Time & Space complexity

→ order of Time ←

→ order of space .

Algorithm ← finite steps to complete a task

selecting Best Algo

→ Time complexity

space complexity ←

- Best Case
- Average Case
- Worst Case

→ $A_1 \leftrightarrow A_2$

Asymptotic Notations → Notations to represent $T(n)$

(1) Θ (Theta)

(2) O (Big O)

(3) Ω (Big Omega)

(4) small o (o)

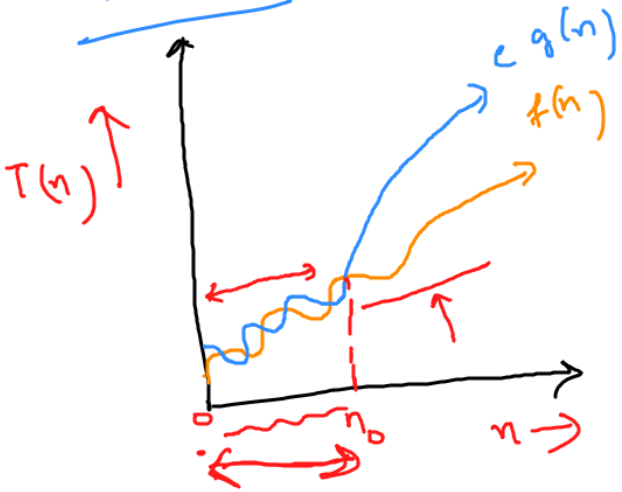
(5) small omega (ω)

Average

worst

Best

Big O



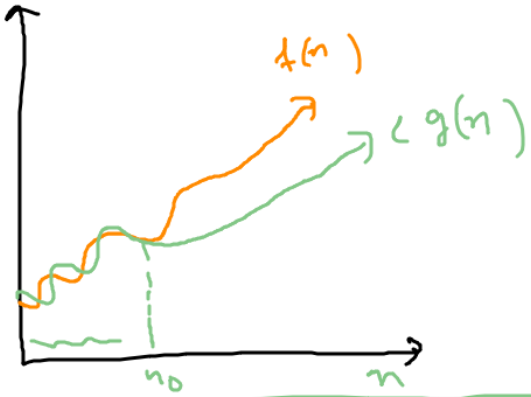
Constant

$c > 0$

$0 < f(n) \leq c\,g(n)$

$n_0 \geq 1$

Big Omega



$f(n) \geq c\,g(n) > 0$

$n_0 \geq 1$

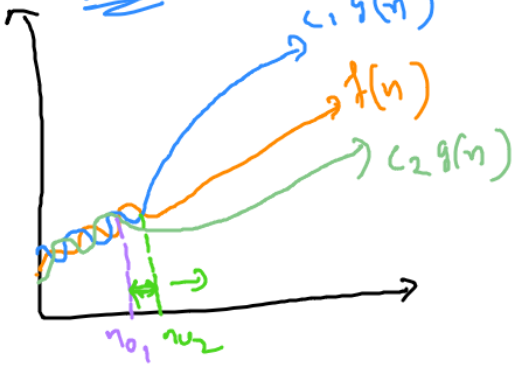
n → input size

no. of inputs → Array

value → fact(n)

T

Theta (Θ)



$c_1, c_2 > 0$

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

$n_0, n_2 \geq 1$

```
int i; → 4 B

for(int i=0; i<=n; i++)        // n: 4B
{
   → cout<<"Hi";
}
```

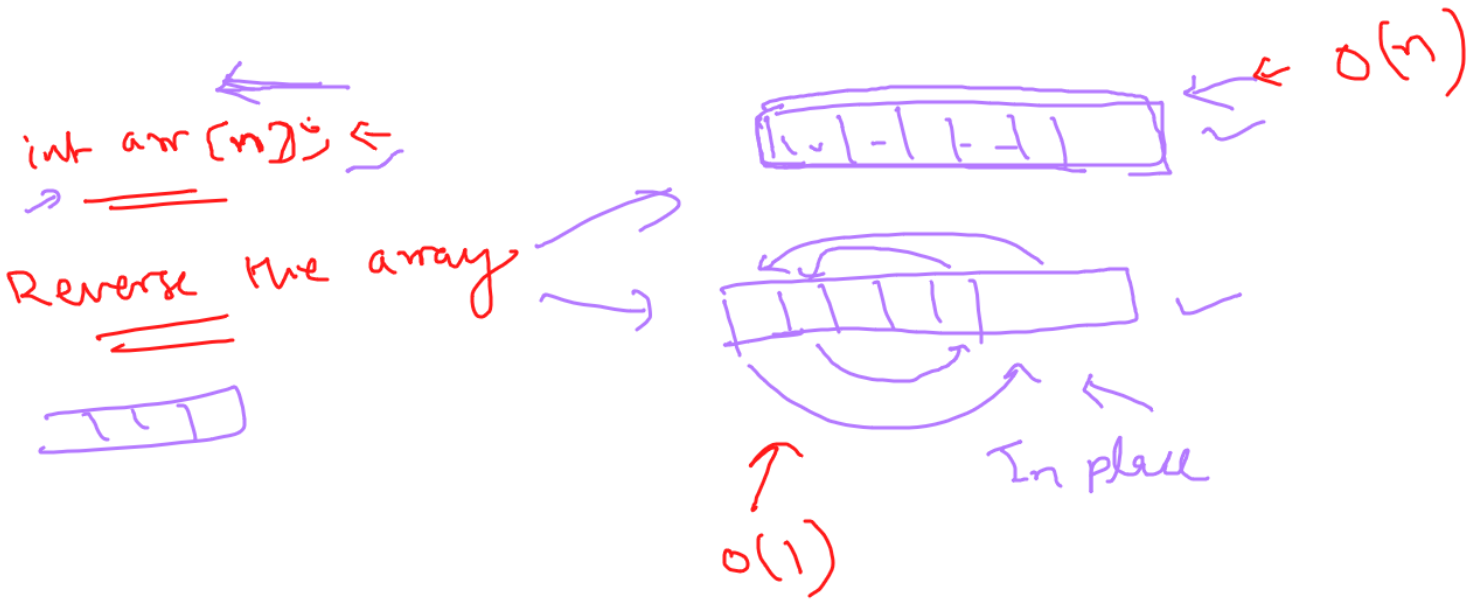
$S(n) = 2$ (constant)

$O(1)$ $O(c)$

```
int arr[n];
```

```
for(int i=0; i<n; i++)
{
    s.o.p[arr[i]);
}
```

$\leftarrow O(1) = S(n)$


int arr[n];
Reverse the array
O(n)
In place
O(1)

Types of Time complexity :-

```
System.out.println("Hi");
```

O(1)

```
if(a < b)        <- O(1)
{
    System.out.println("Hi");
}                <- O(1)
```

=> O(1)

```
System.out.println("Hi");      <- O(1)
if(a == 5)                     <- O(1)
    System.out.println("Hi");  <- O(1)
else                           <- O(1)
    System.out.println("Bye");
```

O(1)

Constant Time complexity

T(n) -> order of n

```
for (i=1; i<=n; i++)
    System.out.println("Hi");
```

→ O(n)

→ Linear T(n)

```
fun()
{
    if (a < 6)
    {
        System.out.println("Hi");
        System.out.println("Hello");
    }
    else
    {
        for (i=0; i<n; i++)
        {
            System.out.println("Bye");
        }
    }
}
```

O(1)

O(n)

O → n-1

for (i=1; i<=n; i+=2)
s.o.p ("Hi");

$O(n/2) \rightarrow O(n)$ ✓

Linear

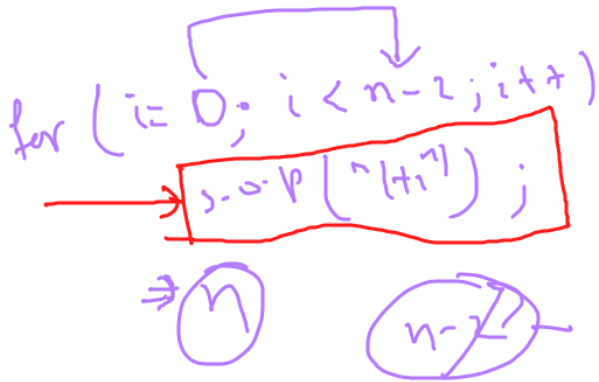
$n \rightarrow \infty$

+2 −2 ✓2

$T(n) = 3n^2 + 5n + 10$

$= n^2 + n$

$= O(n^2)$



$O(n)$

```
for (i=1; i*i <= n; i++)
    S.o.p("hello");
```

$i^2 <= n$

$i <= \sqrt{n}$

$O(\sqrt{n})$

```
for (i=n; i>=1; i /= 2)
    s.o.p(n+n);
```

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

```
for (i=1; i<=n; i*=2)
    S.o.p("hi");
```

$n = 64$

i: 1, 2, 4, 8, 16, 32, 64, 128

$\log_2 n$

$\log_2 n$

$i \Rightarrow 64$

$\Rightarrow 2^k = 64$

$\Rightarrow k = \log_2(64)$

$\Rightarrow k = \log_2(n)$

$7 \Leftrightarrow 64$

$i \Leftrightarrow 64$

$\log(n) + 1$

Logarithmic

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

```
for (i=1; i<=n; i++ )        <- O(n)
{   for (j=1; j<=n; j++)     <- O(n)
    {   s.o.p("Hi");
    }
}
```

$T(n) = O(n*n)$

if $n = n$

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

→ Quadratic

i	j
1	1,2,..n — n times
2	1,2...n — n times
⋮	⋮ — ⋮
n	1,2,..n — n times

$\underbrace{n+n+\dots+n}_{n \text{ times}} = n*n$

```
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        for(k=1;k<=100000;k++)
        {
            s.o.p("hi");
        }
    }
}
```

O(n)

O(n)

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

n

$T(n) = O(n^3)$

↳ cubic

$0, 1, 2, 3, \ldots \log n$

```
for ( i=1; i<=n; i*=2 )      ← log(n)      n log(n) ✓
  for ( j=1; j<=i; j++ )     ← (n)    T(n) =
    s.o.p("Hi");
```

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

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

GP series

i	j
1	1 -1
2	1,2 = 2
4	1,2,3,4 - 4
8	1,2 ... 8 - 8
⋮	⋮
n	- n

$1 + 2 + 4 + 8 + \cdots + n$

$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{\log n}$

$$a\left(\frac{1-r^n}{1-r}\right) \qquad a = 1,\ r = 2,\ n = \log n + 1$$

$$1\left(\frac{1-2^{\log n+1}}{1-2}\right)$$

$\Rightarrow 2^{\log n + 1}$

$\Rightarrow 2^{\log n} \cdot 2 \Rightarrow n$

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

a = 1, b = 1

```
while (b <= n)
{   a += 1;
    b += a;
    s.o.p("hi");
}
```

$T(n) = ?$

$n = 10$

a	b
1	1+2 = 3
2	3+3 = 6
3	6+4 = 10
4	

$1 + 2 + 3 + 4 = 10$

↑ 4th step

$1 + 2 + 3 + 4 \cdots + k \approx n$

↑ kth step

$\frac{k(k+1)}{2} = n$

$k^2 \approx n \Rightarrow \boxed{k = O(\sqrt{n})}$