# Honors Computer Science I (COP 3502) Exam #1
**Date: 2/6/2019**

**Name: ___________________________________**

1) (a) (5 pts) What is $3201_4$ converted to base ten?

(b) (5 pts) What is $845_{10}$ converted base 7?

2) (4 pts) What is the return value of f(11), where f is the function defined below?

```
int f(int n) {
    if (n == 0) return 0;
    if (n%2 == 1) return -n + f(n-1);
    return n + f(n-1);
}
```

__________

3) (10 pts) Simplify the following summation: $\sum_{i=n+1}^{2n}(3i-2)$. Your final result should be a quadratic function in n.

4) (18 pts) A function is given below, answer, the three questions that follow. You may assume that the array contains all positive integer values and that no integer overflow occurs in the execution of the function.

```
int f(int* array, int n, int m) {
    int low = 0, high = n;
    while (low < high) {
        int mid = (low+high+1)/2;
        int sum = 0;
        for (int i=0; i<mid; i++)
            sum += array[i];

        if (sum <= m)
            low = mid;
        else
            high = mid-1;
    }
    return low;
}
```

(a) (7 pts) What is the best case run-time (Big-Oh Bound) of the function, in terms of the input parameter n? Provide proof of your answer.

(b) (7 pts) What is the worst case run-time (Big-Oh Bound) of the function, in terms of the input parameter n? Provide proof of your answer.

(c) (4 pts) Explain, in English, what value the function returns, using array, n and m, in your description.

5) (10 pts) Using the integral method, find a reasonably tight upper bound for the summation $\sum_{i=1}^{n} \sqrt{i}$, in terms of n. (A reasonably tight upper bound will be some function f(n) where $\sum_{i=1}^{n} \sqrt{i} \ \leq f(n)$, but such that the value of f(n), for large n, the ratio between the sum and f(n) comes pretty close to 1, but just under 1.)

6) (20 pts) In the game of Chess, a bishop can only travel on diagonals. Consider writing a recursive function that calculates every square a given bishop on a chess board can reach with zero or more consecutive moves. The recursive function will take in a two dimensional character array representing the chess board, a two dimensional integer array storing which squares can be visited by the bishop, and the size of the array (both rows and columns). The character array will store a single 'b' representing the original location of the bishop, the '.' character for each passable square and the '#' character for each blocked square. Complete the program so that the integer array, after the call to the recursive function, stores 1 in each cell corresponding to a reachable square by the bishop and 0 in each cell that is not reachable by the bishop. (In the program, this used array is printed after each case is processed.) Complete the blank portions of the code that follows so that it correctly solves this problem.

```
#include <stdio.h>
#include <stdlib.h>
#define NUMDIR 4

const int DX[] = {___ , ___ , ___ , ___};
const int DY[] = {___ , ___ , ___ , ___};

void go(char** grid, int** used, int n, int curx, int cury);
int findCh(char** grid, int n, char ch);

int main(void) {
    int numCases;
    scanf("%d", &numCases);
    for (int loop=0; loop<numCases; loop++) {
        int n;
        scanf("%d", &n);
        char** grid = malloc(sizeof(char*)*n);
        int** used = malloc(sizeof(int*)*n);
        for (int i=0; i<n; i++) {
            grid[i] = malloc(sizeof(char)*(n+1));
            used[i] = calloc(n, sizeof(int));
            scanf("%s", grid[i]);
        }

        int loc = findCh(grid, n, 'b');
        go(grid, used, n, loc/n, loc%n);

        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++)
                printf("%d", used[i][j]);
            printf("\n");
        }
        for (int i=0; i<n; i++) {
            free(grid[i]);
            free(used[i]);
        }
        free(grid);
        free(used);
    }

    return 0;
}
```

```
void go(char** grid, int** used, int n, int curx, int cury) {
```

```
}

int findCh(char** grid, int n, char ch) {
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            if (grid[i][j] == ch)
                return n*i + j;
    return -1;
}
```

7) (26 pts) In this problem you will write two functions for a struct that manages a dynamically sized integer array. The struct is given below. The two method signatures will be on the following page. The two functions you will write are makeList, which dynamically allocates a default IntList of size 10 and returns a pointer to it. The other function will be an insert function, which takes in a pointer to an IntList, an index to perform an insertion, and the value to insert. The function returns 0 if the insertion index is invalid and otherwise, performs the insertion and returns 1. If the list is full, before inserting, the list size should be doubled to make room for the insertion (and future insertions). Note that whenever making a change to an IntList, several items must be changed, since the struct manages all information for the list. Here is the struct and initial size constant:

```
#define INIT_SIZE 10

typedef struct IntList {
    int* list;
    int cursize;
    int maxsize;
} IntList;
```

(a) (10 pts) Write the function makeList, which takes nothing in and returns an IntList*. This should be a pointer to a default IntList which has a list of size 10. Note that cursize stores the number of elements currently stored in the list while maxsize represents the current number of ints allocated for the array that list points to. This function should be exactly five lines long, two of which are for memory allocation, 2 for setting the non-pointer variables and one for the return.)

```
IntList* makeList() {








}
```

(b) (16 pts) Write the insert function. This takes in a pointer to the IntList, the index for insertion and the value to insert (called item). If the index is invalid, return 0. Then, if the IntList is full, double the size of the array list is pointing to. Assume that this succeeds. Then, perform the insert and return 1.

```
int insert(IntList* mylist, int index, int item) {








}
```

8) (2 pts) If you were to use the service couchsurfing.com to find a place to stay on vacation, what household furniture might you end up sleeping on?

______________________________

**C Language Reference**

```
// Allocates size bytes of uninitialized storage.
void* malloc(size_t size);

// Allocates a block of memory for an array of num elements,
// each of them size bytes long, and initializes all of its
// bits to 0.
void* calloc(size_t num, size_t size);

// Reallocates the given area of memory. It must be previously
// allocated by malloc, calloc or realloc and not yet freed
// with a call to free or realloc.
// This is done by either expanding or contracting the existing
// area pointed to by ptr, if possible. If not, a new memory
// block of size new_size bytes is allocated, copying memory
// area with size equal the lesser of the new and the old sizes,
// and freeing the old block. If there is not enough memory,
// the old memory block is not freed and null is returned.
void* realloc(void* ptr, size_t new_size);
```

**Scratch Work: Please clearly mark any work below you would like graded.**