Computer Science Foundation Exam

August 24, 2024

Section A

BASIC DATA STRUCTURES

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

PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME

Last Name: ______________________________

First Name: ______________________________

UCFID: ___________________________________

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.

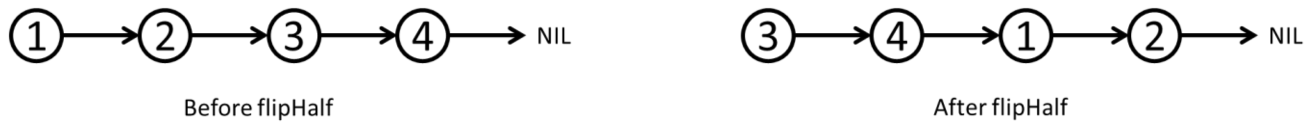
1) (10 pts) DSN (Dynamic Memory Management in C)

The struct `videogame_t` maintains information about video games that are stored in a store's inventory. As a fan of the 1980s, you would like to get a list of all the games that were produced in between the years 1980 and 1989, inclusive. Write the function below that takes in an array of type `videogame_t`, its length (`n`), and a pointer to a variable that will store the number of games produced in the 1980s. The function should return a newly dynamically allocated array storing a copy of all the information for the games that were produced in the 1980s AND set the variable pointed to by ptrNumGames to the size of the array returned by the function. This copy must be a deep copy, where individual component is copied over, including allocating memory for the copy of the name and copying the name into that new memory. **(Note: Due to the length of code, some of the function has been provided. Don't forget to allocate the appropriate space for each string in each struct!)**

```
//struct representing video game information
typedef struct {
    char * name;
    int year;
    double price;
} videogame_t;

videogame_t* getClassicGames(videogame_t * inventory,
                             int n, int* ptrNumGames) {

    videogame_t* res = malloc(n*sizeof(videogame_t));
    int nG = 0;

    res = realloc(res, nG*sizeof(videogame_t));
    *ptrNumGames = nG;
    return res;
}
```

2) (10 pts) DSN (Linked Lists)

Complete the following user defined function that reconstructs the structure of a singly linked list. The function will take a pointer to the head of some list along with the number of nodes `n` and move the first half the nodes to the back of the list while the other half moves up to the front. **<u>For grading purposes, please write your solution iteratively, with NO recursion.</u>** The function will return the new head of the list. **<u>You may assume that the original list has an even number of elements and is not empty.</u>** The following figure shows the scenario.



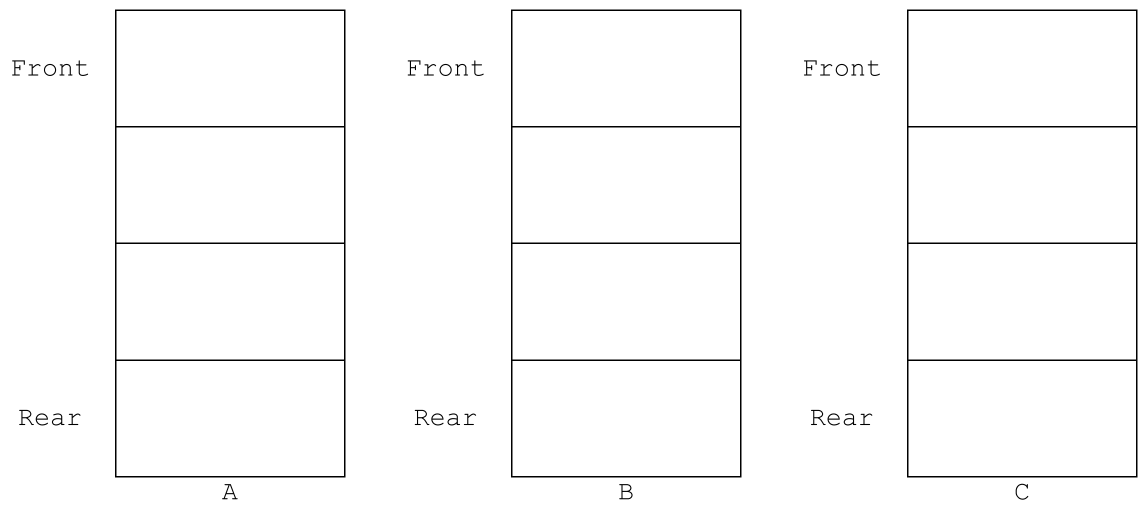
```
typedef struct node_s {
    int data;
    struct node_s* next;
} node_t;

node_t * flipHalf(node_t * head, int n) {
```

```
}
```

3) (5 pts) ALG (Queues)

Consider the following C code that represents a FIFO queue that holds a list of superheroes as strings. Show the contents of the queue after each indicated point commented (A, B, and C).

```
 typedef struct node_s {
    char * hero;
    struct node_s * next;
} node_t;

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

void enqueue(queue_t * heroqueue, char * hero);
char * dequeue(queue_t * heroqueue);

void followQueue(queue_t * heroqueue){
      enqueue(heroqueue, “Hawkeye”);
      enqueue(heroqueue, “Thor”);
      enqueue(heroqueue, “Spider-Man”);
      dequeue(heroqueue);
      enqueue(heroqueue, “Wanda”);
      enqueue(heroqueue, “Vision”); //A
      enqueue(heroqueue, “Ms. Marvel”);
      enqueue(heroqueue, “Dr. Strange”);
      dequeue(heroqueue);
      dequeue(heroqueue);
      enqueue(heroqueue, “Loki”);
      enqueue(heroqueue, “Captain Marvel”);
      dequeue(heroqueue);
      dequeue(heroqueue); //B
      enqueue(heroqueue, “Iron Man”);
      dequeue(heroqueue); //C
}
```



Note: There are exactly the correct number of boxes to be filled for each state. But, the intermediate steps may have the queue store more than 4 elements.

Computer Science Foundation Exam

August 24, 2024

Section B

ADVANCED DATA STRUCTURES

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

PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME

Last Name: ________________________________

First Name: ________________________________

UCFID: ________________________________

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.

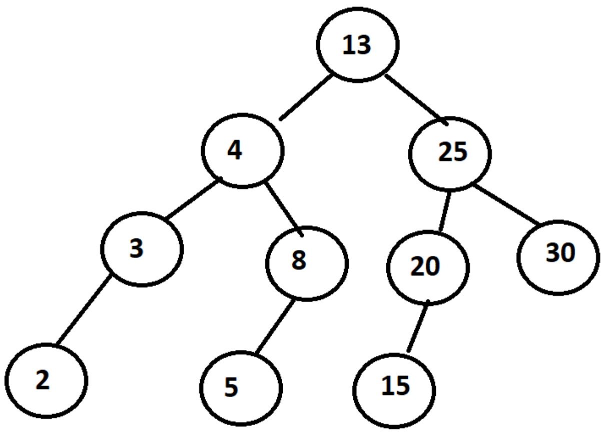
Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) **1**(10 pts) DSN (Binary Trees)

Consider the following mystery function that uses a typical tree node structure of a Binary Tree.

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

int mystery (struct treenode* root) {

    if(root == NULL)
        return 0;

    if(root->data %2 == 0) {
        struct treenode* temp = root->left;
        root->left = root->right;
        root->right = temp;
    }

    return 5 + mystery(root->left) + mystery(root->right);
}
```

a) Redraw (on the right side) the state of the tree below after mystery is called on its root, AND **b)** indicate the value returned by the function.



b) Return Value: __________________________

2) (5 pts) ALG (Hash Tables)

a) (3 pts) Consider a hash table that uses the linear probing technique with the following hash function f(x) = (5x+4)%10. (The hash table is of size 10.) If we insert the values 3, 9, 2, 1, 10, and 6 into the table, in that order, show where these values would end up in the table.

Index	0	1	2	3	4	5	6	7	8	9
Value										

b) (2 pts) Why is the hash function defined in part (a) a particularly bad hash function?

3) (10 pts) ALG (AVL Trees)

There is a unique AVL tree of height 4 with 12 nodes storing the integers 1 through 12, inclusive, such that for every non-leaf node, the left subtree is strictly shorter than the right subtree.

(a) (7 pts) Draw this AVL Tree.

(b) (3 pts) Explain how you constructed the tree based on the prompt.

Computer Science Foundation Exam

August 24, 2024

Section C

ALGORITHM ANALYSIS

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

PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME

Last Name: ______________________________

First Name: ______________________________

UCFID: ___________________________________

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)

Consider the task of sorting $\mathbf{n^2}$ integers. Using an insertion sort, this task would take O($\mathbf{n^4}$) time. Using a single heap sort, this task would take O($\mathbf{n^2 lg\ n}$). Consider this hybrid approach and, **with proof, determine its worst case run time, in terms of n.** Assume efficient implementations of each of the heap and linked list operations described. Leave your answer in Big-Oh notation.

1. Separate the $\mathbf{n^2}$ integers into **n** groups of **n** integers each.

2. Create heaps out of each of the **n** groups of integers.

3. Call delete min on each of the **n** heaps, storing these **n** deleted values in a linked list, also storing which heap each value came from.

4. Repeat the following $\mathbf{n^2}$ times:

    a. Loop through the linked list, locating the minimum integer in it, noting which heap it was from. Name the integer **x** and the heap **H**.

    b. Place **x** next in the sorted list and delete it from the linked list.

    c. If **H** isn't empty, delete the minimum item from **H** and add it to the end of the linked list, also storing that the value came from heap **H**.

2) (10 pts) ANL (Algorithm Analysis)

A brute force algorithm which processes all permutations of n routines runs in $O(n \times (n!))$ time. On a particular computer, executing the algorithm for **n = 9** takes 180 milliseconds. How many seconds is the algorithm expected to take on an input of size **n = 11**, run on the same computer?

3) (5 pts) ANL (Summations)

Determine a closed form solution to the following summation in terms of **n**. Simplify your answer to standard polynomial form (not factored).

$$\sum_{i=1}^{3n}(2i+5)$$

Computer Science Foundation Exam

August 24, 2024

Section D

ALGORITHMS

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

PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME

Last Name: ________________________________

First Name: ________________________________

UCFID: ________________________________

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 (Recursion)

Write a recursive function that given an array, the length of the array, and a range of indices finds the number of even integers in an array in the specified index range. **Solutions that use a loop will receive 0 points**. (For example, if the arr stored [3, 5, 8, 6, 7, 9, 4, 11, 8], the function call evens(arr, 9, 2, 6) should return 3 because the contents of index 2, 3 and 6 are all even numbers. In particular, arr[2] = 8, arr[3] = 6 and arr[6] = 4.)

```
int evens(int * arr, int len, int lowInd, int hiInd) {













}
```

2) (10 pts) DSN (Sorting)

For this problem, fill in the blank to finish the stable (elements with the same values are kept in their original order) quicksort on a linked list. We are using the head of the linked list as a pivot. You can assume that the following linked list functions have all been implemented and take O(1) operations.

Note: Each blank is worth one point and involves either making calls or filling in parameters to the functions whose prototypes and descriptions are given below.

```
typedef struct Node {
    int value;
    struct Node * next;
} Node;

typedef struct List {
    Node* front;
    Node* back;
} List;

void addToTail(List * list, Node * node); // Add to tail

// Returns a list that is the combination of 2 given lists.
List * merge(List * front, List * back);

Node* getAndRemoveHead(List * list); // Removes and returns the head
List* createEmptyList(); // Returns dynamically allocated empty List
int isEmpty(List * list); // Returns 1 if empty and 0 otherwise

// Sort code on next page
```

```
List * sort(List * lst) {

   if (_______________________(lst)) return lst;

   Node * pivot = getAndRemoveHead(lst);

   List * first = _______________________();

   List * last = _______________________();

   List * middle = createEmptyList();

   addToTail(middle, pivot);

   while (!isEmpty(lst)) {

      Node * cur = _______________________(lst);

      if (cur->value < pivot->value)

         addToTail(_______________________, cur);
      else if (cur->value == pivot->value)

         addToTail(_______________________, cur);
      else

         addToTail(_______________________, cur);
   }

   first = sort(_______________________);

   last = sort(_______________________);

   first = merge(first, middle);
   first = merge(first, last);
   free(middle);
   free(last);
   free(lst);

   return _______________________;

}
```

3) (10 pts) DSN (Bitwise Operators)

A lights out puzzle is a classic puzzle played on a 2D grid that involves turning the cells of the grid off. The cells can be toggled by pressing a particular cell. Doing so will invert the cell pressed and the 4 adjacent cells that are the immediate neighbor (joined by a cell line). Your program stores a "lights out" puzzle represented using an array of integers, where each integer in the array represents a row of the puzzle. The bits when on represent that the cell is on, and when off represent that the cell is off. If in some row the first, sixth, and seventh cell is on, then the value at the spot in the array would be $97 = 2^0 + 2^5 + 2^6$. If we were to press the button in column 1 of this row, then it would turn spot 0 off, spot 1 on, and spot 2 on, changing the bit mask on that row to $2^1 + 2^2 + 2^5 + 2^6 = 102$. In addition, if the row above existed one bit would be changed there and if the row below existed, one bit would be changed there.

You will create a function that will emulate pressing a particular cell. You will be given the row and column (0-based index) of the cell toggled along with the grid and its dimensions. Modify the grid based on the described interaction. Be careful of indexing out of bounds. (Hint: There are 5 bits total that will potentially flip. One of those bits will always flip while the other four bits, the neighboring bits, will only flip if they are within the bounds of the grid.)

```
void cellPress(int row, int col, int height, int width, int * grid) {









}
```