9.A.1 BIG O NOTATION – EXAMPLES

Example 1

Let $f(n) = 3n^3$ for an algorithm. Prove that $f(n)$ of the algorithm is in $O(n^3)$.

Solution

The definition of the Big -Oh notation is that $f(n) \leq c \times g(n)$.

In order to prove that we know:

$f(n) \in O(g(n))$

or, $f(n) = O(g(n))$

Where $g(n)$ is in n^3.

Hence, we can show that:

$3n^3 \leq c \times n^3$, holds good for a positive number c and sufficiently large values of n.

$f(n) = 3n^3$ *can also be written as*:

$f(n) = 3n^3 + 0$

We can write it as:

$3n^3 + 0 \leq 3n^3 + n^3$

$\approx 3n^3 + 0 \leq 4n^3$ [Note $3n^3 = 3n^3$ *but* $3n^3$*is always* $\leq 4n^3$]

[As highest degree of polynomial n *is* 3]

$\approx 0 \leq 4n^3 - 3n^3$

$\approx 0 \leq n^3$

Or, Divide n^2 in both side:

$\approx \frac{\mathbf{0}}{n^2} \leq \frac{n^3}{n^2}$

$\approx \mathbf{0} \leq n$

or, $n \geq \mathbf{0}$

Therefore $n_0 = \mathbf{0}$.

We can again write it as:

$3n^3 + 0 \leq 4 \times n^3$

Hence $c \geq 4$.

Therefore, $f = O(g)$

or in other words, the algorithm is $O(n^3)$.

Hence proved.

Example 2

Let $f(n) = 3n + 8$ for an algorithm. Prove that $f(n)$ of the algorithm is in $O(n)$.

Solution

The definition of the Big -Oh notation is that $f(n) \leq c \times g(n)$.

In order to prove that we know:

$f(n) \in O\big(g(n)\big)$

or, $f(n) = O\big(g(n)\big)$

Where $g(n)$ is in n.

Hence, we can show that:

$3n + 8 \leq c \ \times n$, holds good for a positive number c and sufficiently large values of n.

$\boldsymbol{3n + 8 \leq 3n + n}$

$\boldsymbol{\approx 3n + 8 \leq 4n\ [\ 3n\ is\ always\ \ \leq 4n]}$

[As highest degree of polynomial $\boldsymbol{n\ is\ 1}$]

$\boldsymbol{\therefore 3n + 8 \leq 4n}$

$\boldsymbol{\approx 8 \leq 4n - 3n}$

$\boldsymbol{\approx 8 \leq n}$

$\boldsymbol{or, n \geq 8}$

$$\boxed{\therefore n_0 = 8}$$

$\boldsymbol{3n + 8 \leq 4n}$

Can be written as:

$\boldsymbol{3n + 8 \leq 4\ \times n}$

$\boldsymbol{Hence\ , c \geq 4\ .}$

Therefore, f is O(g)

or in other words $\boldsymbol{O}(\boldsymbol{n})$.

Hence proved.

Example 3

Let $\boldsymbol{f(n) = n^2 + 1}$ for an algorithm. Prove that $\boldsymbol{f(n)}$ of the algorithm is in $\boldsymbol{O(n^2)}$.

Solution

The definition of the Big -Oh notation is that $\boldsymbol{f(n) \leq c \times g(n)}$.

In order to prove that we know:

$\boldsymbol{f(n) \in O(g(n))}$

or, $\boldsymbol{f(n) = O(g(n))}$

Where $\boldsymbol{g(n)}$ is in $\boldsymbol{n^2}$.

Hence, we can show that:

$\boldsymbol{n^2 + 1 \leq c \times n^2}$, holds good for a positive number $\boldsymbol{c}$ and sufficiently large values of $\boldsymbol{n}$.

$\boldsymbol{n^2 + 1 \leq n^2 + n^2}$

$\boldsymbol{\approx n^2 + 1 \leq 2n^2 [\, n^2 \; is \; always \; \leq 2n^2]}$

[As highest degree of polynomial $\boldsymbol{n \; is \; 2}$]

$\boldsymbol{\therefore n^2 + 1 \leq 2n^2}$

$\boldsymbol{\approx 1 \leq 2n^2 - n^2}$

$\boldsymbol{\approx 1 \leq n^2}$

$\boldsymbol{\approx -n^2 + 1 \leq 0}$

$\boldsymbol{\approx -(n^2 + 1) \leq 0}$

$\boldsymbol{\approx -(n^2 - (-1)^2) \leq 0}$

$\boldsymbol{As \; we \; know: x^2 - y^2 = (x + y)(x - y)}$

$\boldsymbol{\approx -1 \times (\; (n + (-1)) \;\; (n - (-1)) \;\;) \leq 0}$

$\approx \boldsymbol{-1} \times (\ \ (\boldsymbol{n-1})\ \ \ (\boldsymbol{n+1})\ \ \ ) \leq \boldsymbol{0}$

$\approx (\boldsymbol{n-1})(\boldsymbol{n+1}) \leq \dfrac{\boldsymbol{0}}{\boldsymbol{-1}}$

$\approx (\boldsymbol{n-1})(\boldsymbol{n+1}) \leq \boldsymbol{0}$

[From Quadratic Inequalities]

$$Say\, n = 1$$

$\approx (1-1) \times (1+1) \leq 0$

$\approx 0 \times 2 \leq 0$

$\approx 0 \leq 0[True]$

Also,

$\approx 0 \geq 0[True]$

$$Say\, n = 2$$

$\approx (2-1) \times (2+1) \leq 0$

$\approx 1 \times 3 \leq 0$

$\approx 3 \leq 0[False]$

But,

$\approx 3 \geq 0[True]$

Hence, we can say $n \geq 1$

$$Say\ n = 0$$

$\approx (0-1)\times(0+1)\leq 0$

$\approx -1\times 1\leq 0$

$\approx -1\leq 0[True]$

$$Say\ n = -1$$

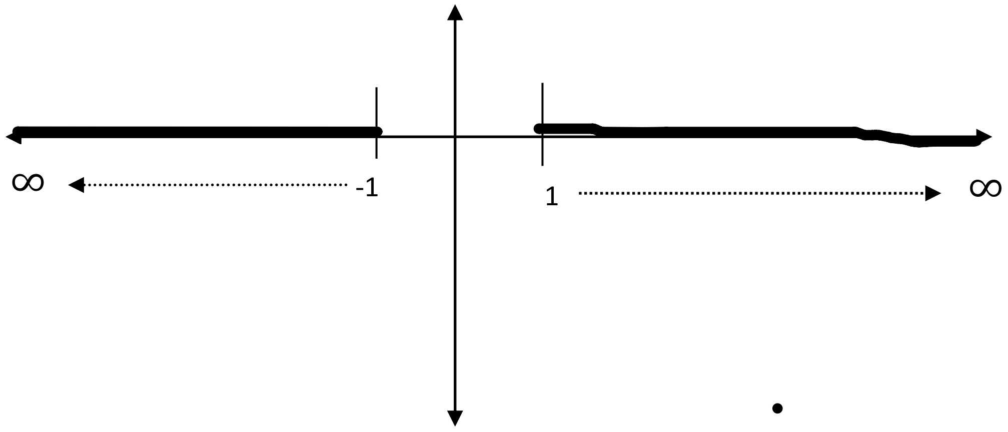
$\approx (-1-1)\times(-1+1)\leq 0$

$\approx -2\times 0\leq 0$

$\approx 0\leq 0[True]$

That implies $n\leq -1$

$$\therefore \boldsymbol{-1\geq n\geq 1}$$



$(-\infty,-1]\cup[1,\infty)\ and\ they\ are\ in\ Sem - open\ and\ semi\ closed\ intervals.$

Hence now we get:

$$n^4 + 1 \leq 2n^2 \ for \ all \ n \geq 1$$

As from the definition of Big – O notation, all should be positive numbers but not negative numbers.

Therefore , we get $n_0 = 1$.

And,

$n^4 + 1 \leq 2 \times n^2 \ in \ terms \ of \ \boldsymbol{f(n) \leq c \times g(n)}$

Hence:

$\boldsymbol{c \geq 2}$ **.**

Therefore, f is O(g)

or in other words $\boldsymbol{O(n^2)}$.

Hence proved.

Example 4

Let $\boldsymbol{f(n) = n^4 + 100n^2 + 50}$ for an algorithm. Prove that $\boldsymbol{f(n)}$ of the algorithm is in $\boldsymbol{O(n^4)}$.

Solution

The definition of the Big -Oh notation is that $\boldsymbol{f(n) \leq c \times g(n)}$.

In order to prove that we know:

$\boldsymbol{f(n) \in O\big(g(n)\big)}$

or, $\boldsymbol{f(n) = O\big(g(n)\big)}$

Where $\boldsymbol{g(n)}$ is in $\boldsymbol{n^4}$.

Hence, we can show that:

$\boldsymbol{n^4 + 100n^2 + 50 \leq c \times n^4}$

$\boldsymbol{\approx n^4 + 100n^2 + 50 \leq n^4 + n^4}$

$\boldsymbol{\approx n^4 + 100n^2 + 50 \leq 2n^4\ [\,n^4\ is\ always \leq 2n^4]}$

[As highest degree of polynomial $\boldsymbol{n\ is\ 4}$]

$\boldsymbol{Now,}$

$\boldsymbol{\approx -2n^4 + n^4 + 100n^2 + 50 \leq 0}$

$\boldsymbol{\approx -n^4 + 100n^2 + 50 \leq 0}$

We can write the above equation as:

$\boldsymbol{\approx -n^4 + 100n^2 + 50 = 0\ and -n^4 + 100n^2 + 50 < 0}$

Taking the equation:

$\boldsymbol{\approx -n^4 + 100n^2 + 50 = 0}$

Rewriting the equation, $\boldsymbol{u = n^2\ and\ u^2 = n^4}$:

$\boldsymbol{\approx -u^2 + 100u + 50 = 0}$

Solving with quadratic equation formula:

Quadratic Equation of the form $\boldsymbol{ax^2 + bx + c = 0}$:

$$\boldsymbol{x_{1,2} = \frac{\left(-b \pm \sqrt{b^2 - 4ac}\,\right)}{2a}}$$

For a $=-1$, b $= 100$, c $= 50$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{100^2 - 4(-1)(50)}\,\right)}{2(-1)}}$$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{100^2 - 4(-1)(50)}\right)}{-2}}$$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{100^2 + 200}\right)}{-2}}$$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{10000 + 200}\right)}{-2}}$$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{10200}\right)}{-2}}$$

$\boldsymbol{Using\ prime\ factorization\ of\ 10200 = 2^3 \times 3 \times 5^2 \times 17}$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{2^3 \times 3 \times 5^2 \times 17}\right)}{-2}}$$

Applying exponent rule: $\boldsymbol{a^{b+c} = a^b \times a^c}$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{2^2 \times 2 \times 3 \times 5^2 \times 17}\right)}{-2}}$$

Applying radical rule: $\boldsymbol{\sqrt[n]{ab} = \sqrt[n]{a} \times \sqrt[n]{b}}$

$$\boldsymbol{u_{1,2} = \frac{\left(-100 \pm \sqrt{2^2} \times \sqrt{5^2} \times \sqrt{2 \times 3 \times 17}\right)}{-2}}$$

Applying radical rule: $\sqrt[n]{a^n} = a$

$$u_{1,2} = \frac{(-100 \pm 2 \times 5 \times \sqrt{2 \times 3 \times 17})}{-2}$$

$$u_{1,2} = \frac{(-100 \pm 10 \times \sqrt{2 \times 3 \times 17})}{-2}$$

$$u_{1,2} = \frac{(-100 \pm 10\sqrt{102})}{-2}$$

$$u = \frac{(-100 + 10\sqrt{102})}{-2}$$

$$= \frac{10(-10 + \sqrt{102})}{-2}$$

$$= -5(-10 + \sqrt{102}) - -i$$

$$u = \frac{(-100 - 10\sqrt{102})}{-2}$$

$$= \frac{-10(10 + \sqrt{102})}{-2}$$

$$= 5(10 + \sqrt{102}) - -ii$$

Substituting back $u = n^2$ and solving for n,

$n^2 = -5(-10 + \sqrt{102})$

$(g(x))^2$ *cannot be negative for* $x \in R$ *, hence no solution.*

$n^2 = 5(10 + \sqrt{102})$

We know $(g(x))^2 = f(a)$ *the solutions are* $\sqrt{f(a)}$, $-\sqrt{f(a)}$

$\therefore n = \sqrt{5(10 + \sqrt{102})}$ **and** $n = -\sqrt{5(10 + \sqrt{102})}$

Now we can easily understand,

$\approx -\boldsymbol{n}^4 + 100\boldsymbol{n}^2 + 50 = 0$

putting n= $-\sqrt{5(10+\sqrt{102})}$ in the above equation we will get a negative value while putting n= $\sqrt{5(10+\sqrt{102})}$ in the above equation we will get a positive value,

Hence:

$$n \leq -\sqrt{5(10+\sqrt{102})}$$

or

$$n \geq \sqrt{5(10+\sqrt{102})}$$

Now what does $\sqrt{5(10+\sqrt{102})}$ $\boldsymbol{stand\ for}: 10.028484537$

As by definition: the function f and g should be set of natural numbers and it should grow by time (growth rate) we take $\boldsymbol{n} \geq 11$.

Or by analysis:

$\approx -\boldsymbol{n}^4 + 100\boldsymbol{n}^2 + 50 \leq 0$

$\approx 100\boldsymbol{n}^2 + 50 \leq \boldsymbol{n}^4$

$\boldsymbol{or}, \boldsymbol{n}^4 \geq 100\boldsymbol{n}^2 + 50$

if we take $\boldsymbol{n} = 10$

$\approx\ 10^4 \geq 100 \times 10^2 + 50$

$\approx\ 10000 \geq 10000 + 50$

$\approx\ 10000 \geq 10050[\boldsymbol{Not\ \ True}]$

if we take $\boldsymbol{n = 11}$

$\approx \ \boldsymbol{11^4 \geq 100 \times 11^2 + 50}$

$\approx \ \boldsymbol{14641 \geq 1210 + 50}$

$\approx \ \boldsymbol{14641 \geq 1260[\ True]}$

Hence, we confirm in both the ways that:

$\boldsymbol{n^4 + 100n^2 + 50 \leq 2n^4\ for\ all\ n \geq 11}$

Therefore , we get $n_0 = 11$.

And,

$\boldsymbol{n^4 + 100n^2 + 50 \leq 2\ \times n^4}\ in\ terms\ of\ \boldsymbol{f(n) \leq c\ \times g(n)}$

Hence:

$\boldsymbol{c \geq 2}$.

Therefore, f is O(g)

or in other words $\boldsymbol{O(n^4)}$.

Hence proved.

Example 5

Let $\boldsymbol{f(n) = n}$ for an algorithm. Let $\boldsymbol{g(n) = n}$. Prove that $\boldsymbol{f(n)\ of\ this\ algorithm\ is\ in\ O(n)}$.

Solution

$\boldsymbol{f(n) \leq c\ \times g(n)}$

$\Rightarrow \boldsymbol{n \leq 1 \times n, for\ all\ n \geq 1}$

$\Rightarrow \boldsymbol{n =\ O(n), c \geq 1\ and\ n_0 = 1}$

Example 6

Let $f(n) = 410$ for an algorithm. Let $g(n) = 410$. Prove that $f(n)\ of\ this\ algorithm\ is\ in\ O(1)$.

Solution

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

$\Rightarrow 410 \le 1 \times 410, for\ all\ n \ge 1$

$\Rightarrow 410 = O(1), c \ge 1\ and\ n_0 = 1$

Example 7

Find upper bound for $f(n) = 2n^3 - 2n^2$. Prove that $f(n)\ of\ this\ algorithm\ is\ in\ O(n^3)$.

Solution

The definition of Big-Oh notation is that $f(n) \le c \times g(n)$.

$\Rightarrow f(n) \le (2n^3 + 2n^3) - 2n^3$ $[As\ growth\ rate\ is\ doubled\ also\ 4n^3 \ge 2n^3]$

$\Rightarrow f(n) \le (4n^3) - 2n^3$

$\Rightarrow f(n) \le 2n^3$

And we can say:

$2n^3 - 2n^2 \le 2n^3$

Deducing it to $f(n) \le c \times g(n), we\ get$:

$\Rightarrow 2n^3 - 2n^2 \le 2 \times n^3$

Hence c = 2 and $g(n) = n^3$

By Inequality deduction:

$\Longrightarrow -2n^2 \leq 2n^3 - 2n^3$

$\Longrightarrow -2n^2 \leq 0$

$\Longrightarrow n^2 \leq 0$

$\Longrightarrow n \leq 0$

More specifically:

N	$2n^3 - 2n^2$	$2n^3$
0	0	0
1	0	1
2	8	16

Hence it starts from 1, hence $n \geq 1$

Therefore $n_0 = 1$.

Hence $f = O(g)$

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

NO UNIQUENESS in Above Method

There is no unique set of values for n_0 and c in proving the asymptotic bounds.

Let us consider, 100n + 5 = O(n). For this function there are multiple n_0 and c values possible.

Solution1:

100n+5 ≤100n+n

≈100n+5 ≤101n, for all $n \geq 5, n_0 = 5$ *and* $c \geq 101$ is a solution.

Solution2:

100n+5 ≤100n+5n

≈100n+5 ≤105n, for all $n \geq 1, n_0 = 1$ *and* $c \geq 105$ is also a solution.

Example 8

Let $f(n) = 3n^3 + 2n^2 + 3$ for an algorithm. Let $g(n) = n^3$. Prove that $f(n)$ *of this algorithm is in* $O(n^3)$.

Solution

The definition of Big-Oh notation is that $f(n) \leq c \times g(n)$. Therefore, one must show that $3n^3 + 2n^2 + 3 \leq cn^3$ holds good for a positive number c and for sufficiently large values of n.

$f(n) = 3n^3 + 2n^2 + 3$

$f(n) \leq 3n^3 + 2n^3 + 3$ (*as growth of functions* n^2 *to* n^3)

$f(n) \leq 3n^3 + 2n^3 + 3n^3$ (3 *is less than* n^3)

$f(n) \leq 8n^3$

It can be observed that $c = 3 + 2 + 3 = 8$ (one can approximate $2n^2$ and 3 to $2n^3$ *and* $3n^3$ *respectively*) . This condition holds good for any values of $c \geq 8$.

Let the polynomial be

$$f(n) = \sum_{i=0}^{m} a_i n^i$$

whose degree is m. Then one can show that $f(n) = O(n^m)$.

$$|f(n)| \leq |a_m|n^m + |a_{m-1}|n^{m-1} + \cdots + |a_1|n + |a_0|$$

$$\approx |f(n)| \leq |a_m|n^m + |a_{m-1}|n^m + \cdots + |a_1|n^m + |a_0|n^m \;\; for\, all\, n \geq 1$$

$$\approx \left(\sum_{i=0}^{m} |a_i|\right) n^m$$

$$\approx c \times n^m$$

$$\approx O(n^m)$$

Hence the above algorithm has $O(n^3)$.

This is another way we can prove the algorithm has *the complexity* .

Example 9

Let $f(n)=\frac{(2x^3+13\log_2 x)}{7n^2}$ $for\ an\ algorithm\ A. Prove\ that\ f(n)$ $of\ algorithm\ A\ is\ O(n)$.

Solution

It can be observed that $logx < x$ is always true. Therefore, one can argue that $13\log_2 x \le 13x$ and as $13x \le 13x^3$ always , one can rewrite $f(n)$ as follows:

$$f(n) \le \frac{2x^3+13x^3}{7n^2}$$

$$f(n) \le \frac{15x^3}{7n^2}$$

$$\cong 2n^{3-2}$$

$\cong 2n$ for all n > 1

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