

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:

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

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

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:

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

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

[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:

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

$$\approx 0 \leq n$$

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

Therefore $n_0 = 0$.

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:

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

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

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 .

$$3n + 8 \leq 3n + n$$

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

[As highest degree of polynomial n is 1]

$$\therefore 3n + 8 \leq 4n$$

$$\approx 8 \leq 4n - 3n$$

$$\approx 8 \leq n$$

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

$\therefore n_0 = 8$

$$3n + 8 \leq 4n$$

Can be written as:

$$3n + 8 \leq 4 \times n$$

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

Therefore, f is $O(g)$

or in other words $O(n)$.

Hence proved.

Example 3

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

Solution

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

In order to prove that we know:

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

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

Where $g(n)$ is in n^2 .

Hence, we can show that:

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

$$n^2 + 1 \leq n^2 + n^2$$

$$\approx n^2 + 1 \leq 2n^2 [n^2 \text{ is always } \leq 2n^2]$$

[As highest degree of polynomial n is 2]

$$\therefore n^2 + 1 \leq 2n^2$$

$$\approx 1 \leq 2n^2 - n^2$$

$$\approx 1 \leq n^2$$

$$\approx -n^2 + 1 \leq 0$$

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

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

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

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

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

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

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

[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 $n = -1$

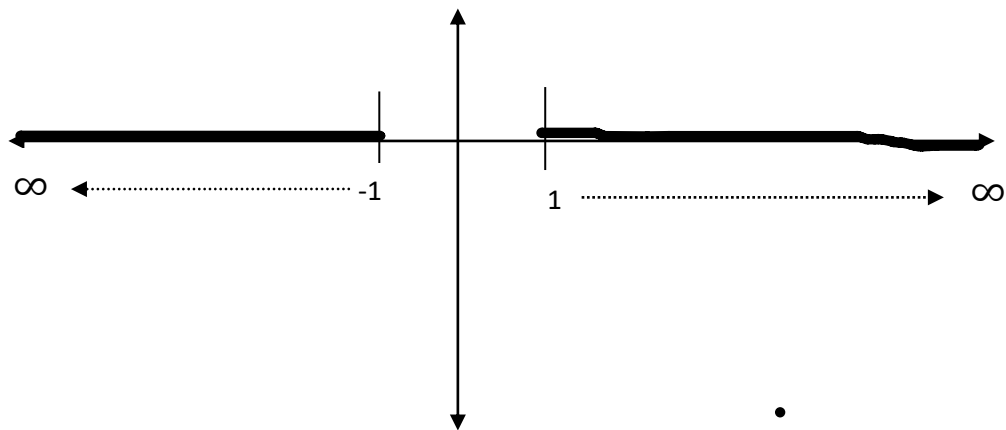
$$\approx (-1 - 1) \times (-1 + 1) \leq 0$$

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

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

That implies $n \leq -1$

$$\therefore -1 \geq n \geq 1$$



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

Hence now we get:

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

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

Therefore , we get $n_0 = 1$.

And,

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

Hence:

$$c \geq 2 .$$

Therefore, f is $O(g)$

or in other words $O(n^2)$.

Hence proved.

Example 4

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

Solution

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

In order to prove that we know:

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

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

Where $g(n)$ is in n^4 .

Hence, we can show that:

$$\begin{aligned}
n^4 + 100n^2 + 50 &\leq c \times n^4 \\
&\approx n^4 + 100n^2 + 50 \leq n^4 + n^4 \\
&\approx n^4 + 100n^2 + 50 \leq 2n^4 \text{ [} n^4 \text{ is always } \leq 2n^4 \text{]} \\
&\text{[As highest degree of polynomial } n \text{ is 4]}
\end{aligned}$$

Now,

$$\begin{aligned}
&\approx -2n^4 + n^4 + 100n^2 + 50 \leq 0 \\
&\approx -n^4 + 100n^2 + 50 \leq 0
\end{aligned}$$

We can write the above equation as:

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

Taking the equation:

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

Rewriting the equation, $u = n^2$ and $u^2 = n^4$:

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

Solving with quadratic equation formula:

Quadratic Equation of the form $ax^2 + bx + c = 0$:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned} u &= \frac{(-100 + 10 \sqrt{102})}{-2} \\ &= \frac{10(-10 + \sqrt{102})}{-2} \\ &= -5(-10 + \sqrt{102}) - -i \end{aligned}$$

$$\begin{aligned} u &= \frac{(-100 - 10 \sqrt{102})}{-2} \\ &= \frac{-10(10 + \sqrt{102})}{-2} \\ &= 5(10 + \sqrt{102}) - -ii \end{aligned}$$

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 $\sqrt{f(a)}$, $-\sqrt{f(a)}$

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

Now we can easily understand,

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

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 $\sqrt{5(10 + \sqrt{102})}$ 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 $n \geq 11$.

Or by analysis:

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

$$\approx 100n^2 + 50 \leq n^4$$

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

if we take $n = 10$

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

$$\approx 10000 \geq 10000 + 50$$

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

if we take $n = 11$

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

$$\approx 14641 \geq 1210 + 50$$

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

Hence, we confirm in both the ways that:

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

Therefore, we get $n_0 = 11$.

And,

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

Hence:

$$c \geq 2.$$

Therefore, f is $O(g)$

or in other words $O(n^4)$.

Hence proved.

Example 5

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

Solution

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

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

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

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

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

$$\Rightarrow 410 \leq 1 \times 410, \text{ for all } n \geq 1$$

$$\Rightarrow 410 = O(1), c \geq 1 \text{ and } n_0 = 1$$

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 $f(n) \leq c \times g(n)$.

$$\Rightarrow f(n) \leq (2n^3 + 2n^3) - 2n^3$$

$$[\text{As growth rate is doubled } \left((2n^3 + n^3) - 2n^3 = n^3 \leq 2n^3 \right), \text{ hence we need } ? \geq 2n^3]$$

$$\text{Therefore, } (2n^3 + 2n^3) - 2n^3 \geq 2n^3]$$

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

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

And we can say:

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

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

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

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

By Inequality deduction:

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

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

$$\Rightarrow n^2 \leq 0$$

$$\Rightarrow n \leq 0$$

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:

$$100n+5 \leq 100n+n$$

$\approx 100n+5 \leq 101n$, for all $n \geq 5$, $n_0 = 5$ and $c \geq 101$ is a solution.

Solution2:

$$100n+5 \leq 100n+5n$$

$\approx 100n+5 \leq 105n$, 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 .

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

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

$$f(n) \leq 3n^3 + 2n^3 + 3n^3 \text{ (3 is less than } n^3)$$

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

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 $f(n) = O(n^m)$.

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

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

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

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

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

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 $\log x < x$ is always true. Therefore, one can argue that $13 \log_2 x \leq 13x$ and as $13x \leq 13x^3$ always, one can rewrite $f(n)$ as follows:

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

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

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

$$\cong 2n \text{ for all } n > 1$$

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

Example 10

Consider an Algorithm that is assume to run in time $O(n^2)$ and that only five seconds to compute results for an instance of size 30. How long will the algorithm take to compute the results if the instance size is increased to 50?

Solution

Recollect that $O(n^2)$ implies that $f(n) \leq c \times g(n)$, that is,

$f(n) \leq c \times n^2$. Here c is a constant that depends on the machine. As mentioned in the problem, the algorithm takes five seconds to compute 30 instances. Therefore, one can calculate the value of c as follows:

$$cn^2 \text{ steps} = 5 \text{ seconds}$$

$$c(30)^2 = 900c \text{ steps.}$$

$$\therefore c = \frac{5}{900}$$

Once the values of c are decided , time for all other instances can easily be calculated . if the algorithm instance is increased to 50 , then the time taken by the program can be calculated as follows:

$$c(50)^2 = 2500c$$

$$\text{Substituting the value of } c \text{ in this equation, one gets } 2500 \times \frac{5}{900} = \frac{125}{9}$$

$$= 13.88 \text{ seconds .}$$

The calculation shows that the algorithm requires 13.88 seconds to execute 50 instances.
