COP 3502 Exam #1 Review Questions

Dynamic Memory Allocation

1) A class has *n* students, s_1 through s_n. Student s_i has taken t_i tests, each scored from 0 to 100. This data is entered via standard input using the following format:

First line stores the number of students, *n*.

The next *n* lines store the student data with the i^{th} of these lines storing the test information for student s_i.

Each of these lines starts with the integer, t_i, the number of tests taken by student s_i. This is followed by the t_i test scores for that student, in order.

Here is a small sample file for 3 students who've taken 5, 2 and 9 tests, respectively:

```
3
5 100 90 95 99 100
2 85 86
9 83 88 85 85 89 96 75 83 95
```

Complete the function below so that reads in this information from standard input and returns an array of arrays (the first array has length t_1, the second array has length t_2, etc.). Note since all of the required information is in the input, no parameters are needed for the function.

```
int** getTestData();
```

2) Using the calloc function, write a single line of code to allocate room for `n` variables of type `struct item`. Assume that n is defined as an integer and stores a reasonable positive value and that the type `struct item` is declared. Name the array `myitems`.

3) This question uses the following set of #defines and struct definitions:

```
#define NAMESIZE 20
#define TEAMSIZE 100

typedef struct {
    char first[SIZE];
    char last[SIZE];
    int PID;
    double gpa;
} studentT;

typedef struct {
```

```
    studentT captain;
    studentT members[TEAMSIZE];
} teamT;
```

3) (25 pts) Write a studentcmp function that takes in pointers to two studentT's and compares them using the following criteria:

a) If the first student (pointed to by ptrA) has a lower GPA than the second, a negative integer is returned. If the first student has a higher GPA than the second, a positive integer is returned.

b) If the two GPAs are equal, then ties are broken by strcmp in the string library applied to the last names. (In these cases, if the last names are different, just return what strcmp does.)

c) If the two last names are the same, then those ties are broken by strcmp applied to the first names. (In these cases, if the first names are different, just return what strcmp does.)

d) Break the tie in this case by returning the difference in PIDs.

```
#include <string.h>

int studentcmp(const studentT* ptrA, const studentT* ptrB);
```

Array Problems

1) A dot product between two arrays, A and B, each with n elements, is defined as follows:

$$A \bullet B = A[0]B[0] + A[1]B[1] + A[2]B[2] + ...A[n-1]B[n-1]$$

Write a function that takes in two integer arrays and their size, and returns the dot product of the two arrays. (For a bit a of extra credit, write this function recursively.)

```
// Pre-condition: A and B are both store size elements.
// Post-condition: The dot product of the n-dimension
//          vectors represented by A and B is
//          returned.
int dotproduct(int A[], int B[], int size);
```

2) Interpolation is the technique of looking at two endpoints and using a linear estimation to determine what lies in between. For example, if we know that there were 200 players in MLB in 1950 and that there were 700 players in MLB in 2000, we'd assume that there was an increase of (700 – 200)/(2000 – 1950) = 10 players a year. With this information, we could estimate that there must have been about 600 players in MLB in 1990. If sorted data in an array is "evenly spaced out", then a binary search that uses interpolation (instead of always looking at the middle index in a range) would perform slightly better than a standard binary search. Write a recursive interpolation search, that works just like a binary search, except for how it picks the index in which in searches. Fill in the prototype below.

```
int InterSearch(int array[], int low, int high, int searchval);
```

Summations

1) Determine a closed form solution for the following summation, in terms of n:

$$\sum_{i=n}^{2n-1}(4i+7)$$

2) Find the closed form solution in terms of n for the following summation. Be sure to show all you work.

$$\sum_{k=5}^{2n}(3k-2)$$

3) Determine a **simplified**, closed-form solution for the following summation in terms of n. **You MUST show your work.**

$$\sum_{k=3}^{n+1}(4k-2)$$

4) Determine a closed-form solution for the following sum in terms of n: $\sum_{i=0}^{n}\left(2\sum_{j=n+1}^{3n}(i+j)\right)$.

Base Conversion

1) Convert 1011101_2 to base 10.

2) Convert 274_{10} to binary.

3) Convert 1654_8 to hexadecimal.

4) Convert 1925_{10} to octal.

Big-Oh Problems

1) A algorithm that sorts n items runs in $O(n\sqrt{n})$. When run on an input of 10,000 items, the algorithm takes 200 milliseconds. How long, in seconds, will the algorithm take when run on an input of 90,000 items?

2) A matrix factorization algorithm that is run on a input matrix of size *n* x *n*, runs in $O(n^3)$ time. If the algorithm takes 54 seconds to run for an input of size 3000 x 3000, how long will it take to run on an input of size 1000 x 1000?

3) A string algorithm with inputs of lengths *n* and *m* runs in $O(n^2m)$ time. If the algorithm takes 2 seconds to run on an input with $n = 1000$ and $m = 500$, how long will the algorithm take to execute on an input with $n = 250$ and $m = 1000$?

4) An algorithm runs in $O(\lg_2 n)$ time. It takes 36 ns for an input size of 8. If another run of the algorithm takes 48 ms, how large was the input? Simplify your answer to a single integer.

5) A search algorithm performs a single search on a database of *n* elements in $O(\sqrt{n})$ time. If 1,000,000 of these searches can be performed on a database of size 10000 in 8 seconds, how long would 500,000 searches take on a database of size 640000?

Analyzing Segments of Code

1) Determine the run-time, in terms of the formal parameter n, of the following function. Leave your answer in a Big-Oh bound and justify your answer.

```
int f(int array[], int n) {

    int i, total = 0;
    for (i=0; i<n; i++) {
        int low = 0, high = n-1;
        while (low < high) {
            int mid = (low+high)/2;
            if (2*array[i] < array[mid])
                high = mid-1;
            else
                low = mid+1;
        }
        total += low;
    }
    return total;
}
```

2) What is the run-time (Big-Oh) in terms of n of the following code segments? (Assume all variables have been previously declared as integers.)

a)
```
int i, sum = 0;
for (i=0; i<n; i++) {
  sum += i;
}
for (i=0; i<n; i++)
  sum +=(3*i - 2);
```

b)
```
int i, sum = 0;
while (n > 0) {
  for (i=0; i<n; i++)
    sum++;
  n = n/2;
}
```

c)
```
int i, sum = 0, saven = n;
while (n > 0) {
```

```
    for (i=0; i<saven; i++)
      sum++;
    n = n/3;
  }
```

Recurrence Relations

1) Using the iteration technique, determine the solution to the following recurrence relation:

T(n) = 4T(n/2) + n, T(1) = 1.

2) Solve the following recurrence using the iteration technique:

$T(n) = T\left(\frac{n}{2}\right) + n$, T(1) = 1.

3) Consider the recursive function sum shown below:

```
double sum(int* array, int low, int high){
    if (low == high)
        return array[low];
    int mid = (low+high)/2, left = 0, right = 0, i;
    for (i=low; i<=mid; i++) left += array[i];
    for (i=mid+1; i<=high; i++) right += array[i];
    if (left > right) return left + sum(array, low, mid);
    return right + sum(array, mid+1, high);
}
```

(a) Let T(n) represent the run time of the function call sum(array, 0, n-1), where array is an integer array of size n. Write a recurrence relation that T(n) satisfies.

(b) Using the iteration method, determine a closed-form solution (Big-Oh bound) for T(n). Assume T(1) = O(1).

4) Consider the following function shown below:

```
int countBitsOn(int n) {
    if (n == 0)
        return 0;
    return n%2 + countBitsOn(n/2);
}
```

(a) Let T(n) be the runtime of countBitsOn with an input of size n. Write a recurrence relation that T(n) satisfies, assuming that a constant number of simple operations takes 1 unit of time.

(b) Use the iteration technique to solve for T(n). To simplify the calculation, you may assume that $n = 2^k$ for some non-negative integer k. Note: You may assume T(1) = 1.

5) Please determine a Big-Oh bound for the following recurrence using the iteration technique:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$

Note: $\sum_{i=0}^{k-1} 4^i = \frac{4^k - 1}{3}$.

Recursion

1) We define a set of strings s_1, s_2, ... as follows: s_1 = "1" and to form s_{i+1} we stick together two copies of s_i next to each other followed by the character i+1. For example, s_2 = 112 and s_3 = 1121123. Write a function that takes in n (guaranteed to be in between 1 and 9, inclusive) and prints out s_n.

```
void printSequence(int n);
```

2) What is the return value of the function call f(3, 5), where f is the function shown below?

```
int f(int a, int b) {
    if (a == 0) return 1;
    return b*f(a-1,b-1)/a;
}
```

3) The Catalan numbers, C_k, can be defined as follows: $C_0 = 1$ and $C_n = \frac{2(2n+1)}{n+2} C_{n-1}$, for all integers n > 0. (Note: The definition above produces only integers for Catalan numbers.) Write a recursive function that computes the n^{th} Catalan number. Do not worry about overflow issues for the purposes of this question. You will get full credit if you use recursion and if your code is "theoretically" correct. **(But do worry about integer division and its effects.)**

```
int catalan(int n);
```

4) Complete the program below so that it prints out all the permutations of 0,1,2, ...,SIZE-1 such that the absolute value of the difference between each pair of adjacent numbers in the permutations is 2 or greater. For example, when SIZE = 4, the code would print out:

```
1 3 0 2
2 0 3 1
```

the only 2 permutations such that the absolute value of the difference between each pair of adjacent terms is 2 or greater.

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

#define SIZE 4
```

```
void printPerms(int perm[], int used[], int k, int n);
void print(int perm[], int n) ;

int main() {
    int perm[SIZE], used[SIZE], i;
    for (i=0; i<SIZE; i++) used[i] = 0;
    printPerms(perm, used, 0, SIZE);
    return 0;
}

void printPerms(int perm[], int used[], int k, int n) {

    if ( ____________ ) print(perm, n);

    int i;
    for (i=0; i<n; i++) {
        if (!used[i]) {
            if ( __________ || ____________________________________ ) {

                used[i] = ______ ;

                perm[k] = ______ ;

                printPerms(______ , ______ , ______ , ______ );

                used[i] = ______ ;
            }
        }
    }
}
```

5) Write a **recursive** function that returns 1 (true) if the characters in the array word from index start to index end, inclusive, forms a palindrome, and 0 (false), otherwise. Note: A palindrome is a string that reads the exact same forwards and backwards. For the purposes of this question, a palindrome is case sensitive. Thus, "Madam" is NOT a palindrome.

```
int isPal(char word[], int start, int end);
```

6) A subsequence in a list of numbers is simply a subset of the values in the list, in the same order as they appear in the list, relatively. For example, the list 2, 6, 3, 1, 8, 7, 9 has the subsequence 2, 3, 7 and 9. (Note that the values chosen do NOT need to be contiguous.) Write a **recursive function** that takes in an array, an integer n, an integer k , and an integer max, and returns the number of sorted subsequences with k values with the maximum value max out of the first n values in the input array. For example, in the array above of length 7, there are 4 subsequences of length 4 with values less than or equal to 9 ([2,3,7,9], [2,3,8,9], [2,6,7,9] and [2,6,8,9]), and there are 10 subsequences of length 2 with values less than or equal to 8. ([2,6], [2,3], [2,8], [2,7], [6,8], [6,7], [3,8], [3,7], [1,8] and [1,7] .) Do not worry about any overflow issues. You may assume that $n \geq 0$

and k ≥ 0. Note that a sorted subsequence may have a repeated value, even though the examples above don't illustrate that situation.

```
int numSortedSubSeq(int* array, int n, int k, int max);
```

Sorting

1) Show the results of running the partition function on the array below, using the element originally in index 0 as the partition element. Use the same partition method outlined in class.

Initial	5	3	7	2	1	6	8	4
After Partition								

2) Write the code for bubble sort.

3) Write the code for insertion sort.

4) Write the code for selection sort.

5) Please provide the best case and worst case run times (Big-O) for each of the following three sorting algorithms, in terms of n, the number of elements being sorted.

Sort	Best Case	Worst Case
Merge Sort		
Quick Sort		
Insertion Sort		

6) Consider running a merge sort on the array shown below. Show the contents of the array after each merge occurs. (Note: There are 7 merges.)

Index	0	1	2	3	4	5	6	7
Value	17	13	27	9	18	15	2	8

7) Which of the following recurrence relations best describes the worst case run time of Quick Sort of n elements? (Circle the correct answer.) Explain why the solution to this recurrence corresponds to the worst case run time a Quick Sort of n elements.

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

(c) $T(n) = 2T(n-1) + O(n)$ (d) $T(n) = T(n-1) + O(n)$