2017 Spring COP 3502 Exam #1 Solution

Date: 2/16/2017

There is a wine enthusiast club with many members. You've been tasked to store a database of information for each of these club members. On the next page you will answer two questions about this scenario.

Here is the struct that stores information about a single wine and a member in the club:

```
typedef struct wine {
    char type[20];
    char brand[20];
    int rating;
} wine;
```

```
typedef struct member {
    char name[20];
    int numRated;
    wine* list;
} member;
```

Here, list is a dynamically allocated array of size numRated, indicating each wine the user has rated. (The member's rating of a wine is stored in the wine struct within that member struct.)

In doing so, you will be reading in input (from standard input) with the following format:

First line stores an integer, *n*, the number of members in the club. The member info follows. The first line for each member stores their name (string of 1-19 letters) and, *w*, number of wines they've rated, separated by a space. The following *w* lines store information about the wines this member has rated. Each of these lines has the type (string of 1-19 letters), followed by the brand (string of 1-19 letters), followed by the rating (1-10), all separated by spaces.

1) (14 pts) Write a segment of code that allocates an array of member and reads in the input described in the format above, from standard input.

```
int i, j, n;
scanf("%d", &n);                                        // 1 pt
member* club = malloc(n*sizeof(member));                // 2 pts

for (i=0; i<n; i++) {                                   // 1 pt
    int nWines;
    scanf("%s%d", club[i].name, &nWines);               // 1 pt
    club[i].numRated = nWines;                          // 1 pt
    club[i].list = malloc(nWines*sizeof(wine));         // 2 pts
    for (j=0; j<nWines; j++)                            // 1 pt
        scanf("%s%s%d", club[i].list[j].type,           // 5 pts
              club[i].list[j].brand, &club[i].list[j].rating);
}
```

2) (6 pts) Free all of the memory allocated in question 1.

```
for (i=0; i<n; i++)             // 1 pt
    free(club[i].list);         // 3 pts
free(club);                     // 2 pts
```

3) (15 pts) Solve the following recurrence relation, ***exactly***, in terms of n, using the iteration technique:

$$T(n) = 2T(n-1) + 2^n, T(1) = 2$$

Solution

Use the iteration technique:

$$T(n) = 2T(n-1) + 2^n$$
$$T(n) = 2(2T(n-2) + 2^{n-1}) + 2^n$$
$$T(n) = 4T(n-2) + 2^n + 2^n$$
$$T(n) = 4T(n-2) + 2(2^n)$$
$$T(n) = 4(2T(n-3) + 2^{n-2}) + 2(2^n)$$
$$T(n) = 8T(n-3) + 2^n + 2(2^n)$$
$$T(n) = 8T(n-3) + 3(2^n)$$

Now, we guess our solution after k iterations to be:

$$T(n) = 2^k T(n-k) + k(2^n)$$

Since we know T(1), we plug in k = n-1:

$$\begin{aligned} T(n) &= 2^{n-1}T(n-(n-1)) + (n-1)(2^n) \\ &= 2^{n-1}T(1) + (n-1)(2^n) \\ &= 2^{n-1}(2) + (n-1)(2^n) \\ &= 2^n + (n-1)(2^n) \\ &= n2^n \end{aligned}$$

Grading: 3 pts for second iteration ($4T(n-2) + 2(2^n)$), 3 pts for third iteration ($8T(n-3) + 3(2^n)$), 2 pts for general guess, 1 pt for plugging in k = n - 1, 3 pts for algebra to get to final answer in simplified form.

4) (7 pts) An O(nlgn) algorithm runs in 20 ms on an input of size n = 2^{20}. How long will it take to run on an input of size n = 2^{25}? (Note: The multiplication is made easier by realizing that 25 x 4 = 100.)

Solution

Let T(n) be the run time of the algorithm. Plugging in the given information we have:

$$T(n) = cnlgn$$
$$T(2^{20}) = c2^{20}\lg(2^{20}) = 20ms$$
$$c2^{20}(20) = 20ms$$
$$c = \frac{1}{2^{20}}ms$$

Now, solve for T(2^{25}):

$$T(2^{25}) = c2^{25}\lg(2^{25}) = \frac{1}{2^{20}}ms \times 2^{25} \times 25 = 2^5 \times 25ms = 8 \times (4 \times 25)ms = 800ms$$

Grading: 3 pts for obtaining c, 4 pts for solving and simplifying the result. Note that .8 seconds (or any equivalent) is also accepted since the units of the final answer weren't specified in the question.

5) (8 pts) Determine the following sum in terms of n: $\sum_{i=n+1}^{n^2} i$. (Note: You can get a bonus point for fully factorizing your answer!)

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

For extra credit (2 pts), you could do $\frac{n^4-n}{2} = \frac{n(n^3-1)}{2} = \frac{n(n-1)(n^2+n+1)}{2}$.

Grading: 2 pts split sum, 2 pts sum to n^2, 2 pts sum to n, 2 pts simplify in poly form. Only give 2 pts EC if they factored the cubic. NO EC for just pulling out the n...

6) (10 pts) A geometric sequence is a sequence where each pair of consecutive terms has a common ratio. For example, 3, 6, 12, 24 is a geometric sequence of 4 terms with a common ratio of 2. Write a ***recursive*** function that takes in three integers: `firstTerm`, the first term of a geometric sequence, `ratio`, the common ratio of the sequence, and `numTerms`, the number of terms of the sequence and returns the sum of the geometric sequence. You may assume that the input values are such that no overflow errors occur on any necessary calculation.

```
int geoSum(int firstTerm, int ratio, int numTerms) {

    if (numTerms == 0)
        return 0;
    return firstTerm + geoSum(firstTerm*ratio, ratio, numTerms-1);
}
```

Grading: BC 3 pts (also nT==1 ret firstTerm is acceptable), 1 pt return 1 pt adding first term, 2 pts rec call, 2 pts new first term, 1 pt for new numterms, 0 pts ratio (allow this error essentially without taking off any credit).

7) (6 pts) The recursive function f is defined below. What do the expressions f(3), f(4) and f(5) evaluate to?

```
int f(int n) {
    if (n == 0) return 3;
    int sum = 0, i;
    for (i=0; i<n; i++)
        sum += f(i);
    return sum;
}
```

f(3) = **12** f(4) = **24** f(5) = **48**

Grading: 2 pts each, no partial credit.

8) (14 pts) Complete the code below so that it prints out all permutations of the string entered by the user. You may assume that the string entered by the user is comprised of 1-10 distinct lowercase letters.

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

void printPerms(int perm[], int used[], int k, int n, char*
letters);

int main() {
    char word[11];
    printf("Enter a word (1-10 distinct letters).\n");
    scanf("%s", word);
    int perm[10], used[10];
    int i;
    for (i=0; i<strlen(word); i++)
        used[i] = 0;
    printPerms(perm, used, 0, strlen(word), word);
    return 0;
}

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

    int i;

    if (k == n) {
        for (i = 0 ; i < n; i++)                    // 1 pt , 1 pt

            printf("%c", letters[perm[i]] );        // 4 pts
        printf("\n");
    }

    else {
        for (i=0; i<n; i++) {
            if ( !used[i] ) {                       // 2 pts

                used[i] = 1;                        // 1 pt

                perm[k] = i;                        // 1 pt

                printPerms(perm, used, k+1 , n  , letters );// 3 pts

                used[i] = 0;                        // 1 pt
            }
        }
    }
}
```

9) (4 pts) Show the result of running the partition function on the array below, using the item in index 0 as the partition element.

Index	0	1	2	3	4	5	6	7	8
Original	45	87	22	13	78	19	74	41	39
Partition	**19**	**39**	**22**	**13**	**41**	**45**	**74**	**78**	**87**

Grading: 2 pts location partition element, 1 pt for left of partition, 1 pt for right of partition, if there are small errors in the left or right, just take off 1 pt total.

10) (6 pts) Show the contents of the following array after each iteration of Insertion Sort.

Index	0	1	2	3	4	5	6	7
Original	99	13	76	22	78	19	3	41
After 1st Iteration	**13**	**99**	**76**	**22**	**78**	**19**	**3**	**41**
After 2nd Iteration	**13**	**76**	**99**	**22**	**78**	**19**	**3**	**41**
After 3rd Iteration	**13**	**22**	**76**	**99**	**78**	**19**	**3**	**41**
After 4th Iteration	**13**	**22**	**76**	**78**	**99**	**19**	**3**	**41**
After 5th Iteration	**13**	**19**	**22**	**76**	**78**	**99**	**3**	**41**
After 6th Iteration	**3**	**13**	**19**	**22**	**76**	**78**	**99**	**41**
After 7th Iteration	3	13	19	22	41	76	78	99

Grading: 1 pt per row, row has to be perfect to get the point.

11) (9 pts) What are the best case, worst case and average case run times of each of the following sorting algorithms, for sorting n integer?

Algorithm	Best Case	Worst Case	Average Case
Insertion Sort	**O(n)**	**$O(n^2)$**	**$O(n^2)$**
Selection Sort	**$O(n^2)$**	**$O(n^2)$**	**$O(n^2)$**
Quick Sort	**O(nlgn)**	**$O(n^2)$**	**O(nlgn)**

Grading: 1 pt each, no partial

12) (1 pt) What color is the cleaning product Simple Green? **GREEN (1 pt give to all)**