Time and Space Complexity - 2

Summary of T.C and S.C-1

Property Name	Property
Log of 1	$\log_a 1 = 0$
Log of the same number as base	$\log_a a = 1$
Product Rule	$\log_a(mn) = \log_a m + \log_a n$
Quotient Rule	$\log_a\left(\frac{m}{n}\right) = \log_a m - \log_a n$
Power Rule	$\log_a m^n = n \log_a m$

Change of Base Rule	$\log_b a = \frac{\log_c a}{\log_c b}$ (OR) $\log_b a \cdot \log_c b = \log_c a$
Equality Rule	$\log_b a = \log_b c \Rightarrow a = c$
Number Raised to Log	$a^{\log_a x} = x$
Other Rules	$\log_{b^n} a^m = \frac{m}{n} \log_b a$ $-\log_b a = \log_b \frac{1}{a}$ (OR) $= \log_{\frac{1}{b}} a$

$$a^{\log_a x} = x$$

$$\log_a^{\log_a x} a = \log_a x$$

$$\log_a x \cdot \log_x a = \log_a a$$

LHS = RHS

$$\log_8^{\log_8 4} 4 = 4^1 = 4$$

Big-Oh [O( )] : Order of

O( ) : Upper Bound

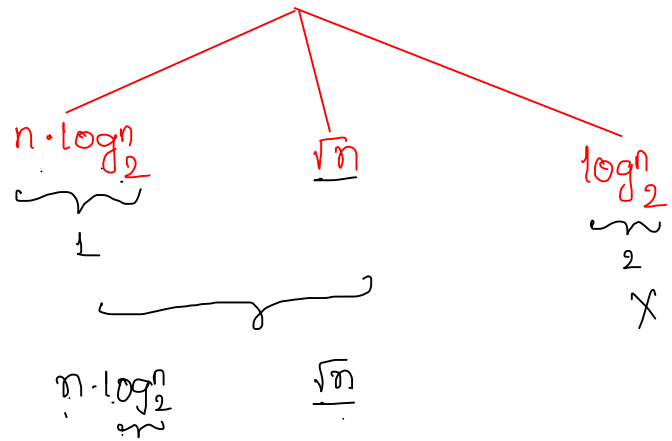
Sno	Function Name	Function Expression
1	Constant	1
2	Logarithmic	log(n)
3	Square root	√n
4	Linear	n
5	Linearithmic	n.log(n)
6	Quadratic	n^2
7	Cubic	n^3
8	Exponential	2^n
9	Factorial	n!

$f(n) = n^2 + 2n + 3\log n + \sqrt{n}$

$\hookrightarrow O(n^2)$

$f(n) = n \cdot \log n + \sqrt{n} + \log_2 n$

$O(\quad)$



Don't By-heart

understand

$O(n \log_2 n)$

$\log_{10}^{n}$ $\quad$ $10^{\underbrace{n}_{2}}$

$\underbrace{n}_{1}$

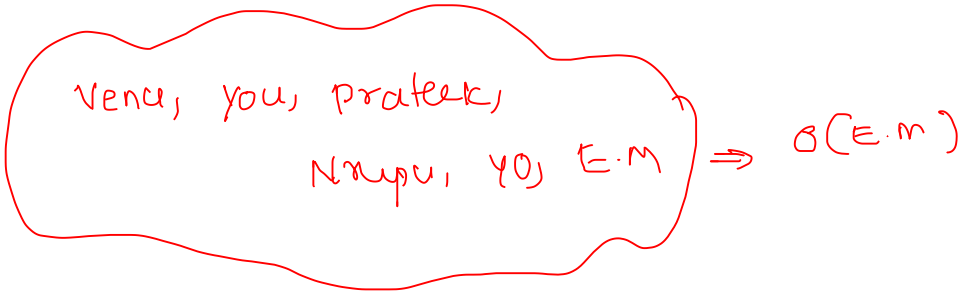
$\rightarrow$

$(\log_{10}^{n})^{k} < n^{\epsilon}$ $\quad$ $\log_{10}^{n} \cdot \log_{10}^{n}$ $\quad$ $n \cdot \underbrace{\log_{10}^{10}}_{1}$

$\underbrace{(\log_{10}^{n})^{2}}$ $< n$ $\Rightarrow$ 10^{n} bigger ✓

Comparison of two functions

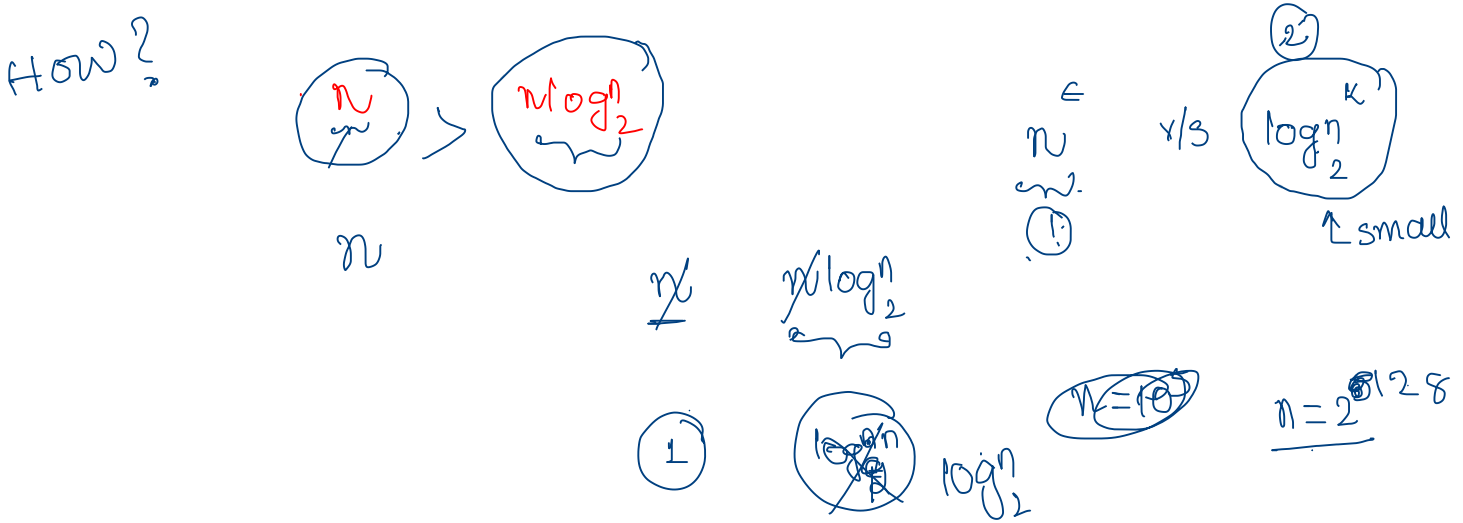
$f(n)+g(n) = O(\max(f(n), g(n)))$



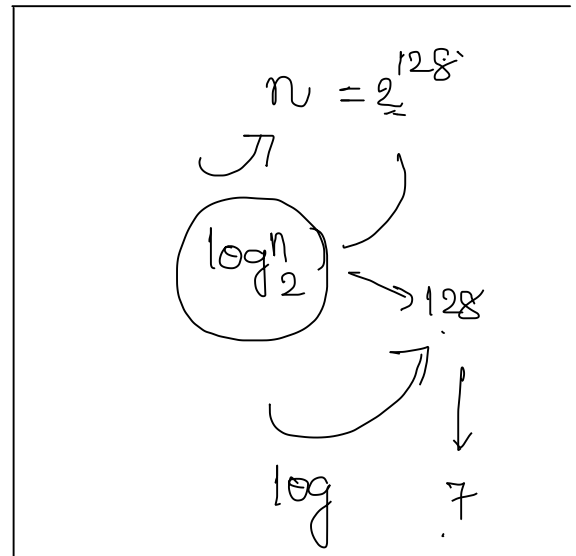
Ex:-

f1(n) = n^2 + nlogn + n ⇒ $O(n^2)$

f2(n) = n + nloglogn + nlogn ⇒ O( )



$$f2(n) = \underset{1}{n} + \underset{2}{n\log\log n} + \underset{3}{n\log n} \Rightarrow O(n \cdot \log_2 n)$$



$n \cdot \log\log_2 n$ (1) vs $n \cdot \log_2 n$ (2)

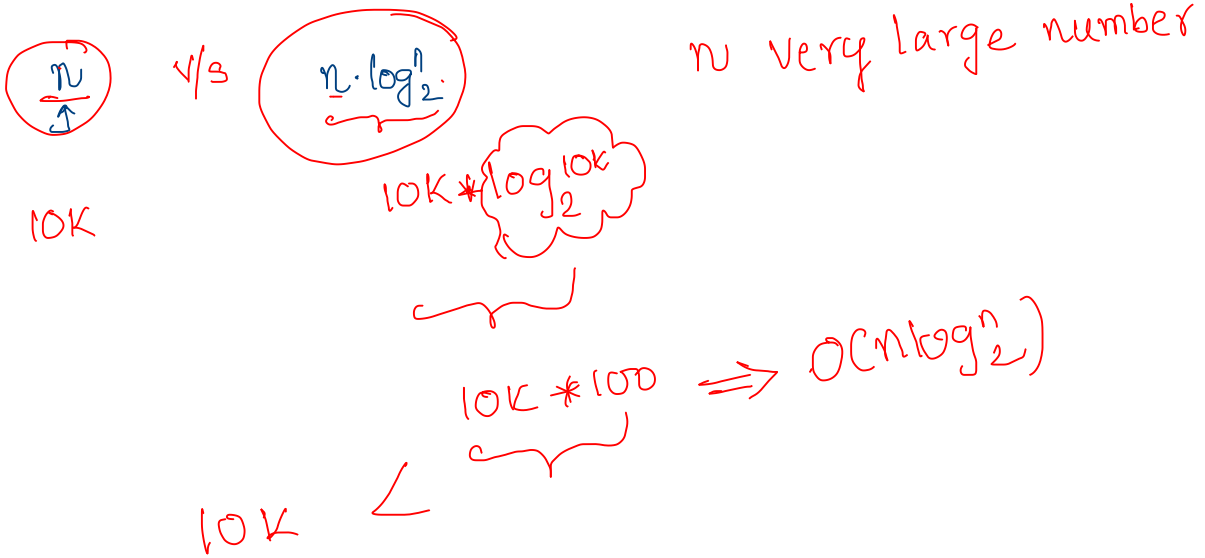
How $\log \cdot \log_2 n$ > $\log_2 n$ $\Rightarrow 128$

$n = 2^{128}$

$128 \cdot \log_2 n$ $\quad n = 2^{128}$

$\log_2^{128}$

$\log_2 \log_2^{2^{128}}$ → 128 (2^7) ⇒ 7


n
v/s
n · log2 n
n very large number
10K
10K * log2 10K
10K * 100
⇒ O(n log2 n)
10K

1) $n^{\epsilon} > (\log n)^{k}$

k : very large value. (1000, 10,000)

ϵ : small value (0.1, 0.001)

t_1 t_2 t_3 t_4 . . .

$\Rightarrow$ $a, ar, ar^2, ar^3, \ldots ar^n$ [common ratio (r) $\Rightarrow \frac{t_3}{t_2} = \frac{t_2}{t_1}$]

2) Increasing G.P :-

$= 1 + 2 + 2^2 + 2^3 + \cdots + 2^n \Rightarrow S_n = \frac{a(r^n-1)}{r-1}$

$a = 1$, $r = 2$ $\Rightarrow S_n = \frac{1(2^n-1)}{2-1} = 2^n - 1 \Rightarrow O(2^n)$

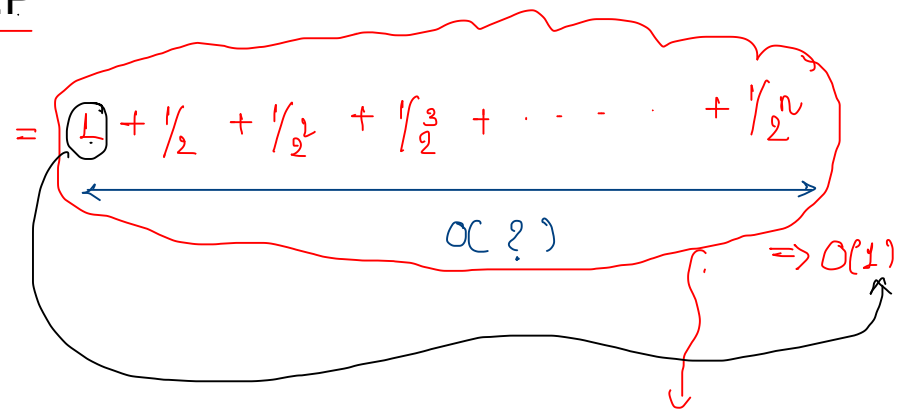
$2^n - 1 \hookrightarrow O(2^n)$

when you are writing O():-

1) ignore the constants

2) which ever is bigger that needs to be kept inside O(--)

3) Decreasing G.P



GP, $r = 1/2$

$S_n = \frac{a(r^n - 1)}{r - 1}$

$S_\infty = \frac{a}{1-r}$ ✓

$= 1 + 1/2 + 1/2^2 + 1/2^3 + \cdots + 1/2^n + 1/2^{n+1} + \cdots \infty$

$S_\infty = \frac{a}{1-r} = \frac{1}{1-1/2} = \frac{1}{1/2} = 2$

constant $\Rightarrow O(1)$

* proof not required (just remember)

4)

$1 + 1/2 + 1/3 + 1/4 + 1/5 + \ldots\ldots\ldots\ldots\ldots 1/n =$ $O(\log_2 n)$

5) n^n v/s n!

① $n! = n \times (n-1) \times (n-2) \times \cdots \cdots 1$

② $n^n = n \times n \times n \times \cdots \cdots n$

$n^n > n!$

BUT if I apply log.

$\log_{10} n^n \simeq \log_{10} n!$

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

n^n , $n!$

$5! = 120$

$5^5 = 3125$

$n = 10!$

after applying the log

$\cong$

$\log n! \cong \log n^n$

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

1) see approximately how many times loop is running
[exact value is not required]

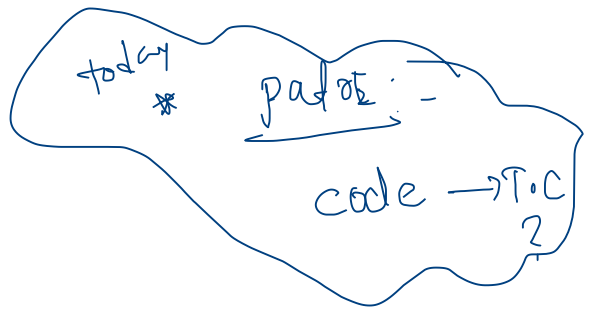
2) very MINUTE / SMALL details we need to take into CONSIDERATION

while finding T.C

part1 : ~~code~~

T.C1 & compared.

T.C2



Template 1:-

```
for(i=1;i<=n;i=i+a)
{
        print("*")
}
```

Template 2:-

```
for(i=1;i<=n;i=i*2)
{
        print("*")
}
```

Template 3:-

```
for(i=n;i>=0;i=i-1)
{
        print("*")
}
```

Template 4:-

```
for(i=n;i>=0;i=i/2)
{
        print("*")
}
```

```
for(i=1;i<=n;i=i+1)
{
    print("*")
}
```

→ n → $\Rightarrow O(n)$

```
for(i=1;i<=n;i=i+2)
{
    print("*")
}
```

→ n/2 times = $\frac{1}{2} \cdot n$ = $O(n)$

```
for(i=1;i<=n;i=i+4)
{
    print("*")
}
```

→ n/4 times = $O(n)$

Note :- print(), if, -else, break, continue, return ... → $\Rightarrow O(1)$

Nested Loops

```
for(i=1;i<=n;i=i+4) ⇒ n/4
{
    print("*") ⇒ Loop : O(n)
}
```

→ $\Rightarrow \frac{n^2}{4}$ $\Rightarrow O(n^2)$

n = O(n)

```
for(i=1;i<=n;i=i+n)
{
    print("*")
}
```

$\Rightarrow O(1)$

reverse

```
for(i=n;i>0;i=i-1)
{
        print("*")
}
```

→ n times : O(n)

```
for(i=n;i>0;i=i-2)
{
        print("*")
}
```

⇒ O(n)

$n = 2^{45}$

// Assume n is power of 2

if $n = 2^5$ then How many * , print

```
for(i=1;i<=n;i=i*2)
{
        print("*");
}
```

i = 1 2 4 8 16 32 64

[* * * * * *]

$n = 2^5$ → 6 = $(n+1)$

$n = 2^3$ →

```
// Assume n is power of 2

for(i=1;i<n;i=i*2)
{
        print("*");
}
```

of (*)

$n = 2^2 \Rightarrow 2$

$2^3 \Rightarrow 3$

$2^4 \Rightarrow 4$

$2^{100} \Rightarrow 100$

$\log_2 2^k = k$

$n = 625 \hookrightarrow \sqrt{n} \hookrightarrow 25$

$n = 169 \hookrightarrow \sqrt{n} \hookrightarrow 13$

$n = 81 \hookrightarrow \sqrt{n} \hookrightarrow 9$

* $i = 1 \xrightarrow{\times 2} 2 \xrightarrow{\times 2} 4 \xrightarrow{\times 2} 8 \xrightarrow{\times 2} 16 \ldots \rightarrow n \Rightarrow O(\log_2 n)$

$i = n \xrightarrow{/2} n/2 \xrightarrow{/2} n/4 \rightarrow n/8 \rightarrow n/16 \ldots \rightarrow 1 \Rightarrow O(\log_2 n)$

1)

```
for( i=1; i<=2^n ; i=i*2 )
{
        print("*");
}
```

$\log_2^{2^n}$

$n \cdot \log_2^2 = n \cdot 1$

$= O(n)$ ✓

1)

```
for( i=1; i<=n ; i=i*2 )   ⇒O(log_2^n)
{
        print("*");
}
```

1)

```
for( i=1; i<=2^n ; i=i*2 )
{
        print("*");
}
```

Ans : O(n)

2)

```
for( i=1; i<=n ; i=i+n )   ⇒ O(1)
{
        print("*");
}
```

2)

```
for( i=1; i<=n ; i=i+n )
{
        print("*");
}
```

Ans) O(1) ✓

Nested-Loop (extra careful!)

3)

Type1: No-dependency

```
OL  i → for( i=1; i<=n ; i=i+1 )  ⇒ n
        {
IL        j →for(j=1; j<=n; j++)  ⇒ ? n
            {
                print("*");
            }
        }
```

$= n * n = O(n^2)$

Type2: Dependency ✓

```
→ i for( i=1; i<=n ; i=i+1 )
    {
        j for(j=1; j<=i; j++) →j depends on i
        {
            print("*");
        }
    }
```

→ later

3)

```
for( i=1; i<=n ; i=i+1 )
{
        for(j=1; j<=n; j++)
        {
                print("*");
        }
}
```

Ans) O(n^2)

4)

No-dep

(trap)

n^2

```
for( i=1; i<=n ; i=i+1 )
{
      for(j=n/2; j<=n; j=j+n/2)
       {
            print("*");
       }
}
```

i → n

j → 2

$j = \frac{n}{2} + \frac{n}{2} = n + \frac{n}{2} = 1.5\,n \leq n$ ×

**

only 2 times

$2 * n \Rightarrow O(n)$

4)

```
for( i=1; i<=n ; i=i+1 )
{
        for(j=n/2; j<=n; j=j+n/2)
        {
                print("*");
        }
}
```

Ans) O(n) ✓

No-Dep

5)

```
i for(i=1;i<=n;i++)  → n
  {
      j for(j=1;j<=n/4;j++)  → n/4
        {
          k for(k=1;k<=n;k++)  → 1
            {

              ↳print("*")
                break;
            }
        }
  }
```

$n \times \frac{n}{4} \times 1 \Rightarrow O(n^2)$

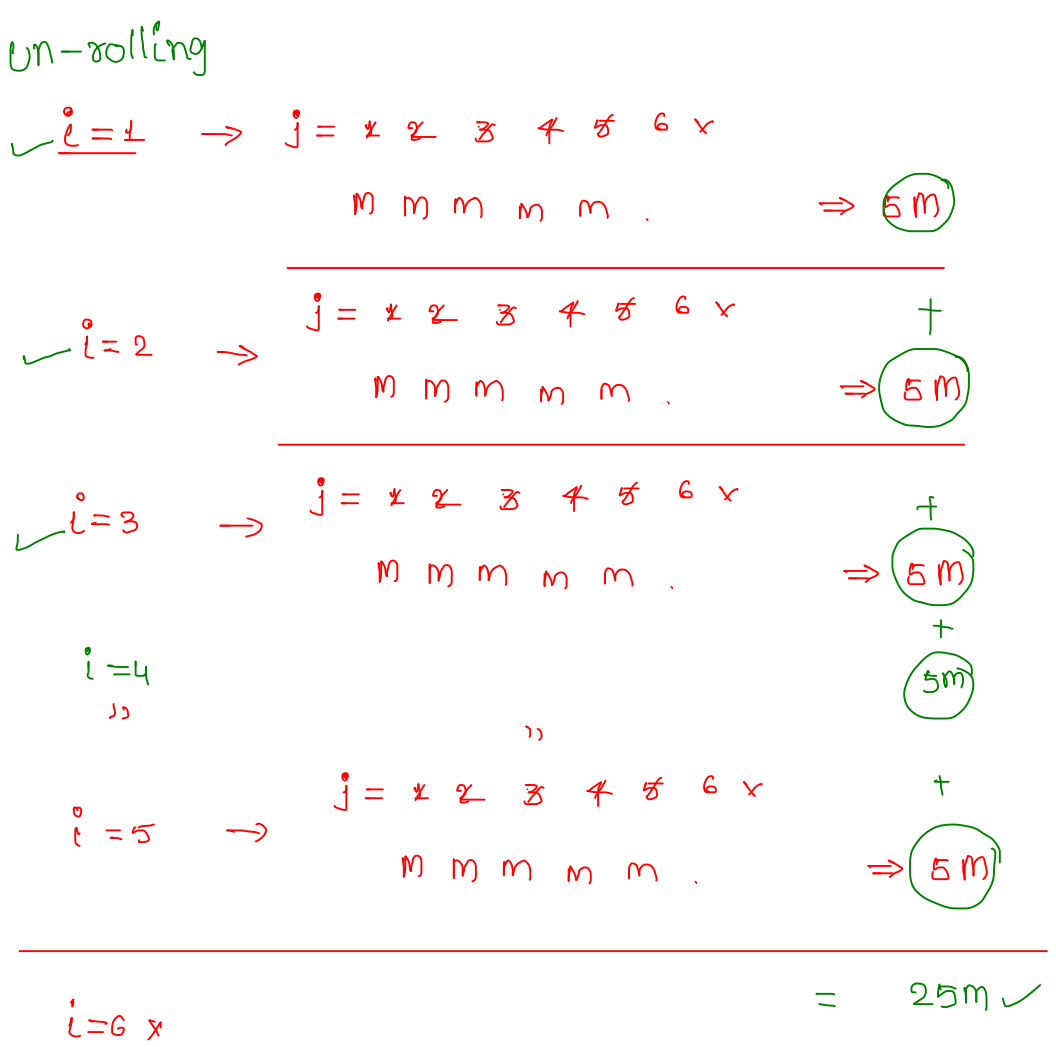
5)

```
for(i=1;i<=n;i++)
{
        for(j=1;j<=n/4;j++)
        {
                for(k=1;k<=n;k++)
                {

                        print("*")
                        break;
                }
        }
}
```

Ans) O(n^2) ✓

```
* for(i=1;i<=5;i++)
  {
      for(j=1;j<=5;j++)
      {
          print("masai")
      }
  }
```



dependency

↳ un-rolling.

6)

```
for(i=1; i<=n; i++)
{
    for(j=1; j<=i; j++)
    {
        print("*");
    }
}
```

⇒ i times

How many times j loop will execute(?)

$i = 1 \rightarrow 1$

$i = 2 \rightarrow 2$

$i = 3 \rightarrow 3$

.

.

.

$i = n \rightarrow n$

x

$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \Rightarrow O(n^2)$

$= \frac{n^2+n}{2}$

```
for(j=1; j<=i; j++) ⇒
{
    print("*");
}
```

6)

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=i; j++)
        {
                print("*");
        }
}
```

Ans : O(n^2)

7)

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

dep
↳ Un-rolling

$j \Rightarrow n/i$

$i = 1 \rightarrow n$

$2 \rightarrow n/2$

$3 \rightarrow n/3$

$\vdots$

$n \rightarrow n/n = 1$

$\frac{n}{i}$

```
for(i=1; i<=n; i=i+2) → n/2
{
    P(*)
}
```

$3 \rightarrow n/3$

$a \rightarrow n/a$

$i \rightarrow n/i$

$n + n/2 + n/3 + \cdots + 1$

$n\left[1 + 1/2 + 1/3 + \cdots + 1/n\right] = O(n \cdot \log n)$

$\log n$

* ~~a) O(n+1) O(n)~~

b) $O(n^2)$

* ~~c) O(1)~~ ✓

d) $O(n \log n)$ → sachin Ranta Ahamed Azad. Yogesh

7)

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

n → $\frac{n}{i}$

$n * \frac{n}{i} = \frac{n^2}{i} \Rightarrow O(n^2)$

$i=1 \quad i=2 \quad \cdots \quad i=n$

$(n) + \left(\frac{n}{2}\right) + \left(\frac{n}{3}\right) \quad \left(\frac{n}{n}\right)$

$n + \frac{n}{2} + \frac{n}{3} + \cdots \frac{n}{n}$

$n\{1 + 1/2 + 1/3 + \cdots + 1/n\}$

$\rightarrow \log n$

7)

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

n → n → n^2

$i = 1 \quad 2 \quad 3 \quad \cdots \quad n$

$(n) + (n) + n + \cdots n$

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

7)

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

Ans) O(nlogn)

8)

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

dep
↳ un-rolling

i loop

i = 1 → 1
2 → 1
3 → 1
⋮
n → 1

1+1+1+ ... n]

= n

O(n)

a) O(n)
b) O(n³) ← Brad
c) O(n²)

```
for(j=1; j<=i; j=j+i) ⇒ 1 time O(1)
{
    printf("*")
}
```

i = 5
j = 1 2 3 4 5 6
*

i = 100
j = 1 2 ... 101
*

```
for(i=1;i<=n;i=i+a) ⇒ n/a times
{
    print("*");
}
```

n/a, 1, n
1, n

8)

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

Ans) O(n)✓

9) Home work

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=i^2; j=j+i)
        {
                printf("*")
        }
}
```

10)

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=n; j=j*2)
        {
                printf("*")
        }
}
```

10)

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=n; j=j*2)
        {
                printf("*")
        }
}
```

Ans) O(nlogn)

11)

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=i; j=j*2)
        {
                printf("*")
        }
}
```

11)

```
for(i=1; i<=n; i++)
{
        for(j=1; j<=i; j=j*2)
        {
                printf("*")
        }
}
```

Ans) O(nlogn)

12)

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

13)

```
for(i=1; i<=n; i=i*2)
{
        for(j=1; j<=i^2; j=j++)
        {
                printf("*")
        }
}
```

13)

```
for(i=1; i<=n; i=i*2)
{
        for(j=1; j<=i^2; j=j++)
        {
                printf("*")
        }
}
```

Ans) O(n^2)

14)

```
for( k=1; k<=n; k++)
{
        for(i=1; i<=n; i=i*3)
        {
                j=i;
                while(j>1)
                {
                        print("*")
                        j=j/3
                }
        }
}
```

14)

```
for( k=1; k<=n; k++)
{
        for(i=1; i<=n; i=i*3)
        {
                j=i;
                while(j>1)
                {
                        print("*")
                        j=j/3
                }
        }
}
```

Ans) O(n(logn)^2) [base-3]

15)

```
for(i=1; i<=n; i++)
{
        j=1
        while(j<=n)
        {
                j=2*j
        }
        for(k=1;k<=n;k++)
        {
                c=c+1
        }
}
```

15)

```
for(i=1; i<=n; i++)
{
        j=1
        while(j<=n)
        {
                j=2*j
        }
        for(k=1;k<=n;k++)
        {
                c=c+1
        }
}
```

Ans) O(n^2)

16)

```
function fun(n)
{
        for(i=1;i<=n;i++)
        {
                p=0
                for(j=n; j>1; j=j/2)
                {
                        ++p
                }

                for(k=1; k<p; k=k*2 )
                {
                        ++q
                }
        }
}
```

17) Consider an array of any n positive integers in random order

```
function fun(arr,n)
{
        j=1;
        for(i=0; i<n; i++)
        {
                while(j<n && arr[i]<arr[j])
                {
                        j++;
                }
        }
}
```

18)

```
for(i=1;i<=n;i++)
{
        arr.sort()
        j=1
        while(j<=n)
        {
                j=j*2
        }
}
```

19)

```
for(i=n; i>=1; i=i/2)
{
        for(j=1; j<i; j=j*2)
        {
                print("masai")
        }
}
```

Why TLE comes?

- **Online Judge Restrictions:** TLE comes because the Online judge has some restriction that it will not allow to process the instruction after a certain Time limit given by Problem setter the problem(1 sec).
- **Server Configuration:** The exact time taken by the code depends on the speed of the server, the architecture of the server, OS, and certainly on the complexity of the algorithm. So different servers like practice, CodeChef, SPOJ, etc., may have different execution speeds. By estimating the maximum value of N (N is the total number of instructions of your whole code), you can roughly estimate the TLE would occur or not in 1 sec.

```
MAX value of N                          Time complexity
    10^8                                O(N) Border case
    10^7                       O(N) Might be accepted
    10^6                                O(N) Perfect
    10^5                                O(N * logN)
    10^4                                O(N ^ 2)
    10^2                                O(N ^ 3)
    10^9                                O(logN) or Sqrt(N)
```

- So after analyzing this chart you can roughly estimate your Time complexity and make your code within the upper bound limit.
- **Method of reading input and writing output is too slow:** Sometimes, the methods used by a programmer for input-output may cause TLE.