1. (a) By definition,

$$\Theta(g(n)) = \{ f(n) \mid (0 \le c_2 \cdot g(n) \le f(n) \le c_1 \cdot g(n)) \\ \forall n \ge n_0, \exists (c_1 > 0, c_2 > 0, n_0 > 0) \}$$

To prove $f(n) \in \Theta(g(n))$, we need to find c_1, c_2 and n_0 such that $f(n) \le c_1 \cdot g(n)$ and $f(n) \ge c_2 \cdot g(n)$ for all $n > n_0$.

$$f(n) = 64n^2 \quad \text{and} \quad g(n) = n^4$$

$$c_1 \cdot n^4 \le 64n^2 \le c_2 \cdot n^4$$

If $c_1 = 1$ and $c_2 = 64$, then $n^4 \le 64n^2 \le 64 \cdot n^4$

When $n > 0$ (assumption), simplified inequality is —

$$n^2 \le 64 \le 64 \cdot n^2$$

This holds true for all $n \ge 1$. For, $n^2 \cdot c_1 \le 64$, for any positive value of n, we can choose c_1 such as 1/64.

For $64 \le c_2 \cdot n^2$, we can choose c_1 such as 100.

Therefore, we can conclude that $f(n) \in \Theta g(n)$

1. (b) By definition,

$$\Omega(g(n)) = \{f(n) \mid 0 \leq c.g(n) \leq f(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}.$$

To prove $f(n) \in \Omega(g(n))$, we need c and n_0 where

$$f(n) \geq c.g(n) \text{ for all } n > n_0.$$

Given, $f(n) = \frac{1}{3}n^2 + 10n - 2$ and $g(n) = n^2$

Prove : $\frac{1}{3}n^2 + 10n - 2 \geq c.n^2$

For $c = 5$ and $n_0 = 10$

$$\frac{1}{3}n^2 + 10n - 2 \geq 5n^2$$

The inequality holds. $f(n)$ is lower bounded by $g(n)$.

1. (c) To prove $f(n) \in O(g(n))$, we need n_0 and c such that

$$f(n) \leq c.g(n) \; \forall \; n \geq n_0, \; c > 0, \; n_0 > 0.$$

Given, $f(n) = 3,00,000n^3 + 1$ and $g(n) = n^4$

$$f(n) = 3 \times 10^5 n^3 + 1.$$

For $c = 5$ and $n = 10^5$.

$$f(n) = 3 \times 10^5 \times 10^{15} + 1 = 3 \times 10^{20} + 1.$$

and $g(n) = 10^{20}$.

The inequality $f(n) \leq c.g(n)$ holds true.

For, $n_0 = 1$, c has to greater than or equal to 3×10^5.

Proof :

$$3 \times 10^5 n^3 + 1 \leq c.n^4.$$

(Dividing by n^3) $\Rightarrow$ $3 \times 10^5 + \frac{1}{n^3} \leq c.n.$

When n approaches infinity $(1/n^3)$ approaches 0. Thus,

$c \geq 3 \times 10^5$.

$\therefore f(n) \in O(n^4)$.

1. (d) Given, $f(n) = 15n^{15}$ and $g(n) = n^{15}$

By definition, to prove $f(n) \in o(g(n))$, we need to prove that $o(g(n)) \Rightarrow f(n) < c \cdot g(n)$ or determine whether the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ is equal to ~~tends to~~ 0, i.e., $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{15n^{15}}{n^{15}} = 15 \neq 0.$$

Since, limit is 15 and not zero, we cannot conclude that $f(n) \in o(g(n))$.

1. (e) Given $f(n) = n^{20} - 17$ and $g(n) = n^{16}$.

By definition, $f(n) > c \cdot g(n)$. $(\forall\, n \geq n_0\ \forall\, c > 0\ \exists\, n_0 > 0)$ or prove that $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

$\lim_{n\to\infty} \frac{n^{16}}{n^{20}-17}$ will be approximately equal to $\frac{1}{n^4}$. ~~and not zero~~.

As n approches ∞, $(1/n^4) = 0$.

Therefore, $f(n) \in \omega(g(n))$. as $g(n)$ grows much slower than $f(n)$.

2. (a). The time complexity of function1(n) is O(n). The outer loop runs from 0 to n, and the inner loop runs from 0 to (i) for each i in 0 to n. Since, after the first iteration of the inner loop, the break statement will exit the inner loop and execute the outer loop for next 'i'. Therefore, the number of times "*" is printed is the same as 'n'. Since, the inner loop executes once at most for each iteration of the outer loop, the time complexity of the function is O(n).

2. (b) when n = 4.

(iterations = 2.)

① $1^2 <= 4$ ✓ → $i = 2$.

② $2^2 <= 4$ ✓ → $i = 3$

$3^2 <= 4$ X.

When n = 10

(iterations = 3)

① $i^2 <= 10$ ✓ → $i = 2$

② $2^2 <= 10$ ✓ → $i = 3$

③ $3^2 <= 10$ ✓ → $i = 4$

$4^2 <= 10$ X

The iterations will continue until i^2 exceeds n, in other words, until i becomes greater than $\sqrt{n}$. Therefore, the time complexity of the functions is $O(\sqrt{n})$.

2. (c) The algorithm repeatedly subtracts the smaller number from the larger number until the two of the number becomes equal.

Thus, the time complexity of the given algorithm is the greatest common divisor of 'm' and 'n'.

2. (d) In the inner loop, due to the integer division, the result of which will be limited to the number of times `j' is halved until $j=0$. Therefore, the time complexity of the inner loop is $O(\log_2 n)$. The outer loop executes as long as i is less than n, also, i starts with 1 and is doubled with each iteration. Thus, for the outer loop as well, the time complexity is $O(\log_2 n)$. As the algorithm contains nested loops, the time complexity for the given function is $[O(\log_2 n)^2]$.

2. (e) The function consists of 3 nested loops. The time complexity can be expressed as:

$$O((n//2), (n//2+1) * \log_2 n)$$

where ① represents the time complexity for the outermost 'for' loop; ② runs for $(n//2+1)$ times. Both these loops will run for $(n/2)$ times. For example, when 'n=4', i ranges from 0 to 2 and j ranges from 1 to 3. The innermost loop compares m and n and doubles m with each iteration - that satisfies the condition `m <= n'; thus, for ③ the time complexity is $\log_2 n$. Therefore, the time complexity for the function is approximately.

$O(n^2 . \log_2 n)/4 \sim O(n^2)$.