Dunya Oguz 40181540
Hugo Joncour 40139230
Sarabraj Singh 29473858
John Purcell 27217439

Assignment 1

Theoretical Part:

1) What is the largest size ***n*** of problem which can be solved in one second by an algorithm with the following running time functions, in microseconds (1 second = 1,000,000 microseconds) ?

A: $\log_2(n)$

$$log_2(n) = 1000000$$

$$n = 2^{1000000}$$

B: $\sqrt{n}$

$$\sqrt{n} = 1000000$$

$$n^{(1/2)} = 1000000$$

$$n = (10^6)^2$$

$$n = 10^12$$

C: n

$$n = 1\ 000\ 000$$

D: n^2

$$n^2 = 1\ 000\ 000$$

$$n^2 = 10^6$$

$$n = 10^3$$

E Option 1: $log_2 n^{(log_2(n))}$

$$log_2 n^{(log_2(n))} = 10\text{^}6$$

$$log_2 n * log_2 n = 10^6$$

$$(log_2 n)^2 = 10^6$$

$$log_2 n = 10^3$$

$$2^{10^3} = n$$

$$n = 1.07151 * 10^{301}$$

Dunya Oguz 40181540
Hugo Joncour 40139230
Sarabraj Singh 29473858
John Purcell 27217439

E Option 2: $log_2(n)^{(log_2(n))}$

$$log_2(n)^{(log_2(n))} = 1000000$$

$$(log_2(n))^{(log_2(n))} = 10\text{^}6$$

Let's assume $log_2(n) = y$, then $y^y = 10^6$

Solving for y: $y = \frac{log_e(10^6)}{LambertW(log_e(10^6))}$

But y = $log_2(n)$ so $log_2(n) = \frac{log_e(10^6)}{LambertW(log_e(10^6))}$

so $n = 2^{\left(\frac{12.8155}{1.9552}\right)}$

so $n \approx 133.973$

2) Justify the following statements using the "big-Oh" definition:

A: $(n+25)^2$ is *O(n²)*

$$n^2 + 2 * 25 * n + 25^2 \quad \text{is } O(n^2)$$

$$n^2 + 50n + 625 \leq (1 + 50 + 625)n^2 = cn^2$$

A: n^2 is $O((n+25)^2)$

$$O((n + 25^2)) = O(n^2)$$

$$n^2 \quad \text{is } O(n^2)$$

B: n^3 is not $O(n^2)$

Proof by contradiction:

Let us assume that $n^3 = O(n^2)$ for all $n \geq 1$ then there is a c that exists in $[c < \infty[$ such that $n^3 \leq cn^2$

$$\frac{n^3}{n^2} \leq c \rightarrow n \leq c$$

This inequality should hold for all (n) but it can be easily broken when $n > c$

Thus $(n^3) \neq O(n^2)$ or (n^3) is not $O(n^2)$

C: Given:

- $f_1(n) = (n+25)^2$
- $f_2(n) = n^3$

What is the big-Oh for $f_1(n)$ x $f_2(n)$?

$$n^3 * (n^2 + 50n + 625) = n^5 + 50n^4 + 625n^3$$

$\rightarrow f_1(n)$ x $f_2(n)$ is $O(n^5)$

Dunya Oguz 40181540
Hugo Joncour 40139230
Sarabraj Singh 29473858
John Purcell 27217439

3) Give the asymptotic (“big-Oh”) running time complexity of the following algorithm, show all the work you have done.

```
Algorithm: ArrayMangle(A[ ], int n)
Input: an array A, an integer n
x = 0;                                  // This operation z is executed  in O(1)
for (i=0; i<=n-1; i++) {                // The loop “i” has n iterations
        for (j=i; j<=n-1; j++) {        // The loop “j” has n-i iterations
                x =  x + A[j];          // This operation a is executed  in O(1)
        }
        for (k=0; k<= n-1; k++) {       // The loop “k” has n iterations
                for (j=0; j< =n-1; j++) {   // The loop “j” has n iterations
                        x =  x + A[j]*A[k]; // This operation b is executed in O(1)
                }
        }
}
```

The time complexity of this algorithm is given by the number of operations:

The operation a is executed $(n - i)$ times inside the j loop
The j loop is inside the i loop, so the operation a is executed $n * \left(\Sigma(n - i)\right)$ times.

The operation b is executed n times in the j loop, which is executed n times in the k loop, which is executed n times in the i loop.

So, our algorithm’s complexity is given by: $n * \left(\Sigma(n - i)\right) + n * n * n$

By linearity of the sum we get: $n * (\mathrm{n} - \mathrm{i}) + n * n * n$

So: $n * (\mathrm{n} - \mathrm{i}) + n^3$

The big-Oh notation of the complexity is $O(n^3)$