(Algorithm)

SYLLABUS:-

1. Analysing Algo
2. Divide and Conqueror
3. Greedy Technique
4. Dynamic Programming
5. Hashing, traversal of Graph.

Algorithm is a combination of ~~finite~~ sequence of finite steps to solve a problem.

Properties of Algorithm:-

(i) It should terminate even it has finite no. of steps, after finite time.

eg:
```
1: while (1)
2: print("A");
```

finite steps, but infinite loop is there, so no termination point.

(ii) It should produce atleast 1 - output.

(iii) It can takes 0 or more inputs.

eg:- function can take, zero or more no. of arguments.

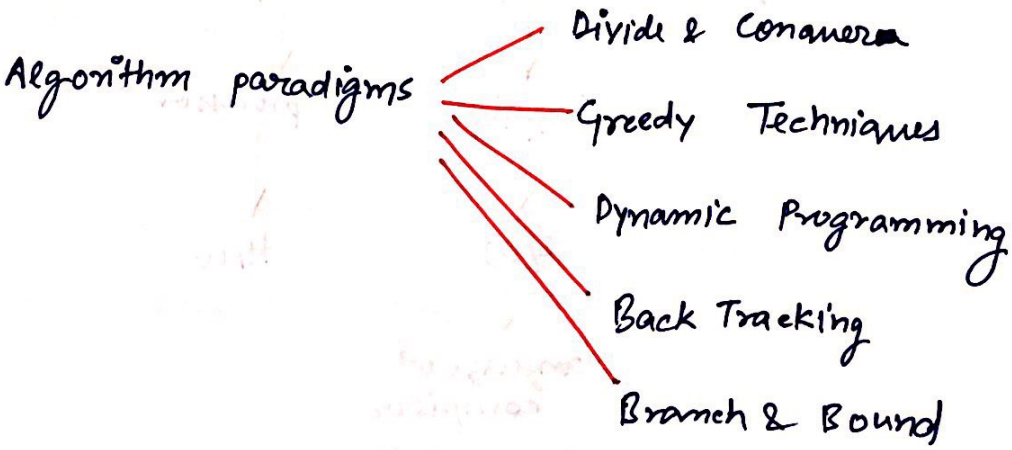
(iv) It should be deterministic .e i.e for a given particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states

(v) It should be effective.

Steps required to design algorithm:-

(i) Understand and analyze the Problem (Problem defination). Findout all possible input and there corresponding outputs.

(ii) Design Algorithm i.e choose the appropriate algorithm pardigm to map input to there corresponding output.



(iii) Draw Flowchart

(iv) Verification

(V) Implementation (Coding) i.e writing executable code.

(VI) Analysis (Time & space complexity are measured).

Algorithm Analysis :-

If problem having more than one algorithm, then best one is decided by analysis based on two factors:

(i) Time complexity (CPU time)

(ii) space complexity (MM space)

Time complexity :-

$$T(A) = \underbrace{C(A)}_{\text{compile time}} + \underbrace{R(A)}_{\text{Run-time}}$$

C(A) → compiler → S/W → language of compiler in which it is written

R(A) → processor → H/W

Since compilation is done only once, so it is ignored

i.e $T(A) = R(A)$

Types of Analysis

Apostiary (Relative) Analysis

(i) It is independent on compiler language & H/W time

(ii) Exact answer

(iii) Different answers on different computer configuration

(iv) Focus on S/W & H/W of computer both collectively.

Apriori (Absolute) Analysis

(i) It is Independent on compiler lang & H/W time.

(ii) Approximate answer

(iii) Same answers on different computer config.

(iv) Only Focused on software Logic.

NOTE:- So Apriori Analysis is performed as it focused on algorithm logic.

Apriori Analysis:-

It is a determination of order of magnitude of a statement i.e no. of time a statement will execute.

Eg.1:

```
main()
{
   S1 x = y+2;        ——  1-statement (executed once)
}
```

so, $O(1)$

Eg-2:

```
main()
{
  S1 x = y+2;                 ——  1-statement (executed once)
   for(i=1; i<=n; i++)
   {
     S2 x = y+2;              ——  2-statement (executed n-times)
   }
}
```

$O(n+1) \cong O(n)$

Eg-3:

```
main()
{
  S1 x = y+z;                 ——————  1-time      1
   for(i=1; i<=n; i++)
   S2 x = y+z;                ——————  n-time      n

   for(i=1; i<=n; i++)
   { for(j=1; j<=n; j++)
     S3 x = y+z;              ——————  n×n time    n^2
   }
}
```

$T(n) = n^2 + n + 1$

So, $\boxed{T(n) = O(n^2)}$

Eg-4:

```
main()
{   i=1
  while (i≤n)
   {
     i = i + K
   }
}
```

$T(n) = \frac{n}{K}$

$\boxed{T(n) = O(n)}$

Eg-5:

```
main()
{   i=n
    while (i≥n)
    {
       i = i-K;
    }
}
```

$T(n) = n/K$

$\boxed{T(n) = O(n)}$

OR

```
main()
{   i=n
    while (i≥n)
    {
        i = i-K;
        i = i-l;
    }
}
```

$i = i-(K+l)$;

$T(n) = \frac{n}{K+l}$

$\boxed{T(n) = O(n)}$

Eg-6:

```
main()
{
    i=1;
    while(i<=n)
    {
        i=i*2;
    }
}
```

$\log_2 n + 1$

$T(n) = O(\log_2 n)$

OR

```
main()
{
    i=1;
    while(i<=n)
    {
        i=i*k;
    }
}
```

$\log_k n + 1$

$T(n) = O(\log_k n)$

OR

```
main()
{
    i=1
    while(i<=n)
    {
        i=i*2;
        i=i*3;
    }
}
```

$i = i * 6;$

$T(n) = O(\log_6 n)$

Eg-7:

```
main()
{
    i = n
    while ( i > 1)
    {
        i = i/k ;
    }
}
```

$\log_k n$

$T(n) = O(\log_k n)$

OR

```
main()
{
    i = n
    while ( i > 1)
    {
        i = i/2 ;
        i = i/5 ;
    }
}
```

$i = i/10$;

$T(n) = O(\log_{10} n)$

Eg-8:

```
main()
{
    i = 2 ;
    while ( i < n)
    {
        i = i^2
    }
}
```

$i = 2 \rightarrow 2$

$(2)^2 \rightarrow 2^{2^1}$

$(2^2)^2 \rightarrow 2^{2^2}$

$((2^2)^2)^2 \rightarrow 2^{2^3}$

$\vdots \qquad \vdots$

$2^{2^k} = n$

$2^k = \log_2 n$

$k = \log(\log_2 n)$

$T(n) = \log_2(\log_2 n)$

OR

```
main()
{
    i=12;
    while(i<n)
    {
        i= i^27;
    }
}
```

12^{27}

12^{27^2}

12^{27^3}

$\vdots$

$12^{27^K} = n$

$27^K = \log_{12} n$

$K = \log_{27}(\log_{12} n)$

$T(n) = O(\log_{27}(\log_{12} n))$

Eg-9:

```
main()
{
    i=n;
    while(i>2)
    {
        i= i^1/5
    }
}
```

$i = n$

$n^{1/5}$

$(n^{1/5})^{1/5} = n^{(1/5)^2}$

$((n^{1/5})^{1/5})^{1/5} = n^{(1/5)^3}$

$\vdots$

$n^{(1/5)^K} = 2$

$(1/5)^K \log_2 n = 1$

$\log_2 n = 5^K$

$K = \log_5(\log_2 n)$

$T(n) = O(\log_5(\log_2 n))$

Eg.10:

```
main()
{
    q=1;
    for(i=1; i<=n; i+=2)          — n/2
    {
        p=1;
        for(j=1; j<=n; j=j*2)     — log2 n
            p++;                  — log2 n
        for(k=1; k<=p; k=k*2)     — log2 P — log2(log2 n)
            q++;
    }
}
```

$$T_n = n/2 \left(\log_2 n + \log_2(\log_2 n) \right)$$

$$T_n = n/2 \log_2 n \rightarrow T_n = O(n \log n)$$

q-value for 1-iteration $= \log_2 P = \log_2(\log_2 n)$

total iteration is $n/2$ so, q-value final:

$$= n/2 \log_2(\log_2 n)$$

$$= O(n \log_2(\log_2 n))$$

Eg-11:

```
main()
{
    x=1
    for( i=1 ; i≤n; i*=5)  ——  log5 n
    {
        j=2;
        while( j≤n) ——  log20 (log2 n)
        {
            j=j^20
            x=x+n; ——
        }
    }
}
```

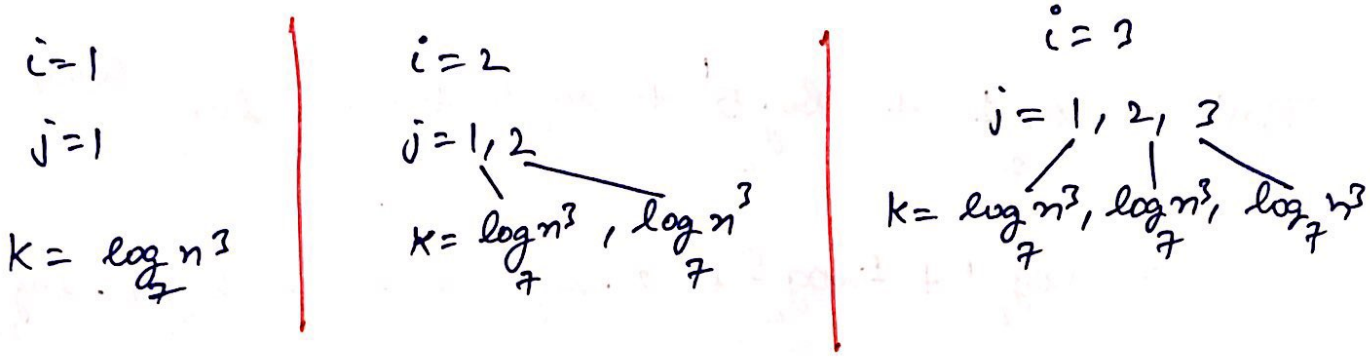
$$T_n = O\left(\log_5 n * \log_{20}(\log_2 n)\right)$$

$$x = O\left(\log_5 n * n \log_{20}(\log_2 n)\right)$$

Eg-12:

```
main()
{
  for(i=1; i≤n; i++) ——  n
  {
    for(j=1; j≤i; j++) ——  i=n
    {
      for( k=1; k≤n^3; k=k*7) ——  log7 n^3
      {
        x=y+z;
      }
    }
  }
}
```

i and k are independent, but j is dependent on i, so we can't perform multiplication of there individual time complexity.



so, $T(n) = 1 \log_7 n^3 + 2 \cdot \log_7 n^3 + 3 \cdot \log_7 n^3 + \cdots\cdots n \cdot \log_7 n^3$

$= \frac{n(n+1)}{2} \log \frac{n^3}{7}$

$T(n) = O(n^2 \log_7 n^3)$

Eg-13:

```
main()
{
   for(i=1; i<=n; i*=5)  —— log_5 n
   {  j=1;
      while(j<=i)  —— log_2 i (dependency)
      {
         j=j*2;
      }
   }
}
```

$i = 1$	$i = 5$	$i = 5^2$	.....	$i = 5^{\log_5 n}$
$j = \log_2 1$	$j = \log_2 5$	$j = \log_2 5^2$	.....	$j = \log_2 5^{\log_5 n}$

$$T(n) = \log_2 1 + \log_2 5^1 + \log_2 5^2 + \cdots + \log_2 5^{\log_5 n}$$

$$= \log_2 1 + 1 \cdot \log_2 5 + 2 \cdot \log_2 5 + \cdots + \log_5 n \cdot \log_2 5$$

$$= \log_2 1 + \log_2 5 \left(1 + 2 + 3 + \cdots + \log_5 n \right)$$

$$= \log_2 1 + \log_2 5 \left(\frac{\log_5 n (\log_5 n + 1)}{2} \right)$$

$\log_2 1$ — constant

$\frac{\log_5 n (\log_5 n + 1)}{2}$ — $\frac{n(n+1)}{2}$

$$T(n) = O\left(\log_5 n\right)^2$$

Eg-14:

```
main()
{
  for(i=1; i≤n²; i+=4)                    — n²/4
  {
    for(j=7; j≤n⁹; j=j¹⁷)                 — log₁₇(log₇ n⁹)
    {
      for(k=n; k≥9; k=k^1/16)             — log₁₆(log₉ n)
      {
        x=y+z;
      }
    }
  }
}
```

(no-dependency) multiply each

$$T(n) = \frac{n^2}{4} * \log_{17}(\log_7 n^9) * \log_{16}(\log_9 n)$$

$$T(n) = \frac{n^2}{4} * \log_{17}(9 \log_7 n) * \log_{16}(\log_9 n)$$

$$T(n) = \frac{n^2}{4} * \left(\log_{17} 9 + \log_{17}(\log_7 n)\right) * \log_{16}(\log_9 n)$$

$\frac{n^2}{4}$ constant, $\log_{17} 9$ constant

$$T(n) = n^2 \cdot \log_{17}(\log_7 n) \cdot \log_{16}(\log_9 n)$$

NOTE :-

```
{
    i = i + 2;
    i = i * 4; ( This has high precidence)
}
```

```
{   i = i * 4;
    i = i^5;  ( high precidence)
}
```

Precedence order :-

$$\begin{bmatrix} i = i + K \\ \text{or} \\ i = i - K \end{bmatrix} < \begin{bmatrix} i = i * K \\ i = i / K \end{bmatrix} < \begin{bmatrix} i = i^K \\ i = i^{1/K} \end{bmatrix}$$

Eg-15:-

```
main()
{
    for( i=1; i≤n; i++)  ——— n
    {
        j=1
        while( j≤n)
        {
            j=j+i;  ——— n/i (dependency)
        }
    }
}
```

i=1 i=2 i=3 ------ i=n

j → n/1 j → n/2 j → n/3 j → n/n

$T(n) = n/1 + n/2 + n/3 + \cdots + n/n$

$= n\left(1 + 1/2 + 1/3 + \cdots + 1/n\right)$

$T(n) = n \log n$

Eg-16:

```
Main()
{
    int i;
    for( i=1; i≤n; i++)  ——— n*1
    {
        for( i=1; i≤n^2; i++)  ——— n^2*1
        {
            for( i=1; i≤n^3; i+=4)  ——— n^3/4
            {
                x=y+z;
            }
        }
    }
}
```

L3 L2 L1

Since after executing L1, $i = n^3/4$ thus checking condition for L2 and L3 will be false -

so, L1 $\Rightarrow$ $n^3/4$

L2 $\rightarrow$ 1

L3 $\rightarrow$ 1

so, $T(n) = n^3/4 = O(n^3)$

Eg-17:

```
main()
{
  for(int i=1; i^2 <= n ; i++)  -> n^1/2
  {
    for(int j=1; i^25 <= n^100; j++)  - n^4
    {
      for(int k=1; k^75 <= n^30 ; k++)  - n^2/5
      {
        x = y+z;
      }
    }
  }
}
```

(no-dependency)

$$T(n) = n^{1/2+4+2/5} = n^{49/10}$$

$$T(n) = O(n^{49/10})$$

Asymptotic Notation :-

(i) Big-Oh- Notation (O)

(ii) Omega- Notation (Ω)

(iii) Theta- Notation (Θ)

Let $f(n)$ and $g(n)$ be two +ve functions,

(i) **Big-Oh- Notation (O)** :- $f(n) = O(g(n))$ if

$f(n) \leq c \cdot g(n)$, $\forall\ n$, $n \geq n_0$

such that some constant $c > 0$ and $n_0 \geq 1$ existed.

eg:-

$f(n) = n^2 + n + 5$

$g(n) = n^2$

$f(n) = O(n^2) = O(g(n))$

$f(n) \leq c\ g(n)$

$n^2 + n + 5 \leq c\ (n^2)$

Let $c = 3$

then $n^2 + n + 5 \leq 3n^2$

for all $n \geq n_0$

where $n_0 = 2$

$n^2 + n + 5 = O(n^2)$

eg- $f(n) = n+10$, $g(n) = n-10$

$f(n) = O(g(n))$

$f(n) \leq c \cdot g(n) \Rightarrow n+10 \leq c(n-10)$

Let $c = 2$

$n+10 \leq 2(n-10)$ $\forall$ n , $n \geq n_0$

$n_0 = 30$

so, $n+10 = O(n-10)$

eg- $f(n) = n-10$ and $g(n) = n+10$

$f(n) \leq O(g(n))$

$n-10 \leq c(n+10)$, $c = 1$

so, $n-10 = O(n+10)$ $\forall$ $n \geq n_0$, $n_0 = 1$ and $c = 1$

eg- $f(n) = n^2$, $g(n) = n$

$f(n) \leq c\, g(n)$

$n^2 \leq c \cdot n$ no c and n_0 possible

so, $n^2 \neq O(n)$

(ii) **Omega-notation (Ω) :-** $f(n) = \Omega(g(n))$ if

$f(n) \geq c \cdot g(n)$, $\forall n$, $n > n_0$,

such that c and n_0 are constants

and $c > 0$ and $n_0 \geq 1$

eg:- $f(n) = n - 10$, $g(n) = n + 10$

$f(n) \geq c \cdot g(n)$

$n - 10 \geq c(n+10)$ as $c > 0$ so, $c = 1/2$ possible

$c = 1/2$

so, $n - 10 \geq 1/2 (n+10)$ when $n \geq 30$

so, $n_0 = 30$

$n - 10 = \Omega(n+10)$ $c = 1/2$, $n_0 = 30$

eg:- $f(n) = n^2 + n + 5$, $g(n) = n^2$

$f(n) \geq c \cdot g(n)$

$n^2 + n + 5 \geq c(n^2)$

$c = 1$

$n^2 + n + 5 = \Omega(n^2)$

eg:- $f(n) = n$, $g(n) = n^2$

$f(n) \geq c \cdot g(n)$

$n \geq c \cdot g(n^2)$

if $c = 1/n$ then it can be possible but c should be a constant not a func^n.

so, $n \neq \Omega(n^2)$

(iii) Theta-notation (Θ) :- $f(n) = \Theta(g(n))$ if

- $f(n) \leq c_1 \cdot g(n)$ and
  $f(n) \geq c_2 \cdot g(n)$ $\forall n$, $n > n_0$

such that c_1, c_2 are constant and $n_0 \geq 1$

eg:- $f(n) = n-10$, $g(n) = n+10$

$f(n) \leq c_1 \cdot g(n)$

$n-10 \leq c_1 (n+10)$

$c_1 = 1$

$f(n) \geq c_2 \cdot g(n)$

$n-10 \geq c_2 \cdot (n+10)$

$c_2 = 1/2$

$n-10 = O(n+10)$ possible

$\forall n$; $n \geq 30$, $n_0 = 30$

$n-10 = \Omega(n+10)$ possible

so,

$n-10 = \Theta(n+10)$

eg:- $f(n) = n$, $g(n) = n$.

then $n \leq c_1(n)$ & $n \geq c_2(n)$

$c_1 = 2$ $\qquad$ $c_2 = 1/2$

$\forall n;\ n \geq 1$

$n_0 = 1$

$n = \Theta(n)$ mathematically equal

eg:- $f(n) = n^2+n+5$, $g(n) = n^2$

$n^2+n+5 \leq c_1(n^2)$ $\qquad$ $n^2+n+5 \geq c_2(n^2)$

$c_1 = 3$ $\qquad$ $c_2 = 1$

$\forall n;\ n \geq 2$

$n_0 = 2$

So, $n^2+n+5 = \Theta(n^2)$ Asymptotically equal

eg:- $f(n) = n$; & $g(n) = n^2$

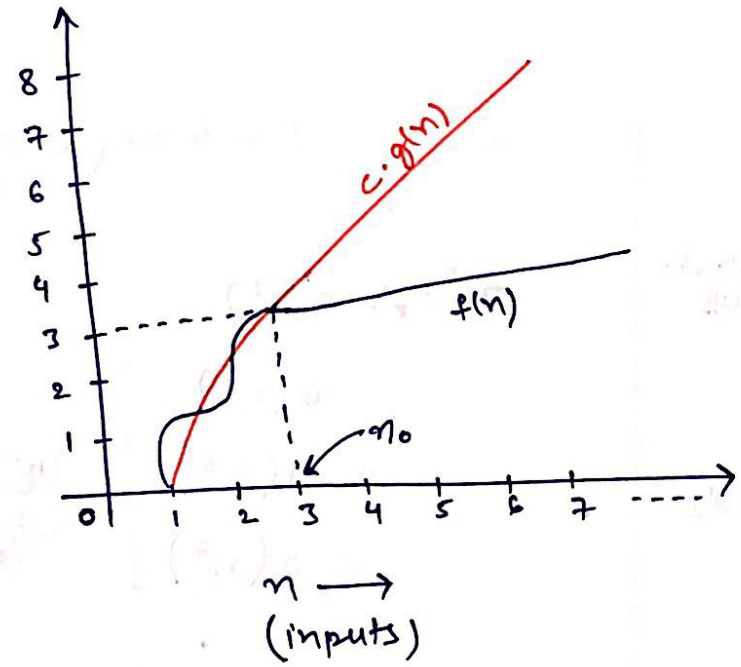
$n \leq c_1 n^2$ ✓ $n \geq c_2 n^2$

$n = O(n^2)$ $n \neq \Omega(n^2)$

possible not-possible

so, $n \neq \Theta(n^2)$

Big-Oh-notation ($O(n)$) :-



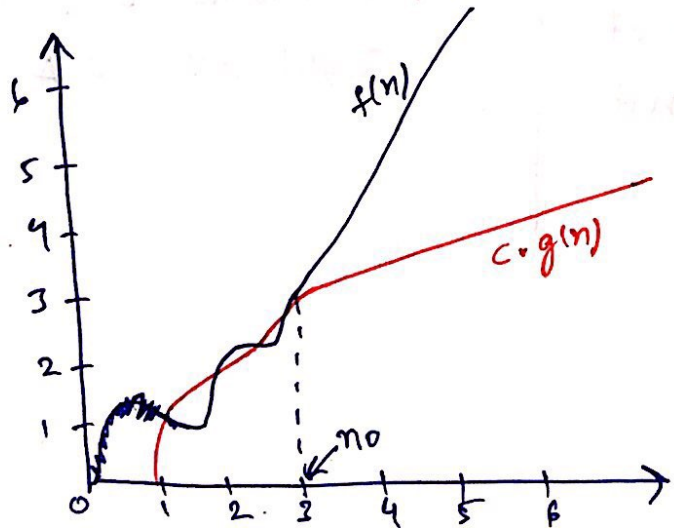
$f(n) = O(g(n))$

$f(n) \leq c \cdot g(n) \quad \forall n$

$n \geq n_0$

where $n_0 = 3$

Omega-notation ($\Omega(n)$) :-



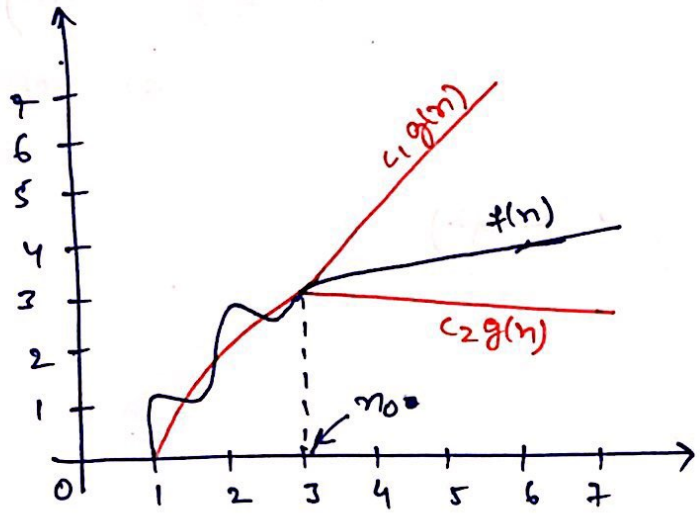
$f(n) = \Omega(g(n))$

$f(n) \geq c \cdot g(n) \quad \forall n$

$n \geq n_0$

where $n_0 = 3$

Theta-notation ($\theta(n)$) :-



$f(n) = \theta(g(n))$

$c_2 g(n) \leq f(n) \leq c_1 g(n)$

$\forall\ n;$

$n \geq n_0$

$n_0 = 3$

Big-Oh-Notation ($\leq$, O)

$n^2 = O(n^2)$ — Tightest UB

$= O(n^3)$

$= O(n^5)$

$= O(n^7)$

Upper bounds

$A = O(B)$

this mean B is UB of A (Tight UB or not Tight UB)

Small-oh-notation ($<$, o)

$n^2 \neq o(n^2)$

$= o(n^3)$

$= o(n^5)$

$= o(n^7)$

UB not tight

$A = o(B)$

this means B is not Tight UB of A

Omega notation ($\geq, \Omega$)

$n^5 = \Omega(n)$
$= \Omega(n^2)$
$= \Omega(n^3)$
$= \Omega(n^4)$
$= \Omega(n^5)$ — Tightest LB
$\neq \Omega(n^6)$

LB's

$A = \Omega(B)$
↓
B is LB of A

Small-omega ($>, \omega$)

$n^5 = \Omega(n)$
$= \Omega(n^3)$
$= \Omega(n^4)$

LB

$\neq \Omega(n^5)$ not tightest LB

$A = \omega(B)$
↓
B is not tight LB of A

Theta - Notation ($\Theta, \geq, \leq$)

$n^2 = O(n^2)$ — TUB
$= \Omega(n^2)$ — TLB

$A = \Theta(B)$
↓
B is TUB, TLB of A