

Computer Science Foundation Exam

August 29, 2026

Section A: BASIC DATA STRUCTURES

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

SOLUTIONS

Question #	Max Pts	Category	Score
1	5	DSN	
2	10	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 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) (5 pts) DSN (Dynamic Memory Management in C)

In a Mario adventure game, Bowser keeps track of the enemies guarding a level. Each enemy has a dynamically allocated name, an integer ID, and the number of coins Mario earns for defeating that enemy. Consider the following typedef struct definition:

```
typedef struct {
    char *name;
    int id;
    int coins;
} enemy_t;
```

The following segment of code allocates some memory dynamically and fills it. (You may assume that the name read in from standard input is a string comprised of 19 or fewer lower case letters.)

```
enemy_t* ptr1 = malloc(sizeof(enemy_t));
enemy_t* ptr2 = malloc(sizeof(enemy_t));
ptr1->name = calloc(20, sizeof(char));
scanf("%s%d%d", ptr1->name, &(ptr1->id), &(ptr1->coins));
ptr2->name = ptr1->name;
ptr2->id = ptr1->id+1;
ptr2->coins = ptr1->coins + 10;
```

```
// Work with the structs here.
```

Write the code below to free all of the dynamically allocated memory that was allocated in the code segment above. Each statement you write will include either `ptr1` or `ptr2`.

```
free(ptr1->name); // or free(ptr2->name);
free(ptr1);
free(ptr2);
```

Grading: 5/5 if exactly those three lines with `free(ptr1->next)` coming first
4/5 if correct three lines but `free(ptr1->next)` comes after `free(ptr1)`
3/5 if four lines are included with both `free(ptr1->name)` and `free(ptr2->name)`
2/5 if only `free(ptr1)` and `free(ptr2)`
1/5 if only 1 free, or if there are 5-7 or more lines of code, period.
0/5 otherwise

2) (10 pts) DSN (Linked Lists)

In a doubly linked list, each node has two pointers: one to the previous node in the list and one to the next node in the list. Here is a struct definition for a node in a doubly linked list:

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

Complete the function below so that it takes in a pointer to the front of a doubly linked list, **listPtr**, and an integer, **value**, and creates a new node storing value, inserts it into the front of the existing list and returns a pointer to the new front of the list. (Make sure your code works when inserting into an empty list!)

```
node_t* insertFront(node_t* listPtr, int value) {  
  
    node_t* newN = malloc(sizeof(node_t));           // 3 pts  
    newN->data = value;                               // 1 pt  
    newN->prev = NULL;                                // 1 pt  
    newN->next = listPtr;                             // 1 pt  
  
    if (listPtr != NULL)                              // 1 pt  
        listPtr->prev = newN;                         // 2 pts  
  
    return newN;                                       // 1 pt  
}
```

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.

1 * (7 - 4) / (5 + ^A 6) + 9 * 2 - 3 / (6 ^B * (4 - 3)) ^C + 4

+
(
/

A

(
/
-

B

/
-

C

Note: A indicates the location in the expression **AFTER** the plus operator and **BEFORE** the value 6. B indicates the location in the expression **AFTER** the value 6 and **BEFORE** the multiplication operator. C indicates the location in the expression **AFTER** the closing parenthesis and **BEFORE** the plus operator.

Resulting postfix expression:

1	7	4	-	*	5	6	+	/	9	2	*	+	3	6	4	3	-	*	/	-	4	+
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Note: There are exactly the correct number of boxes above. These should be filled with 12 numbers and 11 operators.

Grading: 2 pts for each stack (2 pts if correct, 1 pt if one error, 0 pts otherwise), 4 pts for the expression, take off 1 point for each error, capped at 4.

Computer Science Foundation Exam

August 29, 2026

Section B

ADVANCED DATA STRUCTURES

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

SOLUTIONS

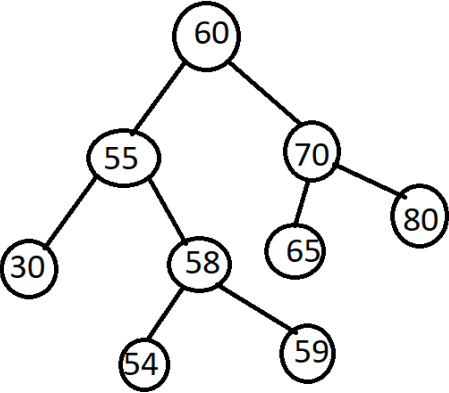
Question #	Max Pts	Category	Score
1	5	ALG	
2	10	ALG	
3	10	ALG	
TOTAL	25	----	

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) (5 pts) ALG (Binary Trees)

a) (2 pts) **Circle** the correct option. Is this a valid binary search tree?

	i) Yes, this is a binary search tree ii) No, this is not a binary search tree. <u>Justify your answer:</u> 54 is in the right subtree of 55 Grading: 1 pt for answer, 1 pt for justification, Justification pt can only be earned for a "No" answer.
---	---

b) (3 pts) Write the pre-order traversal of the above tree:

60, 55, 30, 58, 54, 59, 70, 65, 80

Grading: 1 pt 60 first

1 pt for having the left subtree correct (55, 30, 58, 54, 59)

1 pt for having the right subtree correct (70, 65, 80)

2) (10 pts) ALG (Heaps)

You are given an arbitrary array A containing n elements. The objective is to convert the array into a **binary min heap** using the ***heapify()*** operation. The array uses **1-based indexing**, so $A[0]$ is not used, and the n elements are stored in $A[1]$ through $A[n]$.

For example:

```
int A[] = {-, 42, 17, 8, 23, 4, 15, 30, 2, 19, 11}; //A[0] is not used
int n = 10; //number of elements
```

index	1	2	3	4	5	6	7	8	9	10
Element	42	17	8	23	4	15	30	2	19	11

The following heap-related functions have already been implemented and are available for you to use:

```
void percolateUp(int A[], int index, int size);
void percolateDown(int A[], int index, int size);
void insert(int A[], int *size, int value);
int deleteMin(int A[], int *size);
```

Answer the following two questions:

a) (5 pts) Perform heapify on the above array and show the resulting **min heap as an array.** *[You don't have to show the steps. You will be graded only based on the contents of the array.]*

Please place your final answer in the given slots below:

index	1	2	3	4	5	6	7	8	9	10
Element	2	4	8	17	11	15	30	23	19	42

Grading: Score = floor(numCorrect/2)

b) (5 pts) Heapify Implementation: Write the following ***heapify()*** function that rearranges the n elements in $A[1]$ through $A[n]$ so that they form a valid binary min heap. Parameter A is the array containing the elements, and n is the number of elements currently stored in the array, not the allocated size or capacity of the array. You may use any of the provided heap functions shown above, but you may not modify them or use an additional array. Your implementation must run in **$O(n)$** time and should not make unnecessary calls to the provided heap functions.

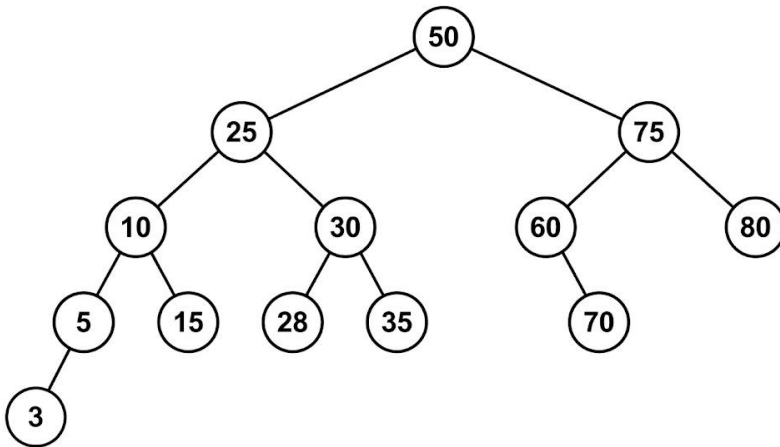
```
void heapify(int A[], int n) {
    for (int i = n / 2; i >= 1; i--)
        percolateDown(A, i, n);
}
```

Grading: Loop – 3 pts (1 pt backwards, 1 pt starting $n/2$, 1 pt ending 1, inclusive)

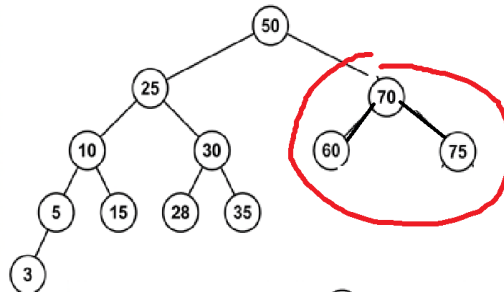
Function Call – 2 pts (1 pt name, 1 pt all parameters, all have to be correct to get the point.)

3) (10 pts) ALG (AVL Trees)

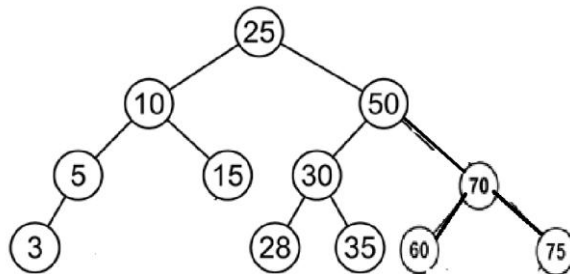
- a) (8 pts) Consider the following AVL tree. Show the updated tree after deleting the node containing 80. There are 2 rebalances that occur in the process of deleting the node and fixing the tree. 3 pts will be for the picture of the tree after the first rebalance and the other 5 pts will be for the final picture of the tree. Please clearly mark each picture.



After deleting 80 and first restructuring: (Grading: 1 pts for leaving untouched 50 and left, 2 pts for right subtree of 50)



After second restructuring, the final tree: (Grading: 1 pt root, 2 pts left subtree, 2 pts right subtree)



- b) (2 pts) What additional information must be stored with each node in an AVL tree that isn't necessarily stored for a regular binary search tree node?

Node's height (distance from the node to furthest leaf node in its subtree) (2 pts all or nothing, no credit for the answer "balance factor".)

Computer Science Foundation Exam

August 29, 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	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 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)

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 the front of a linked list storing n items. $O(1)$
- (b) Inserting 1 item into the back of a linked list storing n items, where there is only direct access to the front node of the list. $O(n)$
- (c) Returning the minimum value in a minheap storing n items. $O(1)$
- (d) **Deleting** the minimum value in a minheap storing n items. $O(\lg n)$
- (e) Sorting an array of n items using Quick Sort. $O(n^2)$
- (f) Sorting an array of n items using Merge Sort. $O(n \lg n)$
- (g) Starting with an empty array, appending n items to the array, one by one, reallocating space before each addition via the realloc function, so that at all times right after an element is added, the array is filled to capacity. $O(n^2)$
- (h) Printing out each subset of a set with n items. Assume that the time it takes to print out a subset of k items is $O(k)$. $O(2^n)$
- (i) Running a floodfill on a grid with r rows and c columns. $O(rc)$
- (j) Searching for a word with d letters in a trie storing n words. $O(d)$

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

2) (10 pts) ANL (Algorithm Analysis)

An algorithm that processes an array of size n runs in $O(n \lg^2 n)$ time. (Note: $\lg^2 n = (\lg n) \times (\lg n)$.) For an array of size $n = 10^6$, the algorithm runs in 108 milliseconds. How many **seconds** will the algorithm be expected to take when processing an array of size 10^7 ?

Let $T(n)$ represent the run-time of the algorithm for an input size of n .

Then, $T(n) = cn \lg^2 n$, for some constant c .

Using the given information, we have $T(10^6) = c(10^6)(\lg 10^6)^2 = 108\text{ms}$. Solve for c :

$$c = \frac{108\text{ms}}{(10^6)(6\lg 10)^2} = \frac{108\text{ms}}{(10^6)6^2(\lg 10)^2} = \frac{3\text{ms}}{(10^6)(\lg 10)^2}$$

We need to solve for $T(10^7)$:

$$\begin{aligned} T(10^7) &= c(10^7)(\lg^2 10^7) \\ &= \frac{3\text{ms}}{(10^6)(\lg 10)^2} (10^7)(7\lg 10)^2 \\ &= \frac{3\text{ms}(10^7)(7^2)}{(10^6)(\lg 10)^2} (\lg 10)^2 \\ &= \frac{3\text{ms}(10^7)(7^2)}{(10^6)} \\ &= 147\text{ms} \times 10 \\ &= 1470\text{ms} \\ &= \mathbf{1.47 \text{ seconds}} \end{aligned}$$

Grading: 1 pt setting up equation with unknown constant

2 pts – solving for c without simplification

1 pt – plugging in 10^7 into equation

2 pts – properly reducing $(\lg^2 10^7)$ can canceling $(\lg 10)^2$.

1 pt – dividing 108 by 36 to get 3 in simplification (either earlier or later)

2 pts – getting to 1470 ms

1 pt – converting answer to seconds

3) (5 pts) ANL (Recurrence Relations)

Consider the following recurrence relation:

$$T(n) = 2T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + n$$

Using the iteration technique, for large n , we can rewrite this recurrence relation in the form:

$$T(n) = aT\left(\frac{9n}{16}\right) + bT\left(\frac{3n}{16}\right) + cT\left(\frac{n}{16}\right) + dn$$

where a , b , c and d are each positive constants. (Note: To get to this form, just plug into each term on the right-hand side once.)

Determine the values of a , b , c and d that fit the form above when plugging into each (recursive) term on the right-hand side once.

$$T(n) = 2T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + n$$

$$T(n) = 2(2T\left(\frac{9n}{16}\right) + T\left(\frac{3n}{16}\right) + \frac{3n}{4}) + T\left(\frac{n}{4}\right) + n$$

$$T(n) = 4T\left(\frac{9n}{16}\right) + 2T\left(\frac{3n}{16}\right) + \frac{3n}{2} + T\left(\frac{n}{4}\right) + n$$

$$T(n) = 4T\left(\frac{9n}{16}\right) + 2T\left(\frac{3n}{16}\right) + \frac{3n}{2} + (2T\left(\frac{3n}{16}\right) + T\left(\frac{n}{16}\right) + \frac{n}{4}) + n$$

$$T(n) = 4T\left(\frac{9n}{16}\right) + 4T\left(\frac{3n}{16}\right) + T\left(\frac{n}{16}\right) + \frac{11n}{4}$$

$$a = \underline{4} \qquad b = \underline{4} \qquad c = \underline{1} \qquad d = \underline{11/4}$$

Grading: 1 pt for a , b and c

2 pts for d

All or nothing for each individual slot.

Computer Science Foundation Exam

August 29, 2026

Section D

ALGORITHMS

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

SOLUTIONS

Question #	Max Pts	Category	Score
1	5	DSN	
2	10	DSN	
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) (5 pts) DSN (Recursive Coding)

A derangement of a list is a reordering of the list so that no item is in the same position in the reordering as the original ordering. For example, a derangement of the list Abbie, Binh, Chopra, Darnell is Binh, Darnell, Abbie, Chopra. D_n , the number of derangements of a list of n items can be defined recursively as follows:

$$D_0 = 1, D_1 = 0$$
$$D_n = (n - 1)(D_{n-1} + D_{n-2}), \text{ for } n \geq 2$$

Write a recursive function that takes in a non-negative integer, n , and returns D_n below. The function prototype has been given. You may assume that $0 \leq n \leq 12$, so that there will be no overflow issues.

```
int derangements(int n) {  
  
    if (n == 0) return 1;                // 1 pt  
    if (n == 1) return 0;                // 1 pt  
  
    return (n-1)*(derangement(n-1)+derangement(n-2));  
    // 3 pts - 1 pt for return, 1 pt for (n-1)*, 1 pt for  
    // both rec calls  
  
}
```

2) (10 pts) ALG (Sorting)

You are given the following subroutine to help you sort an array. It's an implementation of a helper function for a well-known sort, but written differently than typically taught. Use this subroutine to write a function that, in the average case, sorts an array of n integers in $O(n \lg n)$ time.

```
int whatdoesitdo(int* array, int low, int high) {

    int* a1 = calloc(high-low+1, sizeof(int));
    int* a2 = calloc(high-low+1, sizeof(int));
    int i = 0, j = 0, save = array[low];

    for (int idx=low+1; idx<=high; idx++) {
        if (array[idx] < array[low])
            a1[i++] = array[idx];
        else
            a2[j++] = array[idx];
    }

    for (int idx=low; idx<low+i; idx++)
        array[idx] = a1[idx-low];
    array[low+i] = save;
    for (int idx=low+i+1; idx<=high; idx++)
        array[idx] = a2[idx-low-i-1];
    free(a1); free(a2);
    return low+i;
}

// Sorts the contents of array from index low through index high,
// inclusive, in non-decreasing order.
void sort(int* array, int low, int high) {

    if (low >= high) return; // 2 pts
    int mid = whatdoesitdo(array, low, high); // 4 pts
    sort(array, low, mid-1); // 2 pts
    sort(array, mid+1, high); // 2 pts

}
```

Note to students: The function `whatdoesitdo` performs a partition, but not in place, it is stable though. Thus, the intended sort to perform is Quick Sort.

Grading notes: first 2pts – if can be opposite w/o return with code inside if
4 pts split – 1 pt to store answer in a variable, 1 pt rec call, 2 pts params
2 pts split – 1 pt for mid-1, 1 pt for rest
2 pts split – 1 pt for mid+1, 1 pt for rest

3) (10 pts) DSN (Bitwise Operators)

The answers to a True/False test of k questions can be stored in a single integer, provided that $k < 32$. Let the questions be numbered 0, 1, 2, ..., $k-1$. If the answer to question i is True, we set bit i (the one with value 2^i) to 1, otherwise, we set bit i to 0. Complete the `scoreTests` function below, which takes in an array, `studAnswers`, where `studAnswers[i]` stores the answers that the i^{th} student (0-based) gave for all of the test questions, an integer, `n`, representing the number of students who took the test, an integer `key`, which stores the correct answers to the test, and an integer `k`, representing the number of test questions. Your function should score each test (compute the number of correct answers for each student out of k questions) and store this value in the array `studScores` and return a pointer to this array. (Note: `studScores[i]` should store the number of correct answers given by student i on the test.) Complete the function below to perform this task. (Note: In order to receive full credit, you must use bitwise operators in a manner that shortens your code.)

```
int* scoreTests(int* studAnswers, int n, int key, int k) {

    int* studScores = calloc(n, sizeof(int));

    for (int i=0; i<n; i++) {                                // 1 pt
        studScores[i] = k;
        int res = studAnswers[i]^key;                        // 3 pts
        for (int j=0; j<k; j++)                               // 1 pt
            if (res & (1<<j) )                                // 2 pts
                studScores[i]--;
    }

    // 3 pts - mechanism for counting score. Score is total
    // minus number of 1 bits. Multiple ways to do this. I
    // started at k and subtracted for each wrong answer.
    // Just as easy to start from 0 and add for 0 bits or
    // add for 1 bits but then subtract from k when storing.

    return studScores;
}
```