# Computer Science Foundation Exam

**January 14, 2023**

**Section A**

**BASIC DATA STRUCTURES**

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

**Name:** ______________________________________________

**UCFID:** ______________________________________________

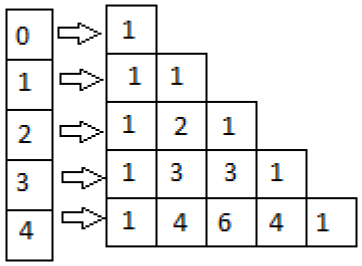
| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 10 | DSN | |
| 2 | 10 | DSN | |
| 3 | 5 | ALG | |
| TOTAL | 25 | ---- | |

**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 <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)

Using 0-based indexing, on row i of Pascal's Triangle, there are i+1 positive integer values. One way we can efficiently store the triangle is to allocate the correct amount of memory for each row. Here is a picture of the first five rows of the triangle (rows 0 through 4, inclusive.):



If the name of the array is **tri**, then the rule to fill in the entries in the table are as follows:

```
tri[i][0] = 1, for all positive ints i
tri[i][i] = 1, for all positive ints i
tri[i][j] = tri[i-1][j-1]+tri[i-1][j], for all ints j, 0 < j < i
```

Write a function that takes in an integer, ***n***, dynamically allocates an array of ***n*** arrays, where the ith array (0-based) is allocated to store exactly i+1 ints, fills the contents of the array with the corresponding values of Pascal's Triangle as designated above, and returns a pointer to the array of arrays. **You may assume that 1 < n < 31.**

```
int** getPascalsTriangle(int n) {









}
```

**2)** (10 pts) DSN (Linked Lists)

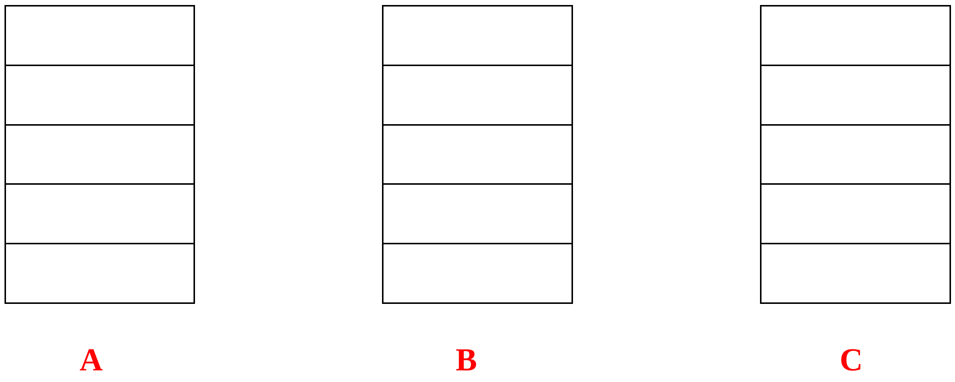
Consider using a linked list to store a string, where each node, in order, stores one letter in the string. The struct used for a single node is included below. Write a function that takes in two pointers to linked lists storing 2 strings, and prints out the letters in the string in alternating order, starting with the first letter of the first string. If one string runs out of letters, just skip over it. For example, if the two strings passed to the function were "hello" and "computer", then the function should print "hceolmlpouter".

```
typedef struct node {
    char letter;
    struct node* next;
} node;

void printMixed(node* word1, node* word2) {









}
```

**3)** (5 pts) ALG (Stacks)

Evaluate the following postfix expression, using the algorithm that utilizes an operand stack. Put the value of the expression in the slot provided and show the state of the operand stack (in this case the stacks should just have numbers in them) at each of the indicated points in the expression:

```
                                   A               B               C
4   3   +   5   2   -   *   8   7   9   +   +   6   /   9   4   -   *   -
```



Note: A indicates the location in the expression **AFTER** 7 but before 9.
B indicates the location in the expression **AFTER** 6 but before the division.
C indicates the location in the expression **AFTER** the subtraction but before the multiplication.

Value of the Postfix Expression: ________

# Computer Science Foundation Exam

**January 14, 2023**

**Section B**

**ADVANCED DATA STRUCTURES**

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

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

Write a **recursive function** called *prune()* that takes the root of a binary tree and deletes all leaf nodes from the tree. Be sure to update any pointers that need to be updated to account for deleted nodes, carefully avoid any potential segmentation faults, and avoid memory leaks. (You may assume all the nodes in the tree have been dynamically allocated.)

For example:

```
              Initial Tree                           Resulting Tree
     (leaf nodes underlined and bold)                 (after pruning)

                  18                                       18
                 /  \                                        \
               25    1                                        1
                    / \                                        \
                   2   9                                        9
                        \
                         16
```

This function should return NULL if the root it receives gets deleted. Otherwise, it should return *root* itself. Note that if *root* is NULL, the function should simply return NULL without taking any further action.

You cannot write any helper functions for this problem. All your work must be contained in a single function called *prune()*. The node struct definition and function signature are as follows:

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

node *prune(node *root)
{








}
```

**2)** (5 pts) ALG (Hash Tables)

Consider the following problem: Suppose we have a rather large text file that contains up to 50 million strings of 9-digit integers (each separated by a space), and we know that **exactly one** of those integers occurs twice. (The rest, we know for certain, occur only once.) Suppose our goal is to find the single 9-digit integer that occurs twice.

Some clever cheddar of a computer scientist thinks they have come up with a great solution to this problem. They have a rock-solid implementation of hash tables (written in C) that has been thoroughly tested by the open-source community and is known to be reliable and bug-free. This implementation of hash tables has some great features:

- The hash tables have a lovely, effective, widely-used hash function for dealing with integers.
- The hash tables expand automatically without creating any memory leaks.
- In addition to keeping track of the elements in the hash table, this implementation keeps track of a few extra pieces of information inside the HashTable struct. One of those extra pieces of information is how many collisions have occurred over the lifetime of this hash table. (That value is, of course, initialized to zero (0) by the function used to create new hash tables.)

It is this last property that has caught our clever cheddar's eye. They reason that they can loop through the input file, throw each integer into a hash table one-by-one, and as soon as a collision occurs in the hash table, they must have found the duplicate integer they were searching for! So, they write up a piece of code that effectively does this:

```
function find_that_duplicate(file):
    create a hash table, h
    for each integer, n, in file:
        insert n into h
        if h->num_collisions > 0:
            destroy h
            close file
            return n
```

The idea above is just pseudocode, of course. You may assume that the person implementing this in C is careful to avoid memory leaks and segmentation faults, that there is enough memory to hold all those integers in the hash table, that the hash table implementation works well and is rock solid, and that the code has no syntax errors. You may also assume this function will **only** be called with a file that actually exists, and that we know for sure there is exactly one duplicate integer in that file. (So, we do not need to worry about what happens if there is no duplicate.)

**What is wrong with this proposed solution?** In your response, be as clear and precise as possible. Long, rambling word salads will not receive credit.

**3)** (10 pts) ALG (Tries and AVL Trees)

(a) (5 pts) (Tries) Suppose we insert the following strings into an initially empty trie:

```
cat   cap   catapult   catalyst   phosphorescent   pancake   pancakes
```

How many nodes will the resulting trie contain? Please count carefully, as credit for this question may be all or nothing

(b) (5 pts) Show what the following AVL would look like after inserting the value 42 and performing any necessary rotations.

```
   18
  /  \
 7    38
  \     \
  12     59
```

# Computer Science Foundation Exam

**January 14, 2023**

**Section C**

**ALGORITHM ANALYSIS**

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

**Name: ___________________________________________**

**UCFID: ___________________________________________**

| 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 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.h, stdio.h, math.h, string.h) for that particular question have been made.**

**1)** (5 pts) ANL (Algorithm Analysis)

What is the big O runtime for the following segment of code in terms of N and M? (Note: let min(x, y) denote the minimum of x and y and max(x, y) denote the maximum of x and y. You may use either of these in your answer. **In addition to your answer, please provide justification for your answer.**

```
int fun(int N, int M, int ** grid) {
   int a1 = 0, a2 = 0;
   for (int i = 0; i < N || i < M; i++) {
      if (i < N) a1 += grid[i][0];
      if (i < M) a2 += grid[0][i];
   }
   if (a1 < a2) return a2;
   return a1;
}
```

**2)** (10 pts) ANL (Algorithm Analysis)

You are using an algorithm that can multiply 2 N-digit integers in O($N^{1.5}$) time. It takes $(10/13)^3$ seconds to multiply 2 numbers that have 100,000 digits. What is the expected number of digits of 2 numbers we could multiply together that would take exactly 1 second? **Please show all your work, including algebraic simplification, which is part of what is being tested with this question.**

**3)** (10 pts) ANL (Recurrence Relations)

What is the closed form for the following recurrence relation, T(N)? For full credit your work must be shown. Note: Your answer should be **EXACT** and not a Big-Oh bound.

$$T(N) = 2T(N-1) + N \text{ (for integers N} \geq 1\text{)}$$
$$T(0) = 0$$

# Computer Science Foundation Exam

## January 14, 2023

## Section D

## ALGORITHMS

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

**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 (Recursive Coding)

Define an extreme permutation of the integers 0 to n – 1 as any permutation where each value in the permutation (from left to right) is either the smallest or largest value not yet placed. For example, for n = 6, [0, 1, 5, 2, 3, 4] is an extreme permutation but [0, 5, 2, 4, 1, 3] is not. The latter is not because the only valid values that can be placed where the 2 is are either 1 or 4, the smallest and largest values, respectively, that have not been placed. Complete the recursive function below so that it prints out all extreme permutations of length n. A completed wrapper function has been provided. Note: low represents the lowest unplaced value, high represents the highest unplaced value, and k represents the number of items in the permutation that have already been filled.

```
#include <stdio.h>
#include <stdlib.h>

void printExtremeWrapper(int n);
void printExtreme(int* perm, int n, int low, int high, int k);
void printPerm(int* perm, int n);

void printExtremeWrapper(int n) {
    int* perm = malloc(sizeof(int)*n);
    printExtreme(perm, n, 0, n-1, 0);
    free(perm);
}

void printPerm(int* perm, int n) {
    for (int i=0; i<n; i++) printf("%d, ", perm[i]);
    printf("\n");
}
void printExtreme(int* perm, int n, int low, int high, int k) {

    if (low > high) {
        printPerm(perm, n);
        return;
    }









}
```

**2)** (10 pts) DSN (Sorting)

Complete the following merge function that is used as part of the merge sort process. The function performs the merge operation from left to mid and mid+1 to right index of the array.

```
void merge(int arr[], int left, int mid, int right)
{
    int i, j, k;
    int n1 = mid - left + 1; //size of the left array
    int n2 =  right - mid; //size of the right array

    /* create temp arrays */
    int *L = (int*) malloc(n1*sizeof(int)); //left array
    int *R = (int*) malloc(n2*sizeof(int)); //right array

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of left subarray
    j = 0; // Initial index of right subarray
    k = left; // Initial index of merged subarray
    // Complete the remaining part of the code that will
    // merge L and R array into arr









}
```

**3)** (5 pts) ALG (Base Conversion)

Convert 375 in base 10 to binary. **Please show your work and put a box around your final answer.**