Time Complexity - Part II

Mar 25, 2022

AGENDA :

* Comparing 2 algorithms ... ?
* Asymptotic Analysis : Big O
  - Issues in Big O
  - Graphical representation of Big O
* Worst case, Average Case, Best case
* Space complexity

Recap:

1.

```
s=0
for (i) in range(0,N):          2N
    for j in range(0,N):        N.
        s+=1
```

N * N
= N^2 iterations

2.

```
for i in range(0,N):                    N²
    for j in range(0,N):                → N²
        print("Hello")
for k in range(0,N):                    N
    print("Hello")                      → N
```

log N

No. of iterations = $N^2 + N$

For a given input (N) to a program,
the no. of iterations in the program
= I(N) ← some function of N

finding the time complexity.

I(N) = $N^2 + N$: function of N

e.g. $N^2 + 5N + \log N$ function of N

$5N^2 + N \log N$

Comparison of 2 Algorithms.

① Time
→ ② Space — } ← These are the 2 constraints.

Comparing 2 algorithms on basis of Time.

① Runtime / Execution Time

Soumya

Deepak

Task: You are given an array of size N.
{ 2, 1, 5, -1, 0, 7 } ←
Sort this array in ascending order.
Output: { -1, 0, 1, 2, 5, 7 } ←

SortX
12 sec

Windows 95 laptop
↓
Macbook Pro

8 sec
SORTX is the winner!

C++
↓
Python

13 sec

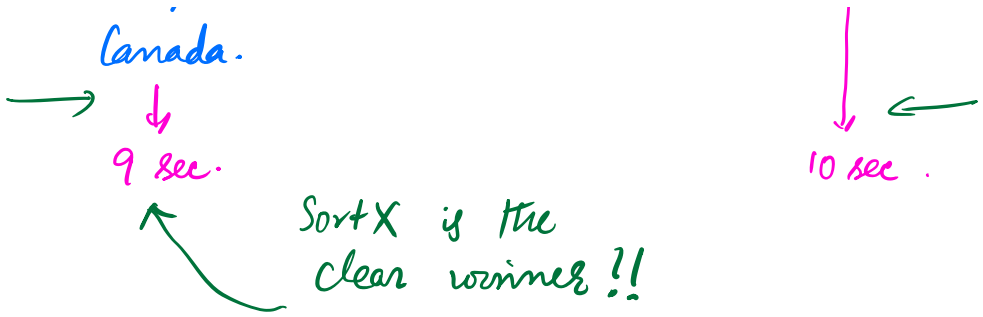
↓
Sitting on a volcano
↓

SORT-RE
10 sec
SORT-RE is the winner!
Macbook Pro

10 sec

Python.

10 sec
SORT-RE is the winner.
Canada



Observations:

1. Dependencies:
   i) Hardware / Processor.
   ii) Programming Language
   iii) Temperature (Environmental conditions)

   Execution Time (Run Time) of your code can be affected by the above.

2. **Execution Time** is not a good criteria to analyse time taken by 2 algorithms.

3. No. of iterations
   ↳ is constant.
   independent of hardware, language or environment.

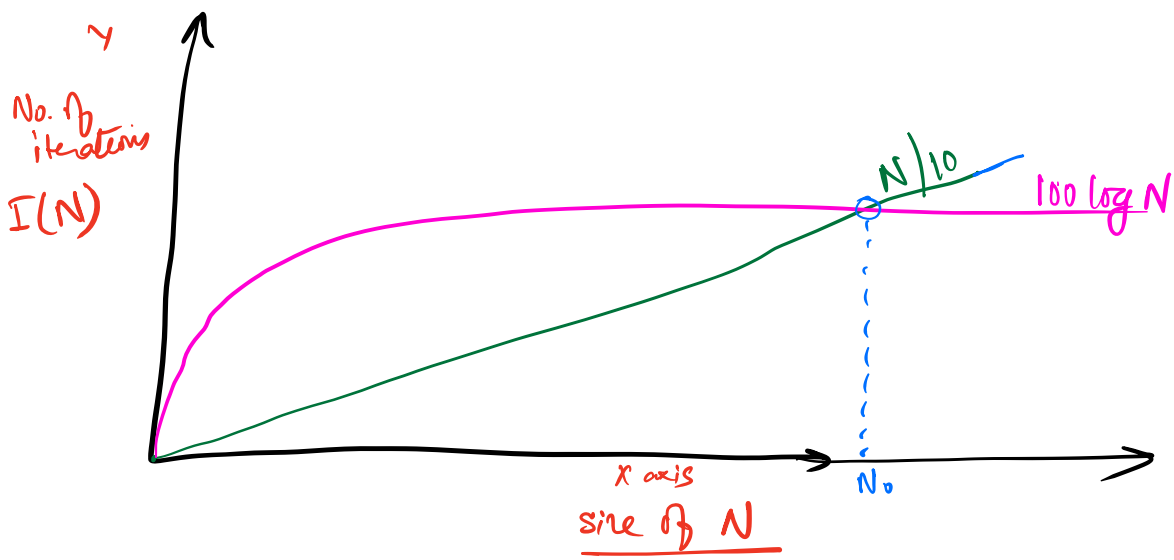
```
for i in range (0,N):
    ≡
```

if N increases,
⇒ no. of iterations increase.

also, time taken by code increases.

∴ Time taken $\propto$ $\underbrace{I(N)}_{\text{No. of iterations in terms of } N}$

	Soumya.	Deepak.
	$100 \log N$ iterations	$N/10$ iterations
$N = 2$	$100 \log_2 2$ $= 100$ iterations	0 iterations
$N = 1,000,000$	Lower $100 \log 1,000,000$ $= 19.9$	Higher. $1,00,000$



$N \geq N_0$: $N/10 > 100 \log N$

⇓

Soumya will win

$N < N_0$: $N/10 < 100 \log N$

⇓ ⇓

Deepak will win

$N_0 = 3550.$

** Algorithm that performs well on Large Numbers is better.

↳ Award goes to Soumya!

: Infinity ∞

When comparing 2 codes, always consider that $N \to \infty$.

for . . .

$N \to \infty$.

$N = 100$

$N = 10^{10}$

** Asymptotic Analysis.

→ analysing the code for $N \to \infty$.

~~$N = 10, N = 1$~~ X

** Asymptote

GOOGLE!

Final conclusion

** $100 \log N$ is better than $N/10$

→ $I_1(N)$ → $100 \log N$

$I_2(N)$ → $2N^2 + N$

$I_3(N)$ → $N/10$

. . . . → $\log N$

$I_{23}(N)$ → $N^4 + 2^N$

Arrange these algorithms.

in order of their performance.

I(N) is not standard.

* Come up with some standard practice!

↓ Big O. : standard we will be using to compare

BREAK! 10:10 pm

Big O Notation

1. Find the no. of iterations in your code.
   ↓ I(N)
2. Ignore the lower order terms.
3. Ignore the constant coefficients.

* What are lower order terms?

$I(N) = N^2 + N$ times

N : Lower order term.

$I(N) = N + \log N$ times (ignored)

$\log N < \sqrt{N} < N < N \log N < N\sqrt{N} < N^2 < N^3 < 2^N < N!$

How to come up with this?

$N \geq 2$.

$N = 4$.

2 2 4 8 8 16 64 16

24

Don't care about $N = 4$.

$N = 100$.

$N \to \infty$

Check for $N = 100, 1000, 10^{10}$

$N \to \infty$

* Why can you ignore lower order terms?

$$I(N) = \underbrace{N^2}_{\text{square}} + \underbrace{N}_{\text{linear}}$$

$N = 1$

$I(N) = 2$

→ square = 1 — 50%

→ linear = 1 — 50%

$N = 10$.

$I(N) = 110$

square = 100 → 91%

linear = 10 → 9%

$N = 100$

$I(N) = 10100$

square = 10000 — 99%

linear = 100 — 1%

$N = 10^8$

$I(N) = 10^{16} + 10^8$

square $= 10^{16}$ → 99.999999%

linear $= 10^8$ → 0.000001%

$I(N) = N^2 + N$

$\approx N^2$

(3) Ignore constant coefficients.

$$I(N) = 5N^2 + N$$

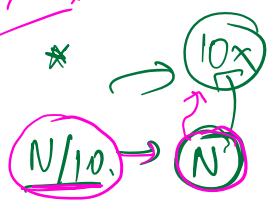
$$= 5N^2$$

$$= N^2$$

Why?

$$\begin{cases} I_1(N) = 100 \log N \\ I_2(N) = N/10 \end{cases}$$

$\log N > N$.



* As $N \to \infty$, 100 or 1/10 doesn't matter!

$\log N$ and N

* $I(N) = 4N^2 + 3N + 10$

$O(I(N)) = O(N^2)$

*

N^2

$N(\log N)$ >1

N

$\log N$.

30-40%

$N \log N$ $\quad N * \log N > \log N$

* $4N \log N + 3N \sqrt{N} + 10^6$

$\sim N\sqrt{N}$

$\sim N \log N$

$\sqrt{N} > \log N$ → ignore

$N\sqrt{N}$

* $2N^2 + 5 \cdot 2^N + \log N$

$= O(2^n)$

* $I(N) = N^2 + N$ → ignored

$O(N) = O(N^2)$

*
```
print(N) ←                          range(0,N) ✗
for i in range(0,10):
    for j in range(0,10):
        print "Hello"
```

$I(N) = 10 * 10$

$= 100$

if $N \to \infty$, $I = 100$

if $N = 0$, $I = 100$ → Time taken by code remains

if $N = 10^5$, $I = 100$ same.

$I(N) = 100$

3 steps.

2. Ignore the lower order terms. ✓
3. Remove the constants.

$I(N) = 100$

$= 100 \cdot N^0$

$= N^0$

$5N^2 + 2N + 6$

$= 5N^2 + 2N^1 + 6 \cdot N^0$

$O(I(N)) = O(N^0)$

$= O(1)$ ← constant time complexity

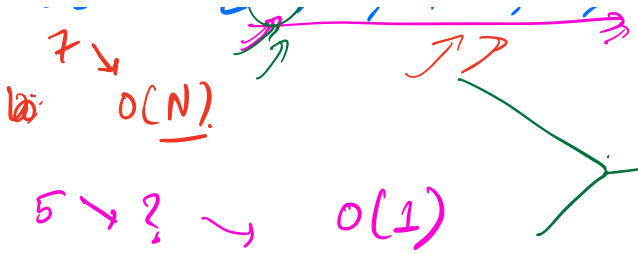
Special Names.

- $O(1)$ → constant
- $O(\log N)$ → logarithmic
- $O(N)$ → linear
- → $O(N \log N)$ → linear logarithmic // This is the best to sort.
- $O(N^2)$ → Quadratic
- $O(N^t)$ → Polynomial. $t \geq 1$ $t \geq 2$ $O(N^3)$ $O(N^4)$
- $O(2^N)$ → Exponential.
- $O(N!)$ → factorial

Worst case, Best case, Average case.

Search.

List = [5 2 1 -1 0 7]



O(1) → Best case time complexity

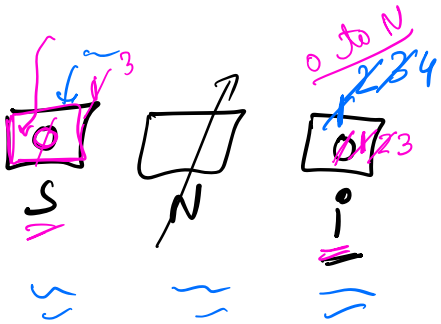
Best case / Worst case → Depends on your input.

Space Complexity

: O(N).

: Denotes memory taken up by your program.

```
s=0
for i in range(0,N):
    s=s+i
```



Time complexity = O(N)

Space complexity = O(1)

Memory consum

S(N) = 3 * k

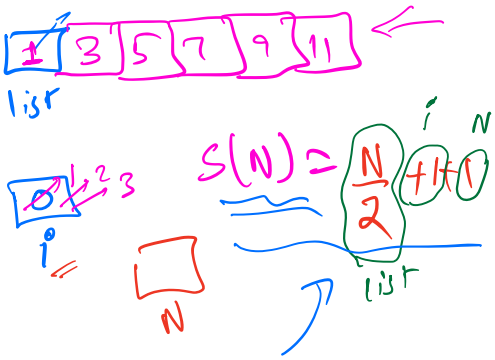
O(S(N) = O(1)

// Return all odd nos. b/w 1 to N.

```
→ list = []
  for i in range(1,N+1):
    - if i%2!=0:
        list.append(i)
  return list
```

Time complexity = O(N)

Space complexity = O(N)



As N increases. list size increases.

↑ memory consumption increases.

$$S(N) = \frac{N}{2} + 1 + 1$$

$$O(S(N)) = \frac{N}{2} + 2 \cdot N \quad \text{(ignore } 2 \cdot N\text{)} \quad \text{(ignore } \tfrac{N}{2}\text{)}$$

$$= O(N)$$

Issues in Big O -

(1) $I(N) = N/10$ → $O(N)$

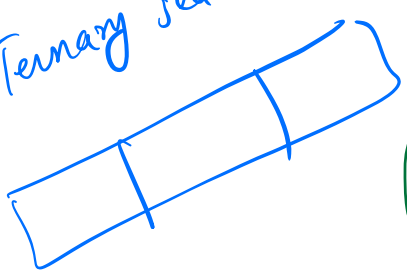
$I(N) = 15N$ → $O(N)$

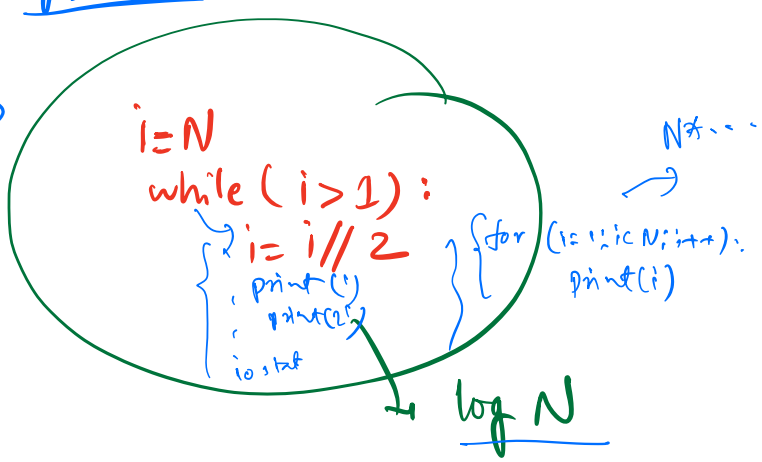
Using Big O Notation → Both these are same

(2)

Binary search → $\log_2 N$

Ternary search → $\log_3 N$.





$$I(N) = N^8 + N^7 + N^6 + N^5$$

$N^5 \ll N^6$

$N^6 << N^7$

$N^7 << N^8$

$N^5 + N^6 + N^7 << N^8$

(10) ←

No. of operations is dependent on N

$I(N) = 100 \leftarrow$ No. of iterations

$= 100 \cdot N^0$ (1)

$N^0 = 1$

N_0 (x) $+ N_1 N_2 N_3$

$O(N^0) = O(1) \leftarrow$ constant time

$O(100) = O(1)$

O

$I(N)$

```
if (condition):
    for
        for        ← O(N^2)
else
    for            ← O(N)
```

$O(N^2)$

N times.

```
for i in range(0,N):
    j = N
    while(j>1)
        j = j // 2
```

↳ Log N

↓

N Log N.

* range(0,N) → [0, 1, 2, 3, 4, 5. . . . N-1]

// Generator in Python.

i = 0 ←

i = 1

Golang .
C++
Python