





Computing > Computer science > Algorithms > Asymptotic notation
Asymptotic notation


 Asymptotic notation


 Big- θ (Big-Theta) notation

 Functions in asymptotic notation

 Practice: Comparing function growth

 Big-O notation

 Big- Ω (Big-Omega) notation

 Practice: Asymptotic notation

< Computing • Computer science • Algorithms • Asymptotic notation

Asymptotic notation

 Google Classroom  Facebook  Twitter  Email

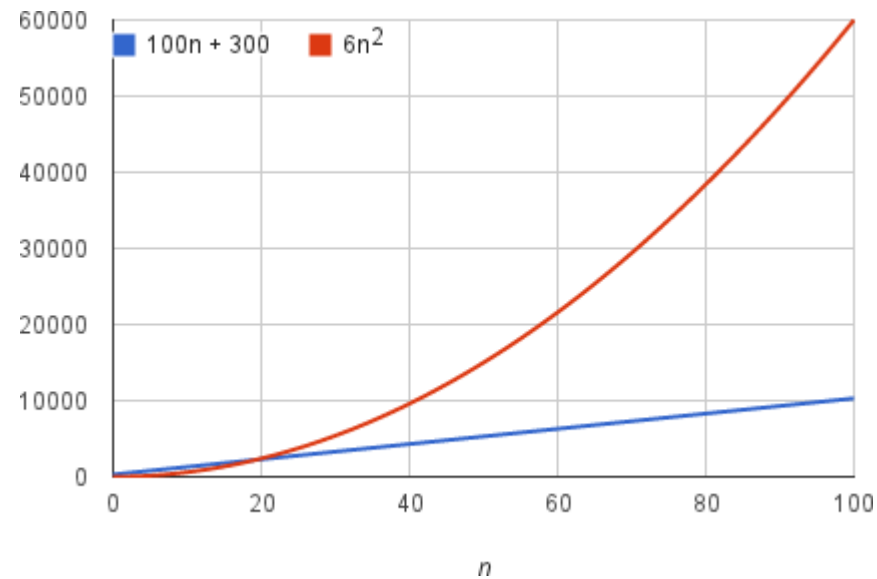
So far, we analyzed linear search and binary search by counting the maximum number of guesses we need to make. But what we really want to know is *how long* these algorithms take. We're interested in *time*, not just guesses. The running times of linear search and binary search include the time needed to make and check guesses, but there's more to these algorithms.

The running time of an algorithm depends on how long it takes a computer to run the lines of code of the algorithm—and that depends on the speed of the computer, the programming language, and the compiler that translates the program from the programming language into code that runs directly on the computer, among other factors.

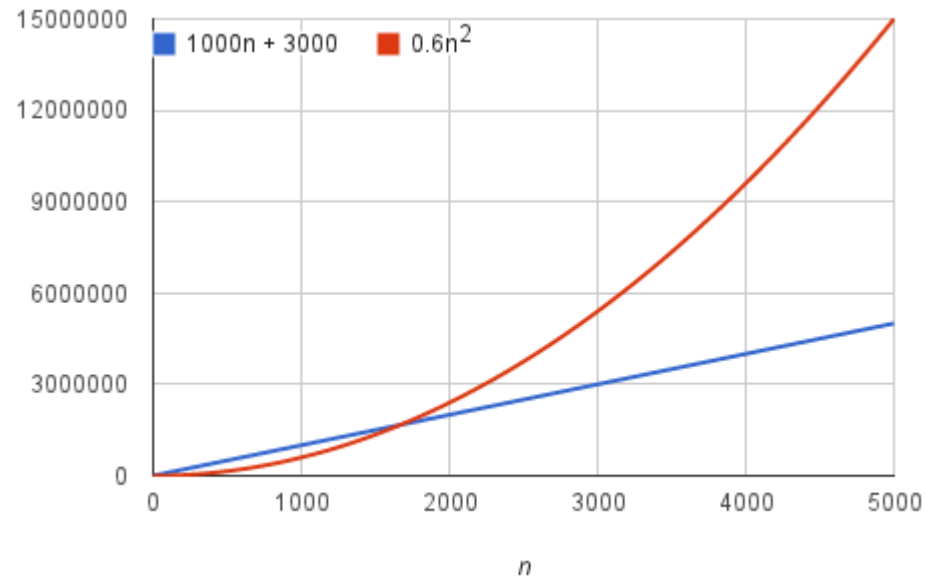
Let's think about the running time of an algorithm more carefully. We can use a combination of two ideas. First, we need to determine how long the algorithm takes, in terms of the size of its input. This idea makes intuitive

sense, doesn't it? We've already seen that the maximum number of guesses in linear search and binary search increases as the length of the array increases. Or think about a GPS. If it knew about only the interstate highway system, and not about every little road, it should be able to find routes more quickly, right? So we think about the running time of the algorithm as a *function of the size of its input*.

The second idea is that we must focus on how fast a function grows with the input size. We call this the **rate of growth** of the running time. To keep things manageable, we need to simplify the function to distill the most important part and cast aside the less important parts. For example, suppose that an algorithm, running on an input of size n , takes $6n^2 + 100n + 300$ machine instructions. The $6n^2$ term becomes larger than the remaining terms, $100n + 300$, once n becomes large enough, 20 in this case. Here's a chart showing values of $6n^2$ and $100n + 300$ for values of n from 0 to 100:



We would say that the running time of this algorithm grows as n^2 , dropping the coefficient 6 and the remaining terms $100n + 300$. It doesn't really matter what coefficients we use; as long as the running time is $an^2 + bn + c$, for some numbers $a > 0$, b , and c , there will always be a value of n for which an^2 is greater than $bn + c$, and this difference increases as n increases. For example, here's a chart showing values of $0.6n^2$ and $1000n + 3000$ so that we've reduced the coefficient of n^2 by a factor of 10 and increased the other two constants by a factor of 10:



The value of n at which $0.6n^2$ becomes greater than $1000n + 3000$ has increased, but there will always be such a crossover point, no matter what the constants.

By dropping the less significant terms and the constant coefficients, we can focus on the important part of an algorithm's running time—its rate of growth—without getting mired in details that complicate our understanding. When we drop the constant coefficients and the less significant terms, we use **asymptotic notation**. We'll see three forms of it: big- Θ notation, big- O notation, and big- Ω notation.

This content is a collaboration of [Dartmouth Computer Science](#) professors [Thomas Cormen](#) and [Devin Balkcom](#) plus the Khan Academy computing curriculum team. The content is licensed [CC-BY-NC-SA](#).

Sort by: Top Voted ▼

[Questions](#)

Tips & Thanks

Want to join the conversation?

You need at least 5000 energy points to get started.



Gangadhar Lahane 5 years ago



more ▼

Why we always care about algorithms worst case performance?

1 comment

(11 votes) Flag more ▼



Jaime Gonzalez 5 years ago



more ▼

Solving worst cases is crucial even for the operation of a modest

computer, like the one I'm working on right now: quick access to billions of pieces of information in the memory devices, carrying out an amazing amount of arithmetical and logical operations per second, being able to display animated pictures, and so forth. All this could not be achieved just by brute-force approaches, like linear search, so we need very efficient algorithms that can do much, much more, in a limited amount of time. And this is what this course is about.

Much beyond a normal computer, just imagine the daunting tasks behind big data problems, like a Google search.

Hope this helps.

1 comment

(85 votes)  Flag [more](#) 

See 8 more replies



Siayan God 3 years ago



[more](#) 

Please Explain How is $f(x) = 4x^2 - 5x + 3$ is $O(x^2)$ derived

$$|f(x)| = |4x^2 - 5x + 3|$$

$$\leq |4x^2| + |-5x| + |3|$$

$$\leq 4x^2 + 5x + 3, \text{ for all } x > 0$$

$$\leq 4x^2 + 5x^2 + 3x^2, \text{ for all } x > 1$$

$$\leq 12x^2, \text{ for all } x > 1$$

Hence we conclude that $f(x)$ is $O(x^2)$

Can someone explain the above proof step by step?

i. Why do we take absolute value?

ii. Why and how were all the term replaced by x^2 term?

(10 votes)  Flag [more](#) 



Cameron 3 years ago



more ▾

By definition, $f(n)$ is $O(g(n))$ if:

There exists constants k, N where $k > 0$, such that for all $n > N$:

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

So to prove that $f(x) = 4x^2 - 5x + 3$ is $O(x^2)$ we need to show that:

There exists constants k, N where $k > 0$, such that for all $x > N$:

$$f(x) \leq k * g(x)$$

$$4x^2 - 5x + 3 \leq k * x^2$$

The way we show that is by finding some k and some N that will work.

The basic strategy is:

- break up $f(x)$ into terms
- for each term find some term with a coefficient $* x^2$ that is clearly equal to or larger than it
- this will show that $f(x) \leq$ the sum of the larger x^2 terms
- the coefficient for the sum of the x^2 terms will be our k

Explanation of provided proof:

$$f(x) = 4x^2 - 5x + 3$$

a number is always \leq its absolute value e.g. $-1 \leq |-1|$ and $2 \leq |2|$

so we can say that:

$$f(x) \leq |f(x)|$$

$$f(x) \leq |f(x)| = |4x^2 - 5x + 3|$$

$4x^2 + 3$ will always be positive, but $-5x$ will be negative for $x > 0$

so we know that $-5x$ is $\leq |-5x|$, so we can say that:

$$f(x) \leq |4x^2| + |-5x| + |3|$$

For $x > 0$ $|4x^2| + |-5x| + |3| = 4x^2 + 5x + 3$, so we can say that:

$$f(x) \leq 4x^2 + 5x + 3, \text{ for all } x > 0$$

Suppose $x > 1$. Multiply both sides by x to show that $x^2 > x$

So we can say $x \leq x^2$.

This let's us replace each of our x terms with x^2 so we can say that:

$$f(x) \leq 4x^2 + 5x^2 + 3x^2, \text{ for all } x > 1$$

$$4x^2 + 5x^2 + 3x^2 = 12x^2 \text{ so we can say that:}$$

$$f(x) \leq 12x^2, \text{ for all } x > 1$$

So our $k = 12$ and since we had to assume $x > 1$ we pick $N = 1$

A slightly different approach that I would use would be:

Suppose $N = 1$, i.e $x > 1$ (This usually has all the nice properties you want for simple big-Oh proofs)

$$f(x) = 4x^2 - 5x + 3$$

$$f(x) \leq 4x^2 + 5x + 3 \text{ (for } x > 1 \text{ a positive number } * x \text{ is larger than a negative number } * x \text{)}$$

Since $x > 1$, we can say $x^2 > x$ (by multiplying both sides of inequality by x)

$$f(x) \leq 4x^2 + 5x^2 + 3x^2$$

$$f(x) \leq 12x^2$$

So our $k = 12$ and $N=1$ (which we assumed at the beginning)

Hope this makes sense

6 comments

(26 votes)  Flag [more](#) 



jmReed 2 years ago



[more](#) 

What age/grade level is this course designed for?

2 comments

(10 votes)  Flag [more](#) 



jdsutton 2 years ago



[more](#) 

This is college-level material, but that doesn't mean you can't learn it sooner.

5 comments

(18 votes)  Flag [more](#) 

See 2 more replies



Miles Lilly 4 years ago

[more](#) 

I this related to the idea of computational complexity?

(6 votes)  Flag [more](#) 





Cameron 4 years ago

[more](#) 

Absolutely. We use asymptotic notation to communicate the computational complexity of algorithms.

2 comments

(7 votes)  Flag [more](#) 

See 1 more reply




abrianna ransom 10 months ago

[more](#) 

i dont really get the whole idea of it....

3 comments

(6 votes)  Flag [more](#) 



shamathmika76 3 years ago

[more](#) 

What would happen when a is negative in the equation $(a * n^2 + b * n + c)$?
What would the rate of growth be?

1 comment

(2 votes)  Flag [more](#) 



Cameron 3 years ago



[more](#) 

The running time equation should always be ≥ 0 . A running time < 0 doesn't really make sense (it would mean your problem would be solved before you started to solve it).

$a * n^2 + b * n + c$ with $a < 0$ would become negative when n becomes large, so it doesn't make sense.

1 comment

(10 votes)  Flag [more](#) 

See 1 more reply



Delilah Montoya 10 months ago

[more](#) 

Why is this sort of confusing.

(3 votes)  Flag [more](#) 



AlaskaJoe 4 years ago

[more](#) 

The whole point of the function example was to explain that there is a point of crossover between a quadratic function and a linear function assuming they both move in the same direction?

(1 vote)  Flag [more](#) 

**Cameron** 4 years ago[more](#) ▾

Some of the major points were:

- given large enough values of n we only have to worry about the fastest growing terms
- no matter what the constants are $a * n^2$ grows faster than $b * n$ (for large values of n)

8 comments(5 votes) Flag [more](#) ▾**Jorge Manuel Almeida Mo...** 6 months ago[more](#) ▾

i don't understand these equations. Which math skills do i need to understand this course?

(1 vote) Flag [more](#) ▾**anonymousdebater** 6 months ago[more](#) ▾

You will probably need some pre-algebra at the least. There are some mentions of logarithms, which are more advanced mathematics.

2 comments(5 votes) Flag [more](#) ▾**Arpan Man Sainju** 4 years ago[more](#) ▾

Please add information on little o , little ω as well. I like the quizzes. I feel like I am learning more from the quizzes. Thank you for your effort.

(2 votes) Flag [more](#) ▾

**Cameron** 4 years ago

more ▾

There isn't much to little-oh and little-omega once you have covered big-Oh and big-Omega.

$f(n)$ is $o(g(n))$ if $f(n)$ grows strictly slower than $g(n)$ asymptotically
i.e. $f(n)$ is $O(g(n))$ but $g(n)$ is not $O(f(n))$ ($f(n)$ and $g(n)$ are not in same complexity class)

i.e. $\lim_{n \rightarrow \infty} (f(n)/g(n)) = 0$

$f(n)$ is $\omega(g(n))$ if $f(n)$ grows strictly faster than $g(n)$ asymptotically
i.e. $f(n)$ is $\Omega(g(n))$ but $g(n)$ is not $\Omega(f(n))$ ($f(n)$ and $g(n)$ are not in same complexity class)

i.e. $\lim_{n \rightarrow \infty} (f(n)/g(n)) = \infty$

(2 votes)  Flag more ▾

See 1 more reply

Show more...

Big- θ (Big-Theta) notation >