory Management in C)

In Super Mario Galaxy, Mario follows a route of planets, and each planet has a known number of Power Stars that Mario can collect. You are given an array route of length `routeLen` that stores planet identifiers, and an implemented helper function `getStarsOnPlanet(int planetId)` that returns the number of Power Stars on a given planet. Write the function `buildOddStarLog` that dynamically allocates an integer array containing the number of Power Stars for each planet in route whose star count is odd, in the same order they appear in the route. The function must store a sentinel value -1 immediately after the last valid array element, set `*outSize` to the number of valid elements stored (not including the sentinel), and return a pointer to the allocated array. If dynamic memory allocation fails, the function must return NULL and set `*outSize` to 0. You may assume that route and `outSize` are not NULL and that `routeLen` > 0.

```
int getStarsOnPlanet(int planetId); //helper function implemented

int* buildOddStarLog(int* route, int routeLen, int* outSize)
{
    int cnt = 0;

    for(int i = 0; i < routeLen; i++)
       if(getStarsOnPlanet(route[i]) % 2 == 1)
          cnt++;

    int* a = malloc((cnt + 1) * sizeof(int));
    if(a == NULL)
    {
       *outSize = 0;
       return NULL;
    }

    int k = 0;
    for(int i = 0; i < routeLen; i++)
    {
        int s = getStarsOnPlanet(route[i]);
        if(s % 2 == 1)
           a[k++] = s;
    }

    a[k] = -1;
    *outSize = k;
    return a;
}
```

Grading: 2 pts – allocating & reallocating int array and returning it
2 pts – returning NULL setting *outSize = 0 in failed mem alloc case
1 pt – loop through route array
1 pt – check odd
2 pts – copy odd values and maintain index into new array
2 pts – copy sentinel value into return array

2) (10 pts) DSN (Linked Lists)

In the Harry Potter series, students at Hogwarts belong to one of four houses. Suppose each student entering the Great Hall is recorded in a singly linked list, where each node stores the student's house and the number of house points they received. You are given the following linked-list node definition, where the string house stores the student's house name:

```
typedef struct node_s {
    char house[20];
    int points;
    struct node_s* next;
} node_s;


int getHousePoints(node_s* head, char* houseName) {

    int res = 0;                                        // 1 pt

    while(head != NULL) {                               // 1 pt

        if (strcmp(head->house, houseName) == 0)        // 3 pts
            res += head->points;                        // 2 pts

        head = head->next;                              // 1 pt
    }

    return res;                                         // 1 pt
}
```

3) (5 pts) ALG (Queues)

For the following question, assume an efficient array implementation of a queue with a fixed array size of 10, where the first element added to the queue gets added to index 0. Below is a segment of code for 135 enqueue and dequeue operations on a queue (variable name q). (Code unnecessary to answer the question is omitted.) List the indexes into the array which the last 5 operations on the queue use. (So, for enqueues, list the index that the element enqueued is added and for dequeues, list the index from which the dequeued element is returned.)

```
struct queue *q = malloc(sizeof(struct queue));
initialize(&q);
for (int i=0; i<2; i++) {
    for (int j=0; j<9; j++)
        enqueue(q, i*9+j);
    for (int j=0; j<8; j++)
        printf("%d\n", dequeue(q));
}

for (int i=3; i<9; i++) {
    for (int j=0; j<8; j++)
        enqueue(q, i*9+j);
    for (int j=0; j<8; j++)
        printf("%d\n", dequeue(q));
}

enqueue(q, 200);                    // 1st answer based on this line
printf("%d\n", dequeue(q));         // 2nd answer based on this line
printf("%d\n", dequeue(q));         // 3rd answer based on this line
enqueue(q, 201);                    // 4th answer based on this line
printf("%d\n", dequeue(q));         // 5th answer based on this line
```

Index into array for Operation #131: **6**

Index into array for Operation #132: **4**

Index into array for Operation #133: **5**

Index into array for Operation #134: **7**

Index into array for Operation #135: **6**

Grading: 5 pts, if all correct,
2 pts, if pattern of x, x-2, x-1, x+1, x occurs under mod 10 with x≠6,
otherwise 0. (Thus only scores you can record on this question are 0, 2 or 5.)

Computer Science Foundation Exam

May 16, 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	DSN	
2	10	DSN	
3	5	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) (Binary Trees)

The *subtree size* of a node in a Binary Search Tree (BST) is defined as the total number of nodes in the subtree rooted at that node, including the node itself.
You are given the following node structure:

```
typedef struct node {
    int data;
    int size; // stores subtree size
    struct node* left;
    struct node* right;
} node;
```

Write a recursive function **`updateSubtreeSize(node* root)`** that updates the size field of every node in the BST so that it correctly reflects the number of nodes in its subtree **and returns this size.** You are not allowed to write any helper function.

```
int updateSubtreeSize(node* root) {
    if (root == NULL)
        return 0;

    int leftSize = updateSubtreeSize(root->left);
    int rightSize = updateSubtreeSize(root->right);

    root->size = leftSize + rightSize + 1;

    return root->size;
}
```

```
Another version:
int updateSubtreeSize(node* root) {
    if (!root) return 0;

    root->size = 1
               + updateSubtreeSize(root->left)
               + updateSubtreeSize(root->right);

    return root->size;
}
```

Grading (10 pts):
2 pts – Checking base case and return 0
2 pts – Logic to traverse the left side and get the size from left
2 pts – Logic to traverse the right side and get the size from right
2 pts – Add the left and right side
2 pts –Return the current size

2) (10 pts) (DSN) (Heap)

Consider an implementation of a heap of elements of the type MysteryStruct. Using the HeapStruct definition given below, write the percolateUp function **recursively**, for the heap struct below. Notes: recall that this function is called when inserting a new item into the heap, the root of the heap is stored in index 1, not index 0, and the heap implemented is a minimum heap.

```
typedef struct HeapStruct{
    MysteryStruct** heaparray; // Stores pointers to each MysteryStruct in heap.
    int capacity; // maximum elements the heap can store. Length of heaparray is
                  // capacity+1.
    int curSize; // current number of elements in the heap.
} HeapStruct;
```

You may assume that curSize has already been updated (do NOT change this in your code) and that the value of index initially passed to the percolateUp function is within the bounds of the heaparray of the struct.

You have access to a compare function with the following function prototype:

```
// Returns a negative integer if the struct pointed to by ptrA comes
// before the struct pointed to by ptrB, a positive integer if the
// struct pointed to by ptrA comes after the struct pointed to ptrB,
// and 0 otherwise.
int compare(MysteryStruct* ptrA, MysteryStruct* ptrB);

// Recursively percolates up the item stored at index in the heap pointed to by
// heapPtr.
void percolateUp(HeapStruct* heapPtr, int index) {

    // 2 pt for this base case.
    if (index == 1)
        return;

    // 3 pts for this base case. Give full credit if they have >.
    // Can also be < if parameters flipped...
    if ( compare(heapPtr->heaparray[index], heapPtr->heaparray[index/2]) >= 0 )
        return;

    // 3 pts for the swap, give partial as you see fit.
    MysteryStruct* tmp = heapPtr->heaparray[index];
    heapPtr->heaparray[index] = heapPtr->heaparray[index/2];
    heapPtr->heaparray[index/2] = tmp;

    // 2 pts - 1 pt rec call and h, 1 pt for second parameter.
    percolateUp(heapPtr, index/2);

}
```

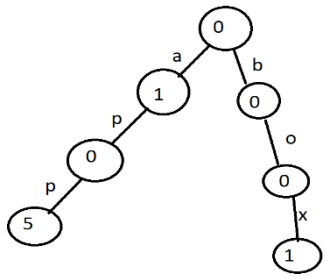
Grading Note: Maximum grade of 5 out of 10 for an iterative solution, give points proportionally up to 5.

3) (5 pts) (Trie)

Given the following trie node structure. The following ***del()*** function takes the root of a trie, a word to delete, and an int k=0. It removes **one occurrence** of the word from the trie and deletes any unnecessary nodes during the process. The del function also calls ***isEmpty()*** function, which returns 1 if the node has no children and 0 otherwise.

However, there are bugs in the ***del()*** function. For the trie shown on the right, deleting the word ***"box"*** once correctly frees all associated memory. But if we try to delete "box" again, the program crashes due to invalid memory access.

Identify the bugs and modify or add the necessary code on top of the existing implementation to fix this issue. [Hint: you just need to change in two places]




```
struct trie {
    int count;//count of the word ending at this node
    struct trie* next[26];
};

struct trie* del(struct trie* r, char word[], int k)
{
    if(r==NULL)
        return NULL;

    if(word[k] == '\0')
    {
        r->count = r->count - 1;

        if(isEmpty(r) && r->count==0)
        {
            free(r);
            r = NULL;  // 3 pts for this one.
        }
        return r;
    }

    int nextIdx = word[k] - 'a';
    r->next[nextIdx] = del(r->next[nextIdx], word, k+1);

    if(isEmpty(r) && r->count == 0)
    {
        free(r);
        r = NULL;   // 2 pts for this one.
    }
    return r;
}
```

Grading Note: No deduction for incorrect marking unless they have marked more than 2 changes. If they have marked more than 2, then give any credit for the correct markings and subtract 1 total for having incorrect ones.

Computer Science Foundation Exam

May 16, 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	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, stdio, math, string) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

Here is a function which takes in an integer array and an integer, k. The array is size $n = 2^k$, exactly. The function returns an array of arrays. Note: An auxiliary function, min, is also provided.

```
int min(int a, int b) { return a < b ? a : b; }

int** f(int* array, int k) {

    int** res = calloc(k+1, sizeof(int*));
    for (int i=0; i<k+1; i++) {

        int sz = (1<<(k-i));
        res[i] = calloc(sz, sizeof(int));

        if (i == 0) {
            for (int j=0; j<sz; j++) res[i][j] = array[j];
        }
        else {
            for (int j=0; j<sz; j++)
                res[i][j] = min(res[i-1][2*j], res[i-1][2*j+1]);
        }
    }
    return res;
}
```

(a) (8 pts) With proof, determine the **Big-Oh** runtime (tight bound) of f **in terms of n**, the size of the input array.

The outer loop runs k+1 times. Recall that since $n = 2^k$, it follows that $k = \log_2 n$, by definition of log. Unfortunately, the inner loop runs different amounts of times, so this observation turns out to be unimportant in gauging the run-time of the code segment. The more important observation is that the inner loop always runs sz number of times, regardless of whether we're in the if or the else. Looking at the code, $sz = 2^{k-i}$, where i ranges from 0 to k, inclusive. It follows that we can set up a summation in terms of k equal to the run-time of the function: $\sum_{i=0}^{k} 2^{k-i}$. If we reverse the order of summation, we can rewrite this summation as $\sum_{i=0}^{k} 2^{i}$.

Finally, $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 = 2 \times 2^k - 1 = 2n - 1 = O(n)$.

Grading: 4 points for observing that the number of times the inner loop runs is successive powers of 2, 2 pts for computing the summation in terms of k, 1 pt to substitute n for k, 1 pt for the Big-Oh answer.

Partial Credit ONLY for an answer of O(nlgn) – 2 pts for the answer, 2 pts for the justification that the outer loop runs lg n times and that the inner loop runs at MOST n times. (So max credit here is 4 pts.)

(b) (2 pts) If k = 4 for a particular call to f, the function would return an array of arrays. What are the sizes of each of the arrays? (Please answer with the length of res[0], then the length of res[1], then the length of res[2], etc.)

The lengths are 16, 8, 4, 2 and 1, respectively.

Grading: 2 pts correct answer, 1 pt for powers of 2 but a different ordered list than the one above, 0 pts otherwise.

2) (5 pts) ANL (Algorithm Analysis)

A $O(2^n)$ algorithm took 625 milliseconds to complete running on a data set with $\boldsymbol{n} = 22$. How long would the expected run-time be, **in seconds**, if the algorithm was run on a different data set with $\boldsymbol{n} = 26$?

Let $T(n) = c2^n$ be the runtime of the algorithm for some constant c. Using the given information, we have:

$T(2^{22}) = c2^{22} = 625$ ms. Solving for c we get: $c = \frac{625\ ms}{2^{22}}$.

We just solve for $T(2^{26})$: $T(2^{26}) = \frac{625\ ms}{2^{22}} \times 2^{26} = (5^4 ms) \times 2^4 = 2^4 5^4 ms = 10^4 ms = \mathbf{10\ seconds}$

Grading: 1 pt set up equation for c
1 pt solve for c without simplifying
1 pt plug in 2^{26} for n
1 pt get to 10^4 ms
1 pt to convert to 10 seconds

3) (10 pts) ANL (Summation)

Determine the following summation in terms of n, **<u>in polynomial form.</u>** (An example of polynomial form is $f(n) = 3n^3 - 2n^2 + 17n + 5$.)

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

$$\sum_{i=1}^{2n}(3i^3 + 3i^2 + i) = \left(3\sum_{i=1}^{2n} i^3\right) + \left(3\sum_{i=1}^{2n} i^2\right) + \left(\sum_{i=1}^{2n} i\right)$$

$$= \frac{3(2n)^2(2n+1)^2}{4} + \frac{3(2n)(2n+1)(4n+1)}{6} + \frac{2n(2n+1)}{2}$$

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

$$= 3n^2(4n^2+4n+1) + (2n^2+n)(4n+1) + 2n^2 + n$$

$$= (12n^4 + 12n^3 + 3n^2) + 8n^3 + 6n^2 + n + 2n^2 + n$$

$$= \mathbf{12n^4 + 20n^3 + 11n^2 + 2n}$$

Grading: 1 pt – split sums
3 pts – plug into each formula so 1 pt for each individual formula
2 pts – clear all fractions correctly
2 pts – multiplying out all terms
2 pts – gathering terms to form one polynomial

Computer Science Foundation Exam

May 16, 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	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)

For this question you'll write two **recursive functions**. The first one will perform a more basic task while the second one will utilize the first by calling it. COP 3223 is famous for having students print out various triangular designs. One of these designs is ***n*** rows long with the ***i*th** row having the first ***i*** positive integers printed out in order. Here is the design for ***n*** = 4:

```
1
1 2
1 2 3
1 2 3 4
```

The first **recursive function** to write is: `printSeq(int n)`, which will print the integers from 1 to n, **with each integer, except 1, preceded by a space.** (This function will NOT print a new line.)

The second **recursive function** to write is: `printTriangle(int n)`, which will print out a triangle of n rows as specified above. This function should call printSeq AND be recursive. **You may assume that the initial value of n passed to both functions is positive.**

Fill in both functions below.

```
void printSeq(int n) {

    if (n == 1) {                   // 3 pts base case and
        printf("1");                // handling spacing.
        return;
    }

    printSeq(n-1);                  // 1 pt
    printf(" %d", n);               // 1 pt
}

void printTriangle(int n) {

    if (n == 0) return;             // 1 pt
    printTriangle(n-1);             // 2 pts
    printSeq(n);                    // 1 pt
    printf("\n");                   // 1 pt
}
```

2) (5 pts) ALG (Sorting)

(a) (2 pts) When executed on the array shown below, how many swaps does Insertion Sort perform? We define a swap for the purposes of this question to be when the values stored at **two different** array indices are exchanged.

index	0	1	2	3	4	5	6	7
value	12	3	7	5	15	10	19	9

10 (Grading: 2 pts correct answer 1 pt for 9 or 11, 0 otherwise)

(b) (2 pts) Give an example array for which the answer to the question in part (a) is maximal (as large as possible.)

index	0	1	2	3	4	5	6	7
value	100	90	80	70	60	50	40	30

Grading: 2 pts for any array in reverse order, 0 pts otherwise

(c) (1 pt) How many swaps would Insertion Sort take to swap the array you provided in part (b)?

28 (Grading: 1 pt only if correct)

3) (10 pts) DSN (Bitwise Operators)

Write a function that takes in a single positive integer, $n \leq (2^{31}-1)$, and returns a dynamically allocated array storing the bit locations that are turned on in the binary representation of n. For example, if $n = 37$, then the function should return an array of size 3 which stores {0,2,5}, since $2^5 + 2^2 + 2^0 = 37$. You should store the bits in sorted order from smallest bit locations to largest. Since the size of the returned array is unknown before the function call, your function will take in a pointer called ptrSize, which points to an integer. Before your function returns the array, store the size of the array in the integer pointed to by ptrSize. (Note: You may NOT assume that the integer ptrSize is pointing to is set to a particular value BEFORE the function call.) **Note: Assume that any necessary libraries are included.**

```
int* bitsOn(int n, int* ptrSize) {

    int* res = calloc(31, sizeof(int));       // 1 pt
    int idx = 0;                              // 1 pt

    for (int i=0; i<31; i++)                  // 1 pt
        if (n & (1<<i))                       // 1 pt
            res[idx++] = i;                   // 2 pts

    res = realloc(res, idx*sizeof(int));      // 2 pts
    *ptrSize = idx;                           // 1 pt
    return res;                               // 1 pt
}
```