Analysis of algorithms

The field of computer science, which studies the efficiency of algorithms is known as analysis of algorithm.

Effeciency of the program depends on many parameters -

- System Level
- Software Level
- compiler Level, ... etc.

Prof. knuth -

Complexity of an algorithm is nothing but the resources required for the execution of an algorithm.

Mostly we use * Space → Running Time (physically measur

* Time → Time complexity

Time complexity →

* A mathematical quantity & it is going to give you an idea on running time.

Time complexity is something that we would like to have even before writting the Program

Analysis of Algorithm →

The mathematical estimation of time complexity & it is not an emperical study.

We need a mathematical way for obtaining the Time complexity.

How to do ...

1) The time complexity can be estimated through the instruction count involved in the algorithm.

Instruction count of which input?

Ex- Sorting.

Instruction count of -

→ 10 (or) nos.

→ 20 (or) nos.

.
.
.

→ 1000 (or) nos

.
.
.

2) Input Size → Measures the quantum of input processed.

Input Size ⟷ Instruction count.

$n \longleftrightarrow f(n)$

Ex

Algorithm 1

$1000 \cdot n$

Algorithm 2

$n \cdot \log_2 n$

which one is better ?

Asymptotic Analysis

Study the behaviour of the function as $n \to \infty$

As n becomes larger & larger how the algorithms are behaving !

When will $n \log n$ overtakes $1000n$?

$$n \to \infty \quad \frac{n \log n}{1000 \cdot n}$$

$$\log_2 n = 1000$$

$$n = 2^{1000}$$

$n \to f(n)$ — first set of n inputs, second set of n inputs

$I_n = \{ i_1, i_2, i_3, \ldots, i_t \}$

t possible inputs of size n.

$Ic(i_1)$, $Ic(i_2)$ $Ic(i_t)$

Number of instructions taken to process first set of ilps by the algorithm.

There could be variation in instruction count among various inputs of same size, so which one should I take?

Here comes the notation of worst-case Time complexity.

$\text{Max}\{ Ic(i_1), Ic(i_2), \ldots, Ic(i_t) \}$

There is another notation of Average-case complexity.

$$\frac{(Ic(i_1) + Ic(i_2) + \ldots + Ic(i_t))}{t}$$

Avg1

Seq 1 → 99, 98, 98, 99, ..., 100

IC(1), IC(2)

Avg2

Seq 2 → 3, 4, 3, 5, 6, 3, 3, ..., 100

Avg(Seq1) ≥ 98

Avg(Seq2) ≥ 5

Finally which instructions in the algorithm considered ?

(1) Perform the instruction count of every task.

(2) Usually it is sufficient to consider the dominant operations (costly operations) & just count them only.

Ex $C_{n \times n} = A_{n \times m} \times B_{m \times n}$

$$C_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$$

would you count both addition & multiplication (or) only one of them?

If you count running time →

i_1	i_2	. . .	i_n
α_1	α_2	. . .	α_n
t_1	t_2		t_n

$\alpha_1 t_1 + \alpha_2 t_2 + \ldots + \alpha_n t_n$.

Summary

- Before even going to the first stroke of the keyboard I must have an idea on how good is the method that I have in my mind.

Time complexity gives you insights

→ Time complexity of ... on the running time. It will not give you the exact value of running time.

→ we are interested in finding out the functional relation which is relating the instruction count with the size of input.

$$n \longrightarrow f(n)$$

→ Worst-case complexity and average case complexity are the two different cases in Time-complexity. (One more is there, which is Best-case complexity - opposite to the worst case complexity).

→ Usually the dominant operations in the algorithm will be counted in Time complexity.