M-01 → Java Programming

M-02 → Data Structure and Algorithm

# Time And Space Complexity

Todays Content :-

→ factors

→ Calculating iterations

→ TC & S.C

Q- Given N, calculate the no of factors.

N = 10 → 1, 2, 5, 10 → 4

N = 16 → 1, 2, 4, 8, 16 → 5

Range
- smallest factor :- 1
- largest factor :- N

```
int c = 0;
for (int i = 1; i <= n; i++)
    if (N % i == 0)
        c = c+1;
return c;
```

[1, N] ⇒ 1+N-1 = N

[a, b] = a+b-1

iterations → N

Ass:- $10^8$ iterations in 1 sec.

$10^9$ → 1 sec.

Input :-

| N | iterations | execution time |
|---|---|---|
| $10^9$ | $10^9$ | 10 sec. |
| $10^{18}$ | $10^{18}$ | $10^{10}$ sec. ≈ 317 years |

$10^9$ → 1 sec.

$1$ → $\frac{1}{10^8}$ sec.

$10^9$ → $\frac{10^9}{10^8}$ sec = 10 sec.

$10^{10}$ → $\frac{10^{10}}{10^8} = 10^2$

1st → 2nd → 3rd → 4th

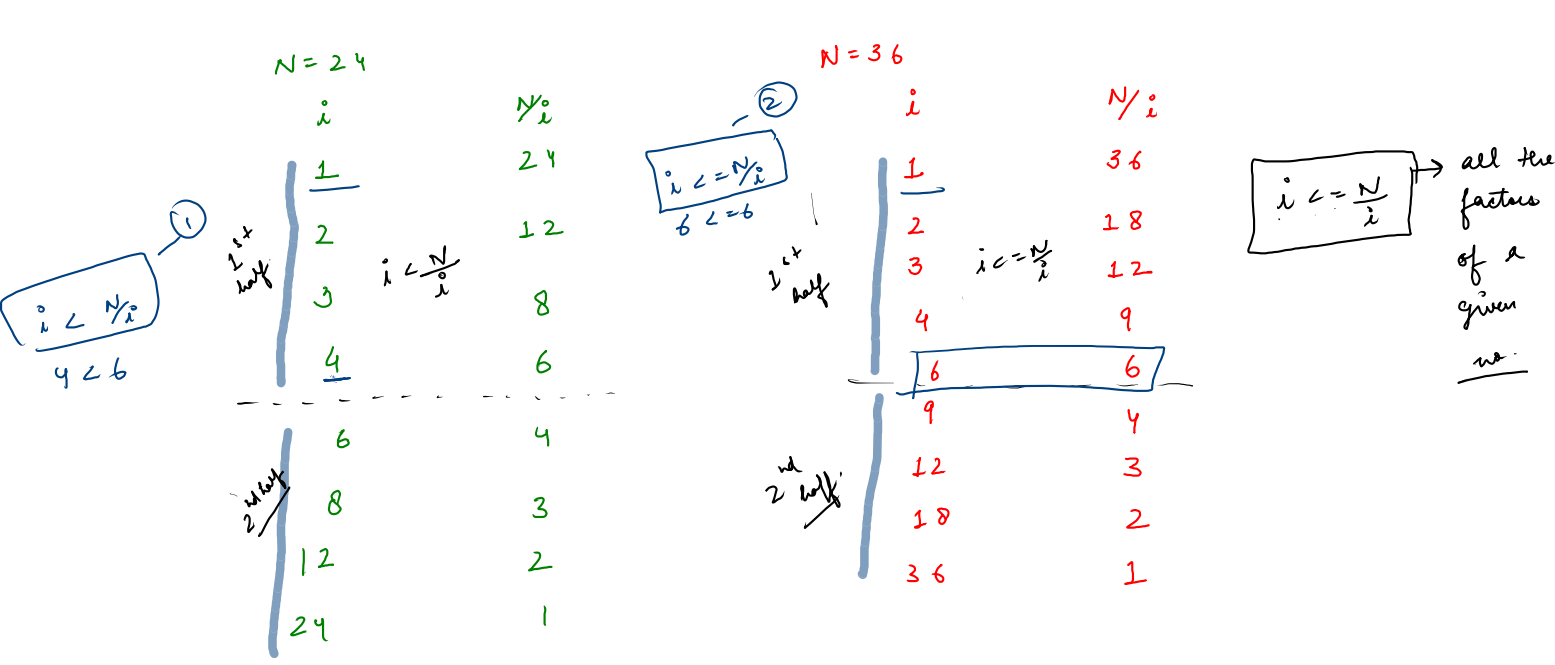
idea:-

$a * b = N$

① ↳ a & b are the factors of N.

② ↳ $b = \frac{N}{a}$

③ ↳ a & $\frac{N}{a}$ are the factors of N.


N = 24
i
N/i
1
24
2
12
3
8
4
6
6
4
8
3
12
2
24
1
i <= N/i
4 <= 6
1
1st half
i <= N/i
2nd half
N = 36
i
N/i
1
36
2
18
3
12
4
9
6
6
9
4
12
3
18
2
36
1
i <= N/i
6 <= 6
2
1st half
i <= N/i
2nd half
i <= N/i
all the factors of a given no.

$$i \, c = \frac{N}{i}$$

$$\boxed{i * i \, c = N}$$

$$\Rightarrow \quad i^2 \, c = N$$

$$\Rightarrow \quad \boxed{i \, c = \sqrt{N}}$$

```
int countfactors (int N) {
    int c=0;
    for( int i=1; i*i<=N; i++) {
        if (N% i==0) {
            if ( i==N/i) c=c+1;
            else c=c+2;
        }
    }
    return c;
}
```

$i <= \sqrt{N}$

↓

Math.sqrt()✗

$\sqrt{10^{18}} = \sqrt{10^9 * 10^9}$

$= \sqrt{(10^9)^2} = 10^9$

No. of iterations

$[1, \sqrt{N}] = 1 + \sqrt{N} - 1 = \sqrt{N}$

317 years

↓

10 sec.

| Input | Iterations | execution time |
|---|---|---|
| $N$ | | |
| $10^{18}$ | $10^9$ | 10 sec. |

$10^8 \rightarrow 1$ sec.

$1^* \rightarrow \frac{1}{10^8}$

$10^9 \rightarrow \frac{10^9}{10^8} = 10$ sec

# Comparing Algorithms

↳ problem statement / Input / Test case

A' algo

→ 20 sec.

(Windows XP)

(Macbook M2)

12 sec.

(C++)

(Volcano)

(Antartica)

10 sec.

B algo

15 sec.

(Macbook M2)

python

(15 sec)

(C++)

10 sec.

(Antartica)

Obs :-

Execution time depends

'n' no. of factors.

Obs 2:- 'iterations' is the parameter, independent of external factors

## Calculating iterations

$[a, b] \Rightarrow a + b - 1$

(a)

```
f :- void fu (int N) {

    int s = 0;

    for( int i = 0; i < N; i++)

        s = s + i;

    }
```

$[0, N-1]$

$\Downarrow$

$0 + N - 1 - 1 = N - 2$

$[0, N-1] \Rightarrow 0 + N - 1 - 1 = N - 2$

p.s. void fu (int N) {

int s = 0;

for ( int i = 1; i * i <= N; i++)

{

s = s + i;

}

$[1, \sqrt{N}] : 1 + \sqrt{N} - 1 = \sqrt{N}$

```
f :- void fn (int N) {
    int i = N;
    while ( i > 1 ) {
        i = i / 2;
    }
}
```

| iteration | After each iteration |
|---|---|
| 1 | $i = N/2 \rightarrow \frac{N}{2^1}$ |
| 2 | $i = N/4 \rightarrow N/2^2$ |
| 3 | $i = i/2 = N/8 \rightarrow N/2^3$ |
| 4 | $i = \frac{N}{16} \rightarrow N/2^4$ |
| ⋮ | ⋮ |
| k | 1 $\rightarrow N/2^k$ |

No. of iterations $\Rightarrow \log_2 N$

// after k iterations, code stops

$\Rightarrow \frac{N}{2^k} = 1 \Rightarrow N = 2^k$

$\log_2$ both sides.

$\log_2 N = \log_2 2^k$

$\log_2 N = k \log_2 2$

$k = \log_2 N$

$i = i/2$

$i = 1$

$i < N$

$\log_c a^b = b \log_c a$

$\log_a a = 1$

Ex:

```
void fun(int N){
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            sysout(===);
        }
    }
}
```

No. of iterations = 10N

$N * N = N^2$

Table :-

| i | j | iterations |
|---|---|---|
| 1 | [1, N] | N iterations |
| 2 | [1, N] | N iterations |
| 3 | [1, N] | N iterations |
| ! | ! | ! |
| 10 | [1, N] | N iterations |

N + N + N + N + — — — 10 times.

⇒ N * 10

```
void fun (int N) {
    for(int i= 1; i<=N; i++) {
        for (int j=1; j<=i; j++) {
            sysout(" == ");
        }
    }
}
```

| i | j | iterations |
|---|---|---|
| 1 | [1, 1] | 1 iteration |
| 2 | [1, 2] | 2 iterations |
| 3 | [2, 3] | 3 iterations |
| 4 | [1, 4] | 4 iterations |
| ⋮ | ⋮ | ⋮ |
| N | [1, N] | N iterations |

$(1 + 2 + 3 + 4 + \cdots\cdots + N) \Rightarrow$ sum of n natural nos.

$\Rightarrow \boxed{\frac{N(N+1)}{2}}$ iterations

```
void fun(int N) {
    for(int i=1; i<=N; i++) {
        for(int j=1; j<=2^i; j++) {
            x = x + i*j
        }
    }
}
```

Table :

| i | j | iterations |
|---|---|---|
| 1 | $[1, 2^1]$ | $2^1$ |
| 2 | $[1, 2^2]$ | $2^2$ |
| 3 | $[1, 2^3]$ | $2^3$ |
| 4 | $[1, 2^4]$ | $2^4$ |
| ⋮ | ⋮ | ⋮ |
| N | $[1, 2^N]$ | $2^N$ |

Sum of N terms :

$$S_N = \frac{a * [r^N - 1]}{r - 1}$$

a = first term

r = common ratio $\Rightarrow \frac{a_{N+1}}{a_N} = \frac{a_N}{a_{N-1}}$

N = no. of terms.

$$S_N = 2^1 + 2^2 + 2^3 + 2^4 + \text{- - - - - -} + 2^N \Rightarrow \text{G.P. series}$$

$$S_N = \frac{2\left[2^N - 1\right]}{2 - 1} = 2\left[2^N - 1\right]$$

$$= \boxed{2^{N+1} - 2}$$

$$r = \frac{2^2}{2^1} = 2$$

# Comparing Algo's using Iterations

↳ Problem Statement | Input | Test Case.

| A | B |
|---|---|
| $100 \log N$ | $N/10$ |

Till $N_c = 13500$ → A > B

$N > 13500$ → A < B

Which algo is optimized

B

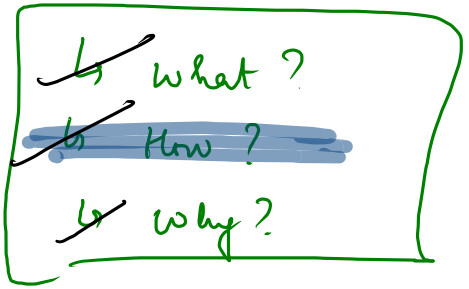
A

→ larger input values

# Asymptotic Analysis

Monitoring performance of algo's for very large inputs

↳ (1) Big O (2) Omega ($\Omega$) (3) Theta ($\Theta$)



Big (O)

10 - 12

Steps to calculate Big O

→ (1) Calculate no of iterations based 'n' inputs.

→ (2) Neglect lower order terms.

→ (3) Neglect constant coefficient terms.

$f(N) = 4N \log N + 3N\sqrt{N} + 10^6 \rightarrow ?$

$\Rightarrow N\sqrt{N}$

$O(N\sqrt{N})$

$f(N) = 3N^2 + 10N + 6 \Rightarrow O(N^2)$

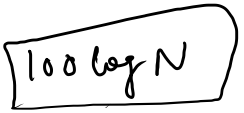
$f(N) = 4N + 3N \log N + 10^6 \rightarrow ? \Rightarrow O(N \log N)$

# Functions comparison

$$\log(N) < \sqrt{N} < N < N\log N < N\sqrt{N} < N^2$$

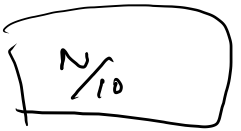
why?

A'





B





Big(O) :

A's less than time in comparison B

Obs: → For larger inputs A's Algo is better

Q:- Why neglect lower order terms ?

Total iter:- $N^2 + 10N$

| Input N | Total iteration : | % contribution of lower order in Total |
|---|---|---|
| 10 | 100 + 100 = 200 | $\frac{100}{200} \times 100\% = 50\%$ |
| 100 | $10^4 + 10^3$ | $\frac{10^3}{10^4+10^3} \times 100\% \simeq 10\%$ |
| $10^4$ | $10^8 + 10^5$ | $\frac{10^5}{10^8+10^5} \times 100\% = 0.1\%$ |

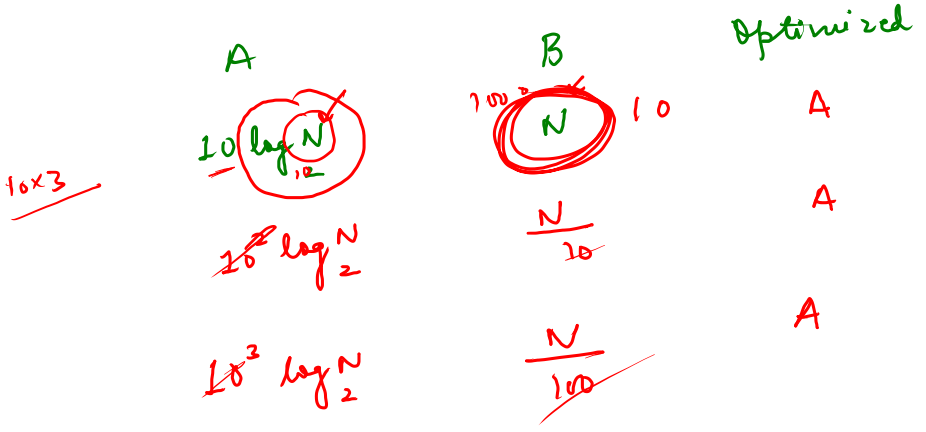
Obs:- If input N increases, contribution of lower order term decreases.

⇒ we neglect lower order terms

Q- Why neglect constant coefficient ?

Problem statement / Input / Test case

1000



1

Issues with Big(O)

| | A | B |
|---|---|---|
| | $100N$ | $N^2$ |
| Big(O) | $O(N)$ | $O(N^2)$ → A is optimized one. |

Count of iterations

| Input N | A | B | Optimized |
|---|---|---|---|
| 10 | $10^3$ | $10^2$ | B |
| 99 | $100*99$ | $99*99$ | B |
| 100 | $100*100$ | $100*100$ | Same |
| 101 | $100*101$ | $101*101$ | A |
| 110 | $100*110$ | $110*110$ | A |

$\forall$ $N > 100$, A is better.

Obs:-

Big O comparison holds only after certain Input values.

(2)

| | A | B | |
|---|---|---|---|
| Total Iteration :- | $10N^2 + 5N$ | $6N^2 + 10N$ → | B has less iteration & optimized |
| Big(O) | $O(N^2)$ | $O(N^2)$ | |

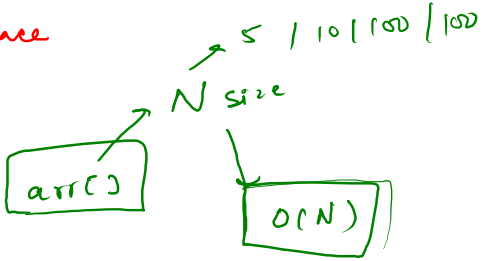
Obs:- If 2 algo's have same Big O, we need iterations to compare.

# Space complexity :-

↳ Amount of extra space used by algorithm.

↳ Space other than input space

↳ Auxiliary space.




e.g. int fun ( int N ) {

int x = 10; → 4B

int y = x * x ; → 4B

}

Total space = 8 B

Space is constant | O(1)

3