Spring 2026 COP 3502H Final Exam Part A 4/30/2026

Last Name: _____ , First Name: _____

1) (10 pts) Complete the function below which uses the binary search method to determine the n^{th} root of x. Both x and n are passed in as parameters. x is guaranteed to be a positive real number less than one billion and n is guaranteed to be a positive integer greater than 1. Note: be careful setting low and high. Also, it's intended for you to use a call to a function in the math library in your solution.

```
#include <math.h>

double nthRoot(double x, int n) {




}
```

2) (15 pts) Complete the code on the next page so that it prints out the first jumping knight tour that visits n unique locations. In your solution, you must fill in the given DR, DC arrays to follow how a knight in chess jumps. Most of what this question is testing is your ability to read the portions of the code given to understand how the rest of the solution must work. The variables are named normally and are NOT obfuscating their role, so use that to your advantage. The grid is input as several strings, each of length COLS. Each character is either 'S', '_' or 'X'. There is exactly one 'S' character. This indicates the starting location. '_' are valid locations for the knight to jump to. 'X' are invalid locations. For the input grid

```
S___X
X__X_
_____
_X___
```

the program prints out

```
(0,0)->(1,2)->(2,0)->(0,1)->(2,2)->(0,3)->(1,1)
```

For this code, the # of rows is 4 and # cols is 5 (and is hard-coded)

All support code is on the back of this sheet.

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

#define ROWS 4
#define COLS 5
#define LEN 6
#define NUMDIR 8

const int DR[] = {-2,-2,-1,-1,1,1,2,2};
const int DC[] = {-1,1,-2,2,-2,2,-1,1};

int go(char board[][COLS+1], int used[][COLS], int* path, int curLoc, int k, int n);
int inbounds(int r, int c);
void init(int used[][COLS], char board[][COLS+1]);
int findS(char board[][COLS+1]);

int main() {

    char board[ROWS][COLS+1];
    for (int i=0; i<ROWS; i++)
        scanf("%s", board[i]);

    int used[ROWS][COLS];
    init(used, board);
    int loc = findS(board);

    int path[ROWS*COLS];
    path[0] = loc;
    go(board, used, path, loc, 1, LEN+1);

    for (int i=0; i<LEN; i++)
        printf("(%d,%d)->", path[i]/COLS, path[i]%COLS);
    printf("(%d,%d)\n", path[LEN]/COLS, path[LEN]%COLS);
    return 0;
}

void init(int used[][COLS], char board[][COLS+1]) {
    for (int i=0; i<ROWS; i++)
        for (int j=0; j<COLS; j++)
            used[i][j] = (board[i][j] == 'X' || board[i][j] == 'S');
}

int findS(char board[][COLS+1]) {
    for (int i=0; i<ROWS; i++)
        for (int j=0; j<COLS; j++)
            if (board[i][j] == 'S')
                return i*COLS + j;
    return -1;
}

int inbounds(int r, int c) {
    return r>=0 && r<ROWS && c>=0 && c<COLS;
}
```

```
int go(char board[][COLS+1], int used[][COLS], int* path, int curLoc, int k, int n)
{
    if (k == n) return 1;

    int curR = _____________________________ ;

    int curC = _____________________________ ;

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

        int nextR = ___________________________________ ;

        int nextC = ___________________________________;

        if ( _____________________________________________) continue;

        if (_____________________________________________) continue;

        int nextLoc = _____________________________________ ;

        ____________________________________________ ;

        ____________________________________________;

        int tmp = go(board, used, path, nextLoc, k+1, n);

        ____________________________________________;

        ____________________________________________;
    }

    ___________________________________ ;
}
```

Spring 2026 COP 3502H Final Exam Part B 4/30/2026

Last Name: _____ , First Name: _____

A1) (10 pts) DSN (Dynamic Memory Management in C)

Each farm in a group of farms has some number of crops. Each crop has a total yield, in pounds. For this question, write a function that takes in the information about all of the crops on all of the farms, and returns a single array (dynamically allocated) storing sum of pounds of all of the crops for each farm. For example, if there were 3 farms with the crop yields of {2, 5, 4}, {2, 1}, and {9, 3, 4, 5}, your function should return a single array storing {11, 3, 21}. The input to the function will be the following:

`int** allcrops` – 2D array with all crop data. allcrops[i] is an array which stores the separate yields for each crop on farm i.
`int* numcrops` – 1D array where numcrops[i] stores the number of different crops for farm i.
`int numFarms` – The number of farms for which allcrops stores data.

Fill in the function below so that it returns a pointer to the array storing the sum of crop yields for eacm farm. Do not modify the contents of either allcrops of numcrops.

```
int* totalYield(int** allcrops, int* numcrops, int numFarms) {

}
```

A2) (10 pts) DSN (Linked Lists)

Consider storing an unsigned binary number in a linked list, storing one bit in each node, where the first node stores the bit value of the least significant bit. Incrementing the value of an integer stored in this manner never requires changing where the pointer to the first node is pointing, thus we can design an increment function to take in a pointer to a linked list storing one of these integers and perform the increment without returning anything. For example, if number was pointing to the list → 0 → 1, then the increment function would change this list to look like → 1 → 1, incrementing the number from a value of 2 to 3. If we again called the increment function on the list storing 3, it would change the list to be → 0 → 0 → 1, The first node isn't changed; rather the value stored in it is flipped from 1 to 0. Same for the second node. Then, a third node is added to the list, storing 1. Complete the function below, **writing it iteratively**, to perform the increment on a binary number stored in this fashion. Use the struct given below. You may assume that number is pointing to a list storing a valid non-negative binary number, meaning that the last node will store 1 unless the one and only node in the list stores 0.

```
typedef struct node {
    int bit;
    struct node* next;
} node;

void increment(node* number) {




}
```

A3) (5 pts) ALG (Queue)

In class, the breadth first search algorithm using a queue was shown to go through visiting squares in a corn maze (a 2d grid with some marked out squares) from a starting point. Assuming that the order of enqueing squares from a current square is up, right, left and down, show the order that the following squares in the grid below get visited. The valid squares are numbered 1 to 20 and the marked out squares are shown with an x. Start the search at square number 6 (my favorite number!) (Hint: Remember that the squares get visited based on distance order, so all squares that are a distance of 2 from the starting point get visited after all squares that are a distance of 1 from the starting point.)

1	**2**	**3**	**4**
5	**x**	**6**	**7**
8	**9**	**10**	**11**

6 , ___ , ___ , ___ , ___ , ___ , ___ , ___ , ___ , ___ , ___

B1) (10 pts) DSN (Binary Trees)

In a binary tree of n nodes, consider the n – 1 paths from the root to each of the other nodes. We define the length of these paths to each equal the number of edges/links that need to be traversed on the shortest path from the root to each of the other nodes. Write a function that takes in a pointer to the root of a binary tree and returns the sum of the lengths of these paths. If the tree has 0 or 1 node, your function should return 0. You may assume that the return value will NOT overflow the int data type. Using the struct below, complete the given function to return this sum. **Notice that for each node, in addition to storing a data value, we store the number of nodes in the subtree rooted at that node. This value in the node is fairly critical for solving this problem efficiently. You may assume this value is always accurate.**

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

int sumPLen(treenode* root) {




}
```

B2) (5 pts) ALG (Binary Heaps)

(a) (3 pts) Draw the binary heap that is stored in the array shown below:

index	1	2	3	4	5	6	7	8	9	10
value	13	55	17	82	57	90	22	83	97	63

(b) (2 pts) Show the resulting binary heap in tree form when 47 is inserted into it.

B3) (10 pts) DSN (Tries)

Write a **recursive** function that takes in a pointer to the root of a trie and returns the total number of words stored in that trie. Use the struct and function prototype given below.

```
typedef struct trienode {
    int isWord;
    struct trienode* next[26];
} trienode;

int numWords(trienode* root) {




}
```

C1) (10 pts) ANL (Algorithm Analysis)

List the **worst case run times** of each of the following operations assuming the efficient implementations of these data structures and algorithms shown in class. Leave your run times in Big-Oh notation in terms of the appropriate variables.

a) Inserting an item into a queue of **n** items. _____

b) Removing the minimum item from a binary heap of **n** items. _____

c) Sorting **n** items using Quick Sort. _____

d) Searching for **k** different items sequentially in an AVL tree of **n** items. _____

e) Printing out each subset of **n** items. Assume printing a single item in a single subset takes O(1) time. _____

f) Inserting **m** items into an initially empty linked list that is sorted in order. _____

g) Running a floodfill on a grid with **r** rows and **c** columns. _____

h) Running a pre-order traversal on a binary tree of **n** nodes. _____

i) Returning but not removing the minimum item in a binary heap. _____

j) Adding a **b** bit binary number to a **c** bit binary number _____

C2) (5 pts) ANL (Algorithm Analysis)

An algorithm which has a run time of O(lg n) takes 300 microseconds when it's run on an input with size n = 10,000. How long, **in microseconds**, would we expect the algorithm to take when it's run on an input with size $n = 10^9$?

C3) (10 pts) ANL (Sums)

Determine a closed form of the following sum in terms of n. Please keep your answer in factored form. Each term in your factored form will either be of the form 2^a or $2^a + b$, where a and b are either constants or linear expressions in terms of n.

$$\sum_{i=1}^{2^n} (6i^2 + 2i)$$

D1) (5 pts) DSN (Recursive Coding)

In class we discussed the idea of a "lowest one bit." Traditionally, this means minimum value of each bit set to 1 in a binary number. For example, for 24, which, in binary is 11000, the lowest 1 bit is 8, since the right most bit set to one is in the third bit from the right, counting from 0, so it has the value 2^3. Consider a different, but related function which returns the place value (power of 2) of the lowest one bit. Write this function **recursively** below. **You may assume that the input to the function is a positive integer.**

```
int lowOneBitPos(int n) {




}
```

D2) (10 pts) ALG (Sorting)

(a) (5 pts) Show the result of running the partition algorithm shown in class on the array below. Use 13 as the partition element.

Before	13	6	2	9	7	15	4
After							

(b) (5 pts) Show the state of the array below after each iteration of selection sort (where we select for the maximum element).

Initial Array	27	6	19	13	42	9
1st Iteration						
2nd Iteration						
3rd Iteration						
4th Iteration						
5th Iteration						

D3) (10 pts) ALG (Bitwise Operators)

We define the bit count of an integer to be the number of bits set to 1 in the unsigned binary representation of that number. Write a function that takes in two integers, a and b, and uses a brute force method to determine the sum of the bit counts of each integer in between a and b. You may assume that both a and b are positive with $a < b < 2^{30}$ and that the result will not overflow the int data type.

```
int sumBitsOn(int a, int b) {




}
```