

# Big-O What are the constants k and n0 in the formal definition of the order of an algorithm?

Asked 2 years, 6 months ago   Active 2 years, 6 months ago   Viewed 1k times

In my textbook I see the following:

4

## Definition of the order of an algorithm

Algorithm A is order  $f(n)$  -- denoted  $O(f(n))$  -- if constants  $k$  and  $n_0$  exist such that A requires no more than  $k * f(n)$  time units to solve a problem of size  $n \geq n_0$ .

**I understand:** Time requirements for different complexity classes grow at different rates. For instance, with increasing values of  $n$ , the time required for  $O(n)$  grows much more slowly than  $O(n^2)$ , which grows more slowly than  $O(n^3)$ , and so forth.

**I do not understand:** How  $k$  and  $n_0$  fit into this definition.


1. What is  $n_0$ ? Specifically, why does  $n$  have subscript 0, what does this subscript mean?
2. With question 1 answered, what does a 'a problem of size  $n \geq n_0$ ' mean? A larger data set? More loop repetitions? A growing problem size?
3. What is  $k$  then? Why is  $k$  being multiplied by  $f(n)$ ? What does  $k$  have to do with increasing the problem size -  $n$ ?


**I've already looked at:**

1. [Big Oh Notation - formal definition](#)
2. [Constants in the formal definition of Big O](#)

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email

 Sign up with Google

 Sign up with GitHub

 Sign up with Facebook



[algorithm](#) [performance](#) [big-o](#) [notation](#)

edited Jun 20 at 9:12



Community ♦

1 • 1

asked Mar 5 '18 at 8:50



Estex

63 • 6

### 3 Answers

Active

Oldest

Votes



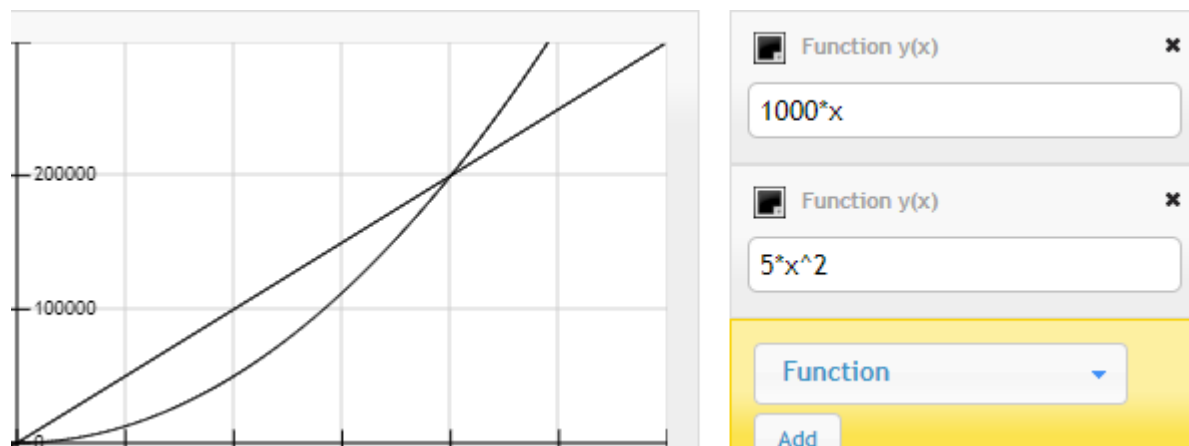
4



1)  $n > n_0$  - means that we agree that for small  $n$  A might need more than  $k \cdot f(n)$  operations. Eg. bubble sort might be faster than quick sort or merge sort for very small inputs. Choice of 0 as a subscript is completely due to author preferences.

2) Larger input size.

3)  $k$  is a constant. Suppose one algorithm performs  $1000 \cdot n$  operation for input of size  $n$ , so it is  $O(n)$ . Another algorithm needs  $5 \cdot n^2$  operations for input of size  $n$ . That means for input of size 100, first algorithm needs 100,000 ops and the second one 50,000 ops. So, for input size about 100 you better choose the second one though it is quadratic, and the first one is linear. On the following picture you can see that  $n_0 = 200$ , because only with  $n$  greater than 200 quadratic function becomes more expensive than linear (here i assume that  $k$  equals 1).



Join Stack Overflow to learn, share knowledge, and build your career.

[Sign up with email](#)[Sign up with Google](#)[Sign up with GitHub](#)[Sign up with Facebook](#)

- 2 @Estex You need to look at an algorithm implementation and just count operations. Suppose you have linear complexity algorithm, so you count number of operation on every loop iteration and that is your  $k$ . – Yola Mar 7 '18 at 5:20
- 2 Note that precisely computing a  $k$  in general is *not feasible*. If you would break it all the way down you would need to count the number of machine instructions. And some statement might trigger multiple instructions internally, you don't know in general. But it does not really matter since the definition allows an arbitrary big  $k$  as long as it is a constant. So in an analysis you would just leave it as abstract  $k$ , not specifying a specific number. And sometimes you would need to increase it, for example if the other side has  $2k$  you would need an  $a > 2k$  (which exists of course). – Zabuzard Mar 7 '18 at 16:29
- 2 Same holds for an  $n_0$ . You can easily compute numbers if you compare ordinary functions, but not with algorithms. You first need to abstract the algorithm into some kind of function and this often involves just counting the iterations in dependency to the input size, like  $27n^2 + 3n$  iterations. This function then, using Big-O definition, yields  $O(n^2)$ . Finding the numbers is now easy, you choose  $k = 28$  and receive  $28n^2 > 27n^2 + 3n$ . But this equation doesn't hold for all  $n$ , you would need to find the  $n_0$  such that it starts holding for  $n > n_0$ . For example for  $n_0 = 10$ . – Zabuzard Mar 7 '18 at 16:33
- 1 Or you choose  $k = 100$ , then it will probably hold for all  $n$ , so  $n_0 = 1$  is fine. So you don't need to determine  $k$  and  $n_0$  that precise that lower numbers won't work. Any combination for which it holds it fine, you may select arbitrary big numbers. – Zabuzard Mar 7 '18 at 16:34
- 1 @Zabuza agree, here i provided this computation to show the principle, however computation of  $k$  and  $n_0$  in real system maybe complicated and we usually agree on some level of abstraction. – Yola Mar 7 '18 at 16:38

1. It means the first value for  $n$  for which the rest holds true (i.e. we're only interested in high enough values for  $n$ )

2. Problem size, usually the size of the input.

3. It means you don't care about the different (for example) between  $3n^2$  and  $400n^2$ , so any value that is high enough to satisfy the equation is OK.



All of these conditions aim to simplify the O notation, making the difference between simple and complex operations mute (e.g. you don't care if an operation is one or 20 cycles as long as the number is finite).

answered Mar 5 '18 at 8:58



Ofir

7,709 ● 2 ● 25 ● 42

Link enough to satisfy which equation? – Estex Mar 5 '18 at 21:22

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email



Sign up with Google



Sign up with GitHub



Sign up with Facebook





1



$n$  is the problem size, however that is best measured. Thus  $n_0$  is a specific constant  $n$ , specifically the threshold after which the relationship holds. The specific value is irrelevant for big-oh, being only interested in its existence.

$k$  is also an arbitrary constant, whose bare existence (in conjunction with  $n_0$ ) is important for big-oh.

Naturally, people are also interested in smaller problems, and in fact the perfect algorithm for a big problem might be decidedly inefficient for a small one, due to the constants involved.

answered Mar 5 '18 at 9:02



Deduplicator

40.1k ● 6 ● 58 ● 101

Join Stack Overflow to learn, share knowledge, and build your career.

[Sign up with email](#)[Sign up with Google](#)[Sign up with GitHub](#)[Sign up with Facebook](#)