Analysis and Design of Algorithms

Asymptotic Notations- Rate of Growth

Contents

- Asymptotic Notation- Rate of Growth
- O-Notation (Big-Oh Notation)
- Θ-Notation (Theta Notation)
- Ω-Notation (Omega Notation)

Algorithms- Growth functions or Rate of Growth

- The **time** required to solve a problem depends on the **number of steps** it uses.

- **Growth functions or Rate of Growth** are used to **estimate the number of steps an algorithm uses as its input grows**.

Asymptotic Notation- Rate of Growth

- Algorithms can be evaluated by a **variety of criteria.**
- Most often we shall be interested in the **rate of growth of the time or space required to solve larger and larger instances of a problem.**
- We will associate with the problem an integer, called the **size of the problem**, which is a measure of **the quantity of input data**.
- Goal: To simplify analysis by getting rid of unneeded information (like “rounding” 1,000,001≈1,000,000)

Types of Asymptotic Notation

1. O(Big - Oh) – Notation
2. o (Little - Oh) - Notation
3. Ω(Big - Omega) – Notation
4. ω (Little - Omega) - Notation
5. Θ(Theta) – Notation

1. O (Big - Oh) – Notation

The Big-Oh notation defines an upper bound of an algorithm, it bounds a function only from above.

The Big-Oh Notation can be used in the following instances:

- For expressing the upper bound or the worst-case complexity of an algorithm.
- For expressing that "time complexity is never more than" or "at most" the given complexity function.

For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O(n^2). Note that O(n^2) also covers linear time.

It would be convenient to have a form of asymptotic notation that means "***the running time grows at most this much, but it could grow more slowly.***" We use "big-Oh" notation for just such occasions.

O(Big – Oh) – Notation

- The "Big-Oh" Notation:
  - Given functions $f(n)$ and $g(n)$,
  - we say that $f(n)$ is $O(g(n))$
  - if and only if there are
    1. positive constant c and
    2. positive constant n_0

    such that $f(n) \le c.g(n)$

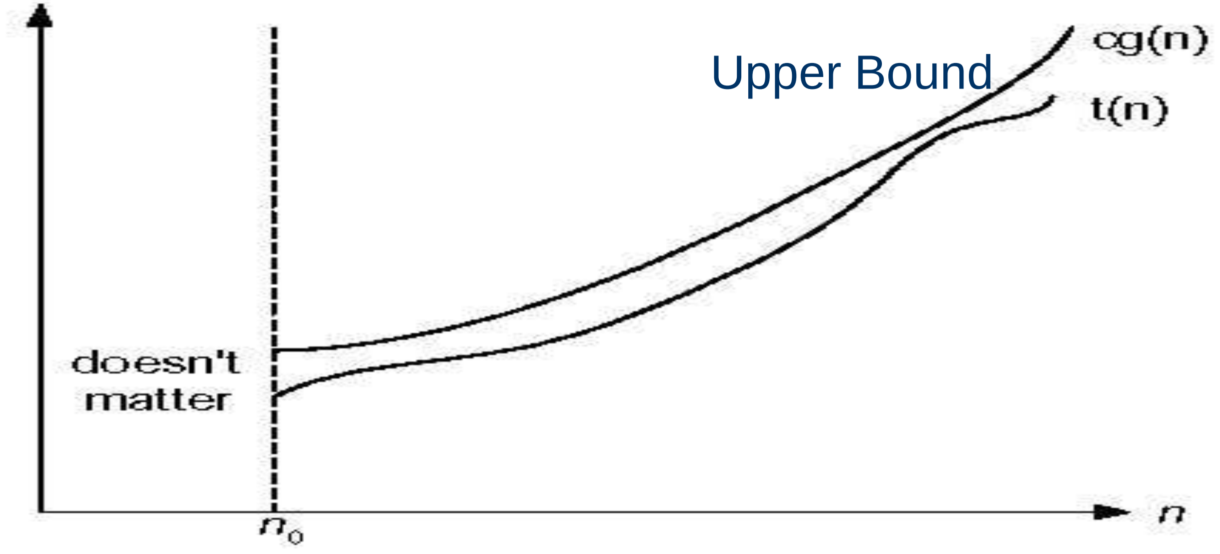
    for $n \ge n_0$

O(Big – Oh) – Notation

- **f(n) = O(g(*n*))** (read as “f of n is big-oh of g of n”)

$$f(n) \le c * g(n)$$

- iff there exist positive **constants c** and
- **n_0 such that for all n, $n \ge n_0$**



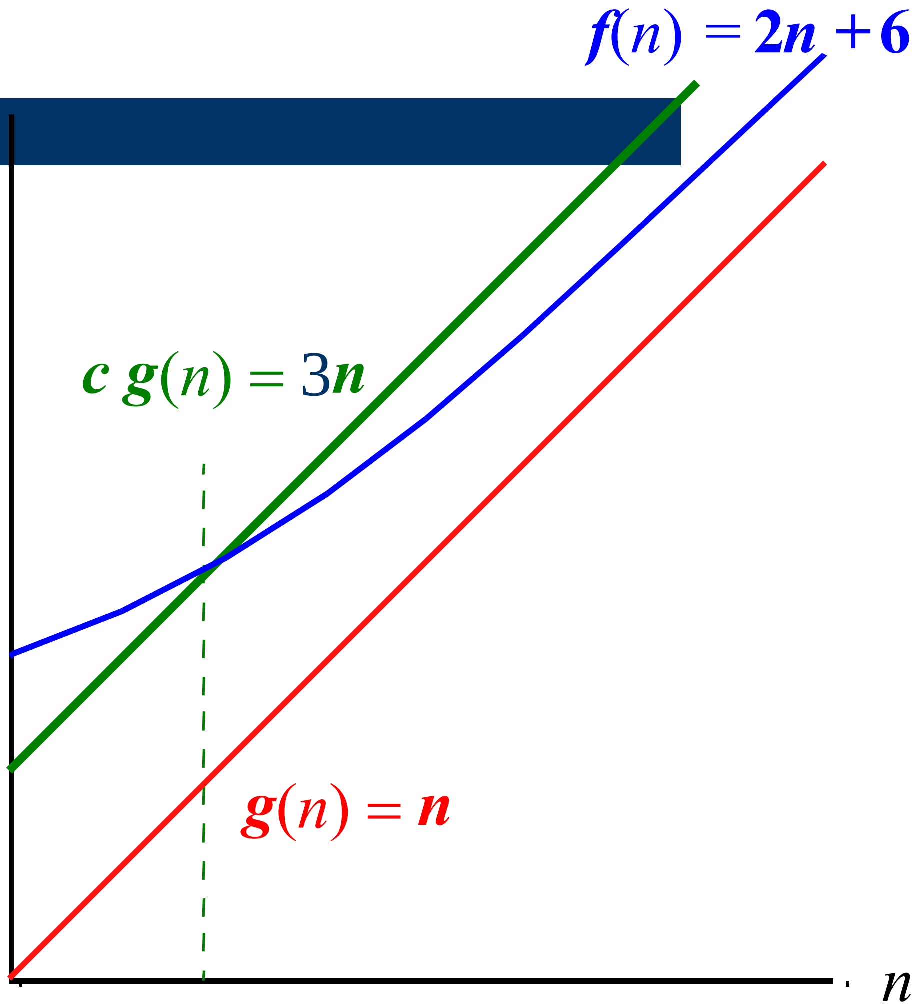
We say that running time is "Big-Oh of f(n)" or just "O of f(n)". We use Big-Oh notation for ***Asymptotic upper bounds,*** since it bounds the growth of the running time from the above for large enough input sizes.

The general step wise procedure for Big-Oh runtime analysis is as follows:

- Figure out what the input is and what 'n' represents.
- Express the maximum number of operations, the algorithm performs in terms of 'n'.
- Eliminate all excluding the highest order terms.
- Remove all the constant factors.

O(Big – Oh) -- Graphic Illustration

- f(n) = 2n+6
  - Need to find a function g(n) and a const. c such as f(n) < c.g(n)
- g(n) = n and c = 3
- f(n) is O(n)
- The order of f(n) is n.

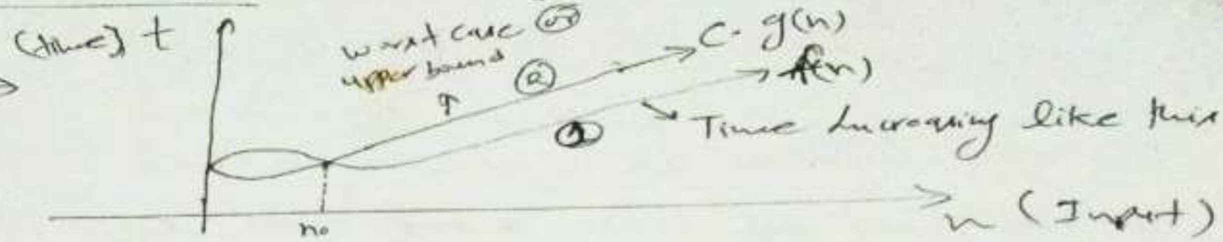


Examples of Big – Oh

Example-1

Asymptotic Notation (Terminology used in Algo.)

① Big oh (O) →



Equation → $f(n) \le C \cdot g(n)$

Conditions → $n \ge n_0$

$C > 0, \quad n_0 \ge 1$

$$\boxed{f(n) = O(g(n))}$$

means → f(n) is smaller than g(n).

Example → suppose $f(n) = 3n+2$ — ①

$g(n) = n$ — ②

To prove $f(n) = O(g(n))$ we need to know c & n_0.

Then we have to follow → $f(n) \le C \cdot g(n)$ for some $C > 0$ & $n_0 \ge 1$

Put equation → $3n+2 \le C \cdot n$ take $C = 4$

$3n+2 \le 4 \cdot n$

$2 \le 4n - 3n$

$\boxed{2 \le n}$

So we can get $C = 4$ & $n \ge 2$

$f(n) = 3n+2$ & $g(n) = n$ → least upperbound or Tightest Bound

Example-2

Eg:- $f(n) = 5n+7 = O(g(n))$

$5n$

for Big-oh notation,

$f(n) \leq c * g(n)$

where c is a constant

$f(n)$ is given function

$g(n)$ is Result function.

$\Rightarrow 5n+7 \leq c * g(n)$

c = (Coeffecient of greatest degree + 1)

$\therefore c = 5+1 = 6$

$\Rightarrow 5n+7 \leq 6 * g(n)$

Let $g(n) = 1$

$5n+7 \leq 6$ $\Rightarrow$ which is false

$g(n) = 2$

$5n+7 \leq 12$ $\Rightarrow$ which is false

$\vdots$

$g(n) = n$

$5n+7 \leq 6n$

Let $n=1$:

$5+7 \leq 6$ $\Rightarrow$ which is false

Let $n=2$:

$10+7 \leq 12$ $\Rightarrow$ which is false

Let $n=3$:

$15+7 \leq 18$ $\rightarrow$ which is false

Let $n=4$:

$20+7 \leq 24$ $\rightarrow$ which is false

Let $n=5$:

$25+7 \le 30$ $\Rightarrow$ which is false

Let $n=6$:

$30+7 \le 36$ $\Rightarrow$ which is false

Let $\boxed{n=7}$:

$35+7 \le 42$ $\Rightarrow$ which is True

Let $n=8$

$40+7 \le 48$ $\Rightarrow$ which is True

$\vdots$

from $n=7$, Equation satisfies where $g(n)=n$.

$\therefore f(n)=O(g(n))$

$f(n)=O(n)$

Here, $c=6$, $n_0=7$

$\therefore f(n)=O(n)$.

Example-3

Eg: $f(n) = 3n^3 + 2n + 7$

$3n^3 + 2n + 7 = O(g(n))$

for Big-oh notation,

$f(n) \leq c * g(n)$

where $f(n)$ is given function

. c is constant where

$c =$ (Coeffecient of higher degree +1)

$c = 3 + 1 = 4$

$f(n) \leq c * g(n)$

$\Rightarrow 3n^3 + 2n + 7 \leq 4 * g(n)$

let $g(n) = 1$

$3n^3 + 2n + 7 \leq 4 * 1$ => false

$g(n) = 2$

$3n^3 + 2n + 7 \leq 8$ => false

⋮

$g(n) = n$

$3n^3 + 2n + 7 \leq 4n$

let $n = 1$:

$3 + 2 + 7 \leq 4$ => false

let $n = 2$:

$24 + 4 + 7 \leq 8$ => false

let $n = 3$:

$81 + 6 + 7 \leq \ldots 12$ => false

let $n=4$:

$192+8+7 \le 16 \rightarrow$ false

let $n=5$:

$375+10+7 \le 20 \rightarrow$ false

$\vdots$

false

let $g(n)=n^2$

$3n^3+2n+7 \le 4n^2$

$n=1$:

$3+2+7 \le 4 \rightarrow$ false

$n=2$:

$24+4+7 \le 16 \rightarrow$ false

$n=3$:

$81+\cancel{6}+21 \le 36 \rightarrow$ false

$n=4$:

$192+8+7 \le 64 \rightarrow$ false

$n=5$:

$375+10+7 \le 100 \rightarrow$ false

$\vdots$

$\rightarrow$ false

Let $g(n) = n^3$

$$3n^3 + 2n + 7 \le 4n^3$$

$n=1$:

$3 + 2 + 7 \le 4$ $\Rightarrow$ false

$n=2$:

$24 + 4 + 7 \le 32$ $\Rightarrow$ false

$\boxed{n=3}$

$81 + 6 + 21 \le 108$ $\Rightarrow$ True

$n=4$:

$192 + 8 + 7 \le 256$ $\Rightarrow$ True

$n=5$:

$375 + 10 + 7 \le 500$ $\Rightarrow$ True

from $n=3$, i.e. Equation satisfies where $g(n) = n^3$.

Here, $n_0 = 3$, $c = 4$

$\therefore f(n) = O(g(n))$

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

$\therefore$ Big-oh notation of $f(n) = O(n^3)$

“Relatives” of Big-Oh

- “Relatives” of the Big-Oh
  - Ω (f(n)): Big Omega – asymptotic *lower* bound
  - Θ (f(n)): Big Theta – asymptotic *tight* bound
- Big-Omega – think of it as the inverse of O(n)
  - g(n) is Ω (f(n)) if f(n) is O(g(n))
- Big-Theta – combine both Big-Oh and Big-Omega
  - f(n) is Θ (g(n)) if f(n) is O(g(n)) and g(n) is Ω (f(n))