Problem

A: $n^2 \rightarrow$ Best

B: n^3

Same $\longleftarrow$ $A = \theta(B)$

B is worst $\longleftarrow$ $A = O(B)$

B is Best $\longleftarrow$ $A = \Omega(B)$

	A	B
Time →	n → Best	n^2
Space →	n^5	n

Preference is given to the time.

1) Big oh O (behaves like $\leq$ $\longleftarrow$ order
2) Big Omega Ω $\geq$
3) Theta θ $=$
4) Small oh o $<$
5) Small omega (ω) $>$

Big oh notation (O)

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

$\in$ → belongs to

$$\boxed{f(n) \le c \cdot g(n) \text{ for some } c>0 \text{ after } \overline{n \ge n_0} \ge 0}$$

constant

$c = ?$

$n_0 =$

eg. $f(n) = n$, $g(n) = 5(n)$

$$\boxed{f(n) = O(g(n))}$$

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

$c = 1$

$n \le c \cdot (5n)$

$5n \ge n$ — for any value $n \ge 0$ → n_0

2) $f(n) = 2n + 10$, $g(n) = n$

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

$f(n) \leq c \cdot g(n)$ $\quad c > 0$, $\underline{n \geq n_0 \geq 0}$

$2n + 10 \leq c \cdot (n)$

$2n + 10 \leq \underbrace{3}_{c}n$

$n \geq \underbrace{10}_{n_0}$

* $\underline{\Omega \text{ (Big Omega)}}$

$\boxed{f(n) = \Omega(g(n))}$ iff $\boxed{f(n) \geq c \cdot g(n)}$

for some $c > 0$

after - $n \geq n_0 \geq 0$

eg. $f(n) = 5n + 10$, $g(n) = 2n$

$$\boxed{\begin{array}{l} f(n) = \Omega(g(n)) \\ f(n) \geq c \cdot (g(n)) \end{array}}$$

$c = \underline{\underline{1}}$

$5n + 10 \geq c \cdot 2n$

$5n + 10 \geq 2n$

$3n \geq 10$

$n \geq \frac{10}{3}$, $n \geq \underbrace{3.3}_{n_0}$

eg. $f(n) = n^2$, $g(n) = n^2 + n$

$f(n) \geq c \cdot g(n)$ $c > 0$ $\underset{(0,1)}{\wedge}$ b/w.

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

$n^2 \geq \frac{1}{2}(n^2 + n)$

$\frac{1}{\cancel{2}} \cancel{n^2} + \frac{1}{\cancel{2}} n^2 \geq \cancel{\frac{1}{2} n^2} + \frac{1}{\cancel{2}} \cdot n$

$$\boxed{\begin{array}{c} f(n) = \Omega(g(n)) \\ c = 1/2 \ , \ n \geq 0 \end{array}}$$

Θ notation

$$f(n) = \Theta(g(n)) \text{ iff}$$

$$\boxed{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$$

for, some $c > 0$ after $n \geq n_0$

Cond^n 1 - $f(n) \geq c_1 \cdot g(n)$ — $f(n) = \Omega(g(n))$

Cond^n 2 - $f(n) \leq c_2 \cdot g(n)$ — $f(n) = O(g(n))$

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

$$\boxed{f(n) = \Theta(g(n))}$$

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

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

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

$5n^2 \geq n^2 + n$

$\boxed{4n^2 \geq n}$ — $n \geq 0$ — n_0 ⟶ always true

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

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

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

$n^2 + n \geq \frac{1}{\cancel{5}}(\cancel{5}n^2)$

$n^2 + n \geq n^2$

$n \geq 0$ — n_0

$c < 1$

$\Downarrow$

$c = \frac{1}{5}$

1)

```
A()
{
 int i, j, k, n;
 for(i=1; i<=n; i++)
 {
    for(j=1; j<=i; j++)
    {
       for(k=1; k<=100; k++)
       {
          printf("Mohan");
       }
    }
 }
}
```

$i=1$ $j=1$ time $k=100$ time	$i=2$ $j=2$ times $k=2\times100$ times	$i=3$ $j=3$ times $k=3*100$ $=300$	$i=4$ $j=4$ $k=4*100$ $=400$	...	$i=n$ $j=n$ time $k=n\times100$

$$100+2\times100+3\times100+\cdots n\times100$$

$$=100(1+2+3+\cdots n)$$

$$=100\left(n(n+1)/2\right)$$

$$=O(n^2)$$

2) A()

```
{
  int i,j,k,n;
  for(i=1 ; i<=n ; i++)
  {
    for(j=1 ; j<=i^2 ; j++)
    {
      for(k=1 ; k<=n/2 ; k++)
      {
        printf("Mohan");
      }
    }
  }
}
```

$i=1$ $j=1$ time $k=n/2*1$ time	$i=2$ $j=4$ $k=n/2*4$	$i=3$ $j=9$ time $k=n/2*9$	.....	$i=n$ $j=n^2$ $k=n/2*n^2$

$$= n/2*1 + n/2*4 + n/2*9 + \cdots n/2*n^2$$

$$= n/2(1+4+9+\cdots n^2)$$ [The sum of first n square]

$$= n/2\left(\frac{n(n+1)(2n+1)}{6}\right)$$

$$= O(n^4)$$

3) A()

```
{
  int i,j,k;
  for(i=n/2; i<=n; i++)          ——— n/2
  {
    for(j=1; j<=n/2; j++)        ——— n/2
    {
      for(k=1; k<=n; k=k*2)      ——> log_2 n
      {
        printf("Mohan");
      }
    }
  }
}
```

$= n/2 * n/2 * \log_2 n$

$= O(n^2 \log_2 n)$

4) A()

```
{
    int i, j, k;
    for(i=n/2; i<=n; i++)                ——— n/2
    {
        for(j=1; j<=n; j=2*j)            ——— log_2 n
        {
            for(k=1; k<=n; k=k*2)        ——— log_2 n
            {
                printf("Mohan");
            }
        }
    }
}
```

$\rightarrow n/2 (\log_2 n)^2$

$\Rightarrow O(n(\log_2 n)^2)$

```
A()
{
  for(int i=1; i<=n; i++)
  {
    for (j=1; j<=n; j=j+i)
    {
      printf("Mohan");
    }
  }
}
```

$i=1$	$i=2$	$i=3$	$i=k$		$i=n$
$j=1$ to n	$j=1$ to n	$j=1$ to n	.. $j=1$ to n	..	$j=1$ to n
n times	$n/2$ times	$n/3$ times	n/k times		n/n times

$$= n\left(1+\frac{1}{2}+\frac{1}{3}+\cdots\frac{1}{n}\right)$$

$$= n(\log n)$$

$$= O(n\log n)$$

```
A()
{
  int n = 2^(2^k);
  for(i=1; i<=n; i++)
  {
      j=2;
      while(j<=n)
      {
          j=j^2;
          printf("mohan");
      }
  }
}
```

$k=1$	$k=2$	$k=3$	...
$n=4$	$n=16$	$n=2^{2^3}=2^8$	
$j=2,4$	$j=2,4,16$	$j=2,2^2,2^4,2^8$	
$n*2$ times	$n*3$ times	$n*4$ times	

$n=2^{2^k} \Rightarrow \log_2 n \Rightarrow 2^k$

$O(\log\log n)$

```
A()
{
 for(int i=1; i<n; i++)
 {
   for(j=1; j<=i; j=j+2)
   {
     for(k=1; k<=n; k=k*2)
     {
       sum=sum+k;
     }
   }
 }
}
```

T.C → Outer loop → $O(n)$

Middle loop -

for each 'i' iterates approximately $i/2$ times on average.

$$1/2 + 3/2 + \frac{5}{2} + \ldots (2k-1)/2$$

→ sum of these iteration is proportional to n^2 in the worst case. $O(n^2)$

Inner loop - $\log(n)$

$O(n) * O(n^2) * O(\log n) \Rightarrow O(n^2 \log n)$

```c
#include<stdio.h>
int main(void)
{
    int sum=0, i, j,
    for(i=0 ; i<=n ; i=i+1)
    {   for(j=1; j<=i; j=j+1)
        {      if(j%i==0)
               {
                 for(k=1 ; k<=n ; k++)
                     {
                       sum= sum+k ;
                     }
               }
        }
    }
}
```

```
int fact(int n)
{
    if(n==0 || n==1)
        return 1;
    else
        return n*fact(n-1);
}
```

$$\boxed{T(n) = k + T(n-1)}$$ Recurrence Relation

$$\Downarrow$$

$$T(n) = \cancel{T(n-1)} + k$$

$$\cancel{T(n-1)} = \cancel{T(n-2)} + k$$

$$\cancel{T(n-2)} = T(n-3) + k$$

$$\vdots$$

$$T(1) = T(0) + k$$

$$\boxed{T(n) = k_1 * n + \cancel{T(0)}}$$

$$T(n) = k_1 * n$$

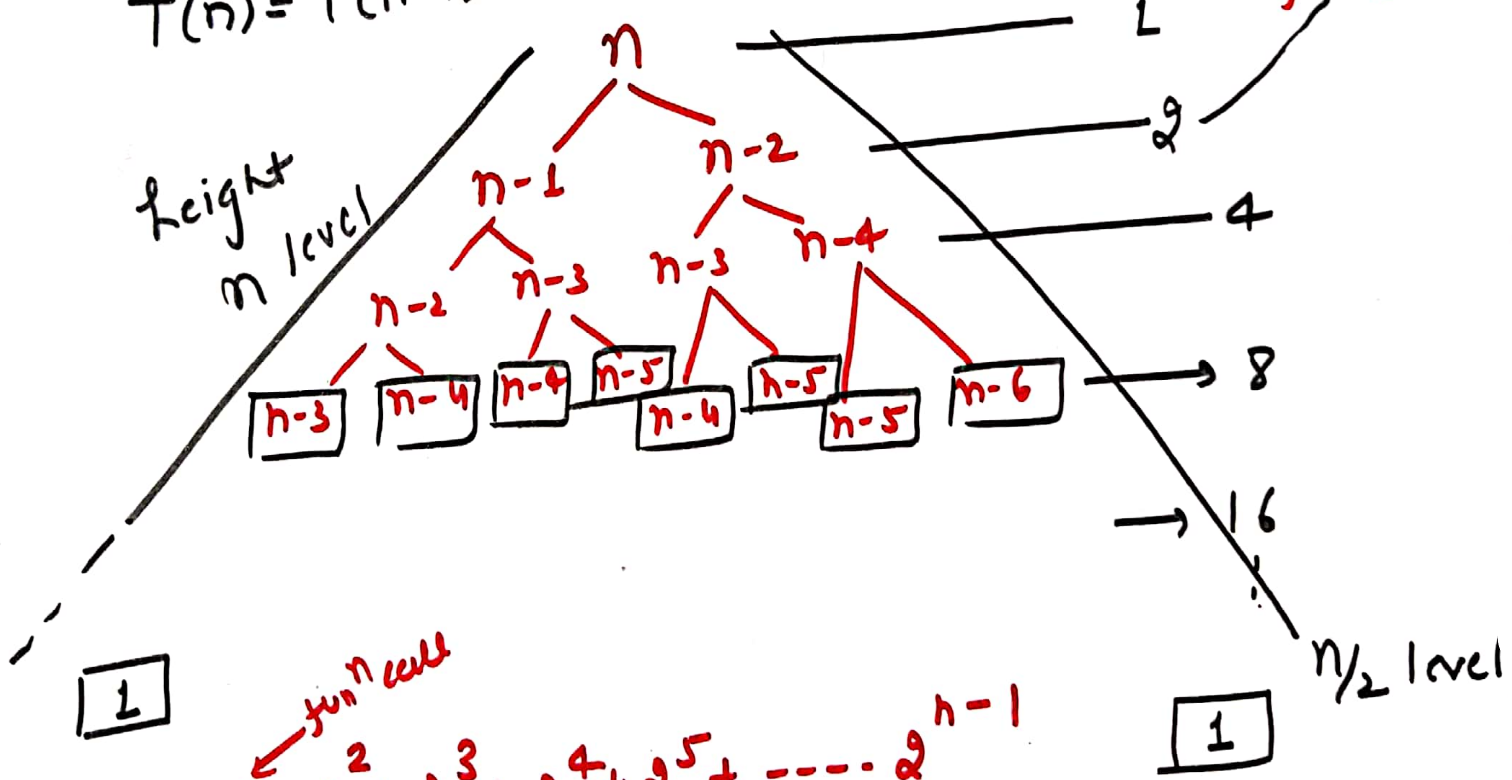
$$\boxed{T(n) = O(n)}$$

```
int fib(int n)
{
    if(n==0)
        return 0;
    if(n==1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

$$\boxed{T(n) = k + T(n-1) + T(n-2)}$$ R. Relation

(How many fun call)

$T(n) = T(n-1) + T(n-2) + k$



fun^n call

$1 + 2 + 2^2 + 2^3 + 2^4 + 2^5 + \cdots 2^{n-1}$

$\boxed{2^n - 1} \longrightarrow O(2^n)$

exponential No. of fun^n call