hw2

CS 302
Data Structures and Algorithm Analysis

Indiana State University

Homework II: Asymptotic Analysis

Name: Tristan Gale

Question	Points	Score
Problem 1	10	
Problem 2	10	
Problem 3	10	
Problem 4	20	
Total:	50	

Problem 1. Consider two non-negative functions $T_1(N)$ and $T_2(N)$. We know that $T_1(N)$ is $\mathrm{O}(F(N))$ and $T_2(N)$ is $\mathrm{O}(F(N))$ for some non-negative function $F(N)$.

For each of the statements below, answer with `True` or `False`. If the statement is `True`, you **must** explain why using the formal definition of O. In other words, you must find constants c and N_0 such that the definition of O is satisfied.

If the statement if `False`, you **must** give a counter example for which the definition would not be satisified. In other words, give examples for $T_1(N)$, $T_2(N)$, and F for which the definion of O is not satisfied.

(a) (5 points)

$$\frac{T_1(N)}{T_2(N)} \in \mathrm{O}(1)$$

$\frac{T_1(N)}{T_2(N)}$ is $O(1)$

$T_1(n)$ is $O(F(n))$, $T_1(n) \leq c_1 \cdot F(n)$

$T_2(n)$ is $O(F(n))$, $T_2(n) \leq c_2 \cdot F(n)$

$\frac{c_1 \cdot F(n)}{c_2 \cdot F(n)} = \frac{c_1}{c_2}$ is $O(1)$,

$\frac{c_1}{c_2} \leq c.1$ True.

(b) (5 points)

$T_1(N)$ is $O(T_2(N))$

$T_1(n)$ is $O(T_2(n))$ $T_1(n)$ is $O(F(n))$

$T_1(n) \leq c \cdot T_2(n)$ for all $n \geq n_0$. $T_2(n)$ is $O(F(n))$

lets say $F(n) = N^4$ $T_1(n) \leq c \cdot F(n)$

$T_1(n) = N^3$ $T_2(n) \leq c \cdot F(n)$.

$T_2(n) = N^2$. $T_1 + T_2 = O(F(n))$.

However, $T_1(n)$ is NOT $O(T_2(n))$

False.

Problem 2. (10 points) Mohammad claims that when dealing with asymptotic analysis, the base in an exponentiation does not matter. Specifically, Mohammad claims that 10^N is $\Theta\left(2^N\right)$.

Do you think that Mohammad's claim is correct? Support your answer using the formal definition of Θ (also that of Ω and O).

Hint: This is easier than it looks.

Hint: Can you express 10^N as a function of 2^N?

$a^b = c^{b \cdot \log_c a}$

$10^N = 2^{N(\log_2 10)}$ ← constant.

$c_1 \cdot 2^N \leq 10^N \leq c_2 \cdot 2^N$ for all $N \geq N_0$

$1 \cdot 2^N < 10^N \leq 2^{N \cdot \log_2 10}$ for all $N \geq 0$

$=$

yes 10^N $\Theta(2^N)$

Problem 3. For each of the following code fragments, find the exact number of times that the statement `sum++` would execute, and then give your Θ analysis of its runtime. **You do not have to prove your Θ analysis, you can simply state it.**

(a) (5 points)

```
for(int i = 0; i < n; i++) {
  for(int j = 0;  j < n * n; j++) {
    sum++;
  }
}
```

Hint: Look carefully at the loop conditions and iterator variables.

code is $\Theta(n^3)$

(b) (5 points) For the code fragment below, you may assume that n is a power of 2.

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

code is $\Theta(n \cdot \log n)$

Problem 4. Given a string of characters, you are given the task of finding whether the characters in that string are unique or not. In other words, you are to find whether any character in that string appears more than once, if so, then the characters are not unique, otherwise, they are.

Here's a couple of examples:

1. Consider the string $\{a, b, c, k, l, a, n\}$. The characters here are not unique since a appears more than once.

2. Consider the string $\{a, k, l\}$. The characters in this string are unique since each of them appear **exactly once** in the string.

Constraint 1: In all of the questions below, we will restrict our attention to lower case alphabet characters, i.e., `a` through `z` only. The problem wouldn't change much if we drop this constraint, but it just makes it nicer.

Constraint 2: In all of the questions below, you are only allowed to use lists (i.e., arrays), you should not use any other data structure (e.g., hash map, set, etc.).

Hint: In all of the questions below, you may assume that the length of your input string is n.

(a) (5 points) Give a naive algorithm to check whether the characters in a string are unique. You can write this in any language you would like or a simple pseudocode.

```
for(i = 0; i < n-1; i++){
    for(j=i+1; j < n; j++) {
        if (string[i] == string[j]) {
            return false
        }
    }
}
return True.
```


(b) (10 points) What is the asymptotic runtime of your naive algorithm above? Show your work. You should be able to count the number of times your most frequent line is executed, then propose an asymptotic runtime using Θ.

```
for(i = 0; i < n-1; i++){   ← T(n)
    for(j=i+1; j < n; j++) {
        if (string[i] == string[j]){
            return false
        }
    }
}
return True.
```

$i=0, j=1,2,3 \dots n-1, n-1$
$i=1, j=2,3,4 \dots n-1, n-2$
$\vdots$
$i=n-3, j=n-2, n-1, 2$
$i=n-2, j=n-1, 1$

$F(n)$ is $O(n^2)$ $\quad \sum_{1}^{n-1} k = \frac{(n-1)(n-1+1)}{2} = \frac{n^2-n}{2}$

$F(n)$ is $\Theta(n^2)$ such that $c_1 \cdot n^2 \le F(n) \le c_2 \cdot n^2$ $\quad c_1 = .25 \quad c_2 = 1$ for all $n \ge 2$

(c) (5 points) Can you suggest an algorithm that does better than the naive solution? What is the asymptotic runtime of your improved algorithm?

Hint: You might find that using another array can be helpful.

Hint: How do we represent characters in programming languages?

Hint: You should be able to do this in $\Theta(n)$.

```
hashmap = {}
for i in range(n):
    if string[i] in hashmap:
        return false
    hashmap[i] = string[i]
return true.
```

$F(n)$ is $\Theta(n)$