Q2)

a) Given $f(n) = \frac{1}{2}n^2 + 900n - 12$ $\quad g(n) = n^2$

$f(n) \in \Omega(g(n))$.

We know the defination for Big Omega.

$\Omega(g(n)) = \{ f(n) \mid 0 \le c.g(n) \le f(n), \forall n \geq n_0,$

$\exists c > 0, \exists n_0 > 0 \}$.

Consider,

$$c = \frac{1}{4}$$

$$\frac{1}{4}n^2 \le \frac{1}{2}n^2 + 900n - 12$$

$$\frac{1}{4}n^2 + 900n - 12 \geq 0$$

roots for $\frac{1}{2}n^2 + 900n - 12 = 0$ are

$$n = \frac{-900 \pm \sqrt{900^2 - 4(1/4)(-12)}}{2(1/4)}$$

$n = 0.0134$ (+ve root)

$\therefore$ For $c = 1/4$, $n_0 = 0.01334$

$\forall n \geq n_0$

$$\frac{1}{4}n^2 \leq \frac{1}{2}n^2 + 900n - 12 \quad \forall n \geq 0.0134$$

$\therefore$ $\boxed{f(n) \in \Omega(g(n))}$ Proved

b) Given:-

$f(n) = 12n^{15} + 1000 \qquad g(n) = n^{20}$

$f(n) \in O(g(n)) \rightarrow$ Need to prove.

We know, Big Oh Notation defination given by.

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

Let $c = 13$,

$$12n^{15} + 1000 \leq 13n^{20}$$

$$13n^{20} - 12n^{15} - 1000 \geq 0$$

$$n^{15}(13n^5 - 1012) \geq 0$$

$$13n^5 - 1012 \geq 0$$

$$n^5 \geq \frac{1012}{13}$$

$$n^5 \geq 77.846$$

$$n \geq 2.39.$$

$\therefore$ For $c = 13$ $n_0 = 2.39$

$$12n^{15} + 1000 \leq 13n^{50} \quad \forall\ n \geq 2.39$$

$\therefore$ $f(n) \in O(g(n))$. <u>Proved</u>

c.) Given:-

$f(n) = 3n^3 + n$ $g(n) = n^3$

To prove, $f(n) \in o(g(n))$.

Consider defination of Little-Oh notation.

$o(g(n)) = \{ f(n) \mid 0 \leq f(n) < c \cdot g(n), \forall n \geqslant n_0,$
$\forall c > 0, \exists n_0 > 0 \}$.

Alternative definition:-

$\{ f(n) \mid \lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0 \}$

Proof

$\lim\limits_{n \to \infty} \frac{3n^3 + n}{n^3}$

$\div$ by n^3 on both num & den

$= \lim\limits_{n \to \infty} \frac{3 + 1/n^2}{1}$

$= 3 + 0$
$= 3 \neq 0$

$\therefore \boxed{f(n) \notin o(g(n))}$ Disproved

d) Given $f(n) = 8n^4 - 90$ $g(n) = n$

To prove :- $f(n) \in \omega(g(n))$

Consider the defination of Little omega notation.

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

Alternative defination:-

$$\left\{ f(n) \mid \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0 \right\}$$

$$= \lim_{n \to \infty} \frac{n}{8n^4 - 90}$$

$\div$ by n^4 on both num & den

$$= \lim_{n \to \infty} \frac{1/n^3}{8 - 90/n^4}$$

$$= \frac{0}{8 - 0} = 0$$

$\therefore \boxed{f(n) \in \omega(g(n))}$ Proved

e) Given:

$f(n) = 16n^2 \qquad g(n) = n^3$

~~To prove~~ $f(n) \in \Theta(g(n))$

Consider the defination of Big theta notation

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

Consider,

$c_1 = 4 \quad c_2 = 2$

$\cancel{17n^3 \le 16n^2 \le 15n^3}$

$$\underbrace{2n^3 \le 16n^2}_{n \le 8} \le \underbrace{\le 4n^3}_{n \geq 4}$$

$\therefore n \in (4, 8)$

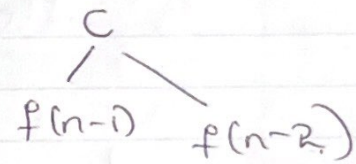
$\therefore \boxed{f(n) \notin \Theta(g(n))}$ Disproved

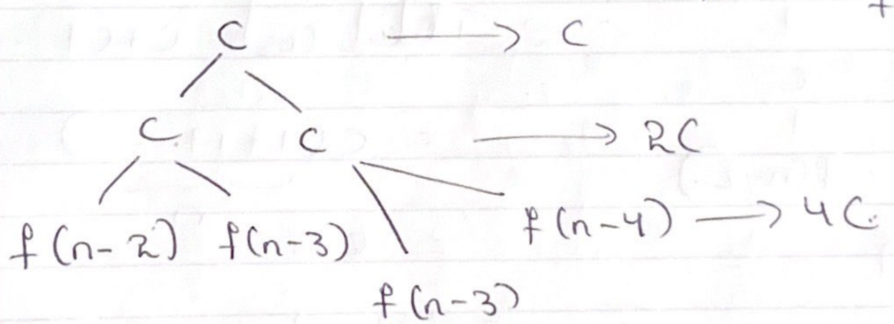
Q3)

1) Given the function, we can write recursive equation as:-

$$f(n) = f(n-1) + f(n-2) + c \qquad n > 1$$

$$f(1) = c$$





$\therefore$ Total cost $= c + 2c + 4c + \cdots$

$$= c(\underbrace{1 + 2 + 4 + \cdots}_{n})$$

which is order of $\underline{\underline{O(2^n)}}$

Time Complexity

2) Given the recursive function:- we can write equation as

$$f(n) = f(n-1) + c \qquad n \geq 1$$

$\therefore$

$$\begin{array}{c} c \\ | \\ f(n-1) \end{array}$$

$$\begin{array}{c} c \\ | \\ c \\ | \\ f(n-2) \end{array}$$

$\therefore$ Total cost $= c + c + \cdots +$

$= c(\underbrace{1+1+\cdots}_{n})$

$= cn$

$\therefore$ Time Complexity $= \underline{\underline{O(n)}}$

3) Given the function:-

```
def time_complexity_3(n):
    k=0
    i=n
    while(i>0):
        for j in range(0,i):
            k+=1
        i = i//2
    print(k).
```

outer while runs $\log_2 n$ time, because of i//2.

Inner for loop runs for:

$$n + \frac{n}{2} + \frac{n}{4} + \ldots$$

the above geometric progression is $O(n)$

Since while loop runs for $\log n$ times.

Total time complexity = $\underline{O(n\log n)}$

4) Given the function

```
def time_complexity_4(n):
    a=1
    b=1
    c=1
    while (c<=n):
        a +=1
        b=a*a
        c +=b
    return.
```

For $n = 5$

$a = 2$ $\quad$ $a = 3$

$b = 2^2$ $\quad$ $b = 3^2$

$c = 2^2$ $\quad$ $c = 2^2 + 3^2$

$\therefore$ For $n = 5$

$c = 2^2 + 3^2 + \ldots + \leq 5$,

loop runs until

c = sum of perfect squares $\geq n$

$$2^2 + 3^2 + 4^2 + \dots + = \frac{1}{6} m(m+1)(2m+1)$$

$$\geq n$$

$$\approx \quad m^3 \geq n$$

$$m \geq \sqrt[3]{n}$$

$\therefore$ Time Complexity = $\underline{\underline{O(\sqrt[3]{n})}}$

5) Given the function :-

```
def time_complexity_5(n):
    i = 1
    while(i<n):
        i += 1
        print(pow(i,n))
```

we know that pow(i, n) has time complexity of $O(n)$, while the loop runs from 1 to n.

$\therefore$ Time Complexity = $O(n^2)$