Assignment 1

Q.1. In each of the following situation, indicate whether $f = O(g)$ or $f = \Omega(g)$ or both ($f = \Theta(g)$)

a. $f(n) = n - 100$
$g(n) = n - 200$

$\rightarrow$ $f = O(g)$
$g = O(f)$ or $f = \Omega(g)$

Therefore, $\underline{\underline{f(n) = \Theta(g(n))}}$

b. $f(n) = n^{1/2}$
$g(n) = n^{2/3}$

$\rightarrow$ $n^{1/2} < n^{2/3}$

Therefore $\underline{\underline{f(n) = O(g(n))}}$

c. $f(n) = 100n + \log n$, $g(n) = n + (\log n)^2$

$\rightarrow$ both are $O(n)$ that is $f = O(g)$ and $g = O(f)$

Therefore $\underline{\underline{f(n) = \Theta(g(n))}}$

d. $f(n) = n \log n$, $g(n) = 10 n \log 10 n$

$\rightarrow$ $f = O(g)$ and $g = O(f)$ or $f = \Omega(g)$

Therefore $\underline{\underline{f(n) = \Theta(g(n))}}$

e. $f(n) = \log 2n$, $g(n) = \log 3n$

$\Rightarrow f(n) = \log 2 + \log n$
$g(n) = \log 3 + \log n$

Therefore $\underline{f(n) = \Theta(g(n))}$

f. $f(n) = 10 \log n$, $g(n) = \log(n^2)$

$g(n) = 2 \cdot \log n$
both are $O(n \log n)$ that is $f = O(g)$ and $g = O(f)$
Therefore $\underline{f(n) = \Theta(g(n))}$

g. $f(n) = n^{1.01}$, $g(n) = n \log^2 n$

$\underline{f(n) = \Omega(g(n))}$

H) $f(n) = n^2 / \log n$, $g(n) = n(\log n)^2$

$\underline{f(n) = \Omega(g(n))}$

I) $f(n) = n^{0.1}$, $g(n) = (\log n)^{10}$

$\underline{f(n) = \Omega(g(n))}$

J) $f(n) = (\log n)^{\log n}$, $g(n) = n / \log n$

$f(n) = n^{\log \log n}$
Therefore $\underline{f(n) = \Omega(g(n))}$

K. $f(n) = \sqrt{n}$, $g(n) = (\log n)^3$

Let $n = 2^k$

$f(n) = 2^{k-1}$

$g(n) = k^3$

Therefore $\underline{\underline{f(n) = \Omega(g(n))}}$

L. $f(n) = n^{3/2}$, $g(n) = 5^{\log_2 n}$

$g(n) = n^{\log_2 5} \approx n^{2.32}$

$n^{3/2} < n^{2.32}$

Therefore $\underline{\underline{f(n) = O(g(n))}}$

M. $f(n) = n2^n$, $g(n) = 3^n$

$2^n < 3^n$

Therefore $\underline{\underline{f(n) = O(g(n))}}$

N. $f(n) = n2^n$, $g(n) = 3^n$

$f = O(g)$

$g = O(f)$ or $f = \Omega(g)$

Therefore $\underline{\underline{f(n) = \Theta(g(n))}}$

O. $f(n) = n!$, $g(n) = 2^n$

$n! = \Theta(n \log n)$

Therefore $\underline{\underline{f(n) = \Omega(g(n))}}$

P) $f(n) = (\log n)^{\log n}$, $g(n) = 2^{(\log_2 n)^2}$

$$f(n) = n^{\log \log n}$$

$$g(n) = \left(2^{\{\log_2 n\}}\right)^{\log_2 n} = n^{\log_2 n}$$

Therefore $\underline{\underline{f(n) = O(g(n))}}$

Q.) $f(n) = \sum_{i=1}^{n} i^{k}$, $g(n) = n^{k+1}$

$\Rightarrow$ both are $O(n)$ that is $f = O(g)$ and $g = O(f)$ or $(f = \Omega(g))$

Therefore $\underline{\underline{f(n) = \Theta(g(n))}}$

Q.4) Is there a faster way to compute the n^{th} Fibonacci number than by fib2? One idea involves matrices

We start by writing the equation $F_1 = F_1$ and $F_2 = F_0 + F_1$ in matrix notation.

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

In General

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

So in order to compute F_n, it suffices to raise this 2×2 matrix, call it X, to the n^{th} power.

a) Show that two 2×2 matrix can be multiplied using 4 additions and 8 multiplication

Consider 2×2 matrices as X and Y

$$XY = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}$$

$$XY = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + Y_{22}Y_{22} \end{bmatrix}$$

$\Rightarrow$ therefore the multiplication of two matrices has been performed by using 4 additions and 8 multiplications.

Computing X^n requires $\log n$ matrix multiplications. Since each matrix multiplication has at most 4 additions and 8 multiplications this becomes $4(\log n) + 8(\log n) = 12(\log n) = O(\log n)$, logarithmic growth.

b) Show that $O(\log n)$ Matrix Multiplications suffice for Computing X^n.

$\Rightarrow$ n can be written as power of 2. i.e

$n = 2^{k1} + 2^{k2} + \dots + 2^{km}$

So X^n can be expressed as:

$X = X^{(2^{k1})} \cdot X^{2^{k2}} * \dots \times X^{2^{km}}$

Therefore X^n can be determined by looking at each bit in binary representation of n, and multiplying all $X^{2^{ki}}$ together. Where k_i is the bit position of the i^{th} standing bit.