CMPT 125 Assignment 2 Solutions

Question 1
(20 marks total)

a) Let’s consider an integer array of size 10. (10 marks, each part is 2 marks)

```
int a[10];
```

I. How would you assign a pointer, called pA, to store the address of element 0 of this array? Write the C code for your answer.

```
int *pA = &a[0]; or int *pA = a;
```

II. Using pA, how would you obtain the value of the next element (element 1) of the array?

```
pA points to the first element of the array. So the next element would be *(pA+1)
```

III. Explain in 2-3 sentences what the statement `pA = a[1];` would do.

The statement would change the address to which pA points to the value stored in a[1], which is meant to store an integer, not an address. While this code will compile, trying to access the value of pA by using *pA may cause a segmentation fault (memory error).

IV. Is the statement `a = pA;` valid (would it cause compilation errors)? How about `a++;`? Explain briefly in 3-4 sentences.

No. There is one key difference between an array name and pointer. A pointer is a variable, so pA=a and pA++ are legal. But an array name is not a variable; constructions like a=pA and a++ are thus illegal.

V. Write C code to print all the elements from this array.

One possible solution:

```
#include <stdio.h>

int main() {
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    for (int *i = a; i <= a+9; i++) {
        printf("%d\n", *i);
    }
}
```

b) Consider the two variables below. (5 marks)

```
char amessage[] = "now is the time";
char *pmessage = "now is the time";
```
I. What is the difference between the two variables amessage and pmessage? (1 mark)

char amessage[] is an array, whereas char *pmessage is a pointer initialized to point to a string constant.

II. Write a C code to change the character ‘t’ from ‘time’ to uppercase ‘T’ in the variable amessage. (5 marks)

One possible solution:

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

int main() {
    char amessage[] = "now is the time";
    int c = 0;
    while (amessage[c] != '\0') {
        if(amessage[c] == 't' && amessage[c+1] == 'i') {
            amessage[c] = 'T';
        }
        printf("Letter: %c \n", amessage[c]);
        c++;
    }
}

c) Consider the following code. (5 marks)

void strcopy(char s1[], char s2[]) {
    int len = strlen(s2);
    for (int i = 0; i<=len; i++) {
        s1[i] = s2[i];
    }
}

int main () {
    char s2[10] = "copy this";
    char s1[10];
    strcopy(s1, s2);
}

The above code copies the character from s2 to s1 using the concept of array, in the strcopy function. Implement another function strcopy2 that achieves the same result, but that takes as input pointers to character arrays – that is, complete the strcopy2 function in the following code so that the code copies the content of s2 into s1.

void strcopy2(char *s1, char *s2) {
}
while (*s2 != '\0') {
    *s1 = *s2;
    s1++;
    s2++;
}

int main () {
    char s2[10] = "copy this";
    char s1[10];

    strcopy2(s1, s2);
}
Question 2
(10 marks, each part is 2 marks)

Assume that each of the expressions below gives the processing time $T(n)$ spent by an algorithm for solving a problem of size $n$. Find the dominant term(s) having the steepest increase in $n$ and specify the lowest Big-O complexity of each algorithm.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Dominant term(s)</th>
<th>$O(...)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.001 \log_4 n + \log_2 (\log_2 n)$</td>
<td>$0.001 \log_4 n$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$n^3 \log_2 n + n(\log_2 n)^2$</td>
<td>$n^3 \log_2 n$</td>
<td>$O(n^3 \log n)$</td>
</tr>
<tr>
<td>$0.01n + 100n^3$</td>
<td>$100n^3$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>$0.1n + 4n^{1.5} + 2.5n^{1.5}$</td>
<td>$4n^{1.5}$, $2.5n^{1.5}$</td>
<td>$O(n^{1.5})$</td>
</tr>
<tr>
<td>$3\log_8 n + \log_2 (\log_2 (\log_2 n))$</td>
<td>$3\log_8 n$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Note that the base of $\log n$ doesn't matter (by default, it's base 2 in this class). In addition, constants are dropped when using the Big-O notation. To compare two terms to see which is dominant, one can take their ratio and let $n \to \infty$.

**Example 1:** Initial evaluation of gives $\frac{\infty}{\infty}$, so we use L'Hopital's rule and take the derivative of the top and bottom.

\[
\lim_{n \to \infty} \frac{\log_2 (\log_2 n)}{0.001 \log_4 n} = 1000 \lim_{n \to \infty} \frac{\frac{1}{\ln 2} \cdot \frac{1}{\ln 2} \cdot \frac{1}{n}}{\frac{1}{\ln 4} \cdot \frac{1}{n}} = \text{constant} \cdot \lim_{n \to \infty} \frac{1}{n} = 0
\]

This means that the $0.001 \log_4 n$ term becomes much larger as $n \to \infty$. Note that the constant in front, no matter how small, does not make a difference as $n \to \infty$.

**Example 2:** First we can cancel a bunch of things.

\[
\lim_{n \to \infty} \frac{n(\log_2 n)^2}{n^3 \log_2 n} = \lim_{n \to \infty} \frac{\log_2 n}{n^2}
\]
At this point one can see that the $n^2$ will is much bigger than the $\log_2 n$, so the dominant term is $n^3 \log_2 n$. However, to be completely pedantic, let's take $n \to \infty$. Initial evaluation of gives $\frac{\infty}{\infty}$, so we use L'Hopital's rule and take the derivative of the top and bottom.

$$
\lim_{n \to \infty} \frac{\log_2 n}{n^2} = \lim_{n \to \infty} \frac{\frac{1}{\ln 2} \cdot \frac{1}{n}}{2n} = \frac{1}{\ln 2} \lim_{n \to \infty} \frac{1}{2n^2} = 0
$$

Question 3
(8 marks, each part is 2 marks)

For the following parts, try to get the best Big-O estimate that you can and briefly justify your answers.

**Part a)**

```c
int i, j;
int n = 100;
for (i = 1; i <= n; i++) {
    for (j = 3*i; j <= n; j++) {
        printf("programming is fun\n");
    }
}
```

The inner loop goes from $j = 3i$ to $j = n$. This is $n - 3i + 1$ times. Now, $i$ goes from 1 to $n$, so the total number of printf calls is $\sum_{i=1}^{n}(n - 3i + 1)$. This is at most $\sum_{i=1}^{n}(n)$, which is $n^2$, so the total running time is $O(n^2)$.

**Part b)**

```c
int i, j;
int n = 1000000;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= 10000; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Outer loop is executed $O(n)$ times. For every execution of the outer loop, inner loop is executed 10000 times, which is a constant number of times. So, total running time is $10000n = O(n)$.

**Part c)**

Page 5 of 7
```c
int i = 0;
int n = 10;
int j;

while (i < n) {
    i++;
    j = i;
    while (i < n) {
        printf("hello %d\n", i);
        i++;
    }
    i = j;
}
```

Outer loop executes $n$ times. For every execution of the outer loop, the inner loop is executed for $j = i$ (the value of $i$ in the outer loop) to $n$, which is $n - i + 1$ times. So the total execution count of printf is $\sum_{i=1}^{n}(n - i + 1)$, which is at most $\sum_{i=1}^{n}(n) = n^2 = O(n^2)$.

**Part d)**

```c
int i = 0;
int n = 10;
int j;

while (i < n) {
    i++;
    j = i;
    while (i < n) {
        printf("hello %d\n", i);
        i++;
        break;
    }
    i = j;
}
```

Outer loop executes $O(n)$ times. Inner loop has a break statement so it becomes $O(1)$, constant. Total running time is $O(n)$.

**Question 4**

(15 marks)

Write a program in C that prompts a user to select any five beverages of your choice (e.g. Coke, Lemon Tea, etc.). User should be able to select one of these options using only integers from 0 to 4 as inputs. Once a user has made the choice, print the chosen drink along with its calorie information. You may look up the calorie information online.

Example input: 3

Example output:

```
Selected drink: Coca Cola
Calorie count: 140 Calories
```

Page 6 of 7
Notes:
- You can assume that the user will input integers. You do not need to handle invalid inputs of other types (such as strings, floats, etc.).
- Your program should gracefully terminate if a user enters wrong input three times. In addition, after every invalid attempt, your program should tell the user how many tries he/she has remaining. For example, the prompt should work something like this:
  o Prompt 1: User enters 6. Your program should tell that input is invalid and enter again. Also print the remaining tries (2).
  o Prompt 2: User enters -1. Your program should tell that input is invalid and enter again. Also print the remaining tries (1).
  o Prompt 3: User enters 200. Your program should tell that input is invalid and terminate. Also print the remaining tries (0).
  o If the user enters a correct input on or before the last try, the program should behave normally, and output should be like the provided example.

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

int beverages() {
    printf("Please make a selection of your favourite beverage by typing a number\n"
           "0 - Coca Cola\n"
           "1 - Pepsi\n"
           "2 - Lemon Tea\n"
           "3 - Redbull\n"
           "4 - Coffee\n");

    int tries = 3;
    int selection;

    const char *display[] = {
        "Selected drink: Coca Cola. Calorie count: 140 Calories\n",
        "Selected drink: Pepsi. Calorie count: 150 Calories\n",
        "Selected drink: Lemon Tea. Calorie count: 90 Calories\n",
        "Selected drink: Redbull. Calorie count: 200 Calories\n",
        "Selected drink: Coffee. Calorie count: 160 Calories\n",
    };

    while(tries > 0) {
        tries--;
        printf("Enter a number\n");
        scanf("%d", &selection);

        if(selection >= 0 && selection <= 4) {
            printf(display[selection]);
            break;
        } else {
            printf("Input is invalid. Remaining tries (%d)\n", tries);
        }
    }
    return 0;
}
```