



Computing > Computer science > Algorithms > Asymptotic notation
Asymptotic notation

Computing • Computer science • Algorithms
• Asymptotic notation

Big-O notation

Google Classroom

Facebook

Twitter

Email



Functions in asymptotic notation



Practice: Comparing function growth



Big-O notation



Big-Ω (Big-Omega) notation



Practice: Asymptotic notation

Next lesson

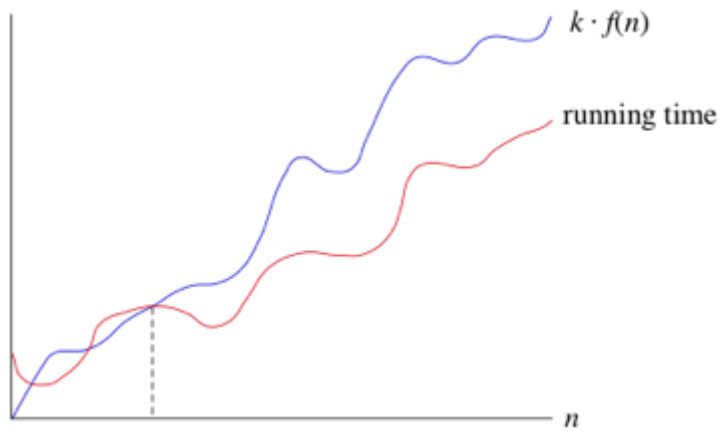
Selection sort

We use big- Θ notation to asymptotically bound the growth of a running time to within constant factors above and below. Sometimes we want to bound from only above.

For example, although the worst-case running time of binary search is $\Theta(\log_2 n)$, it would be incorrect to say that binary search runs in $\Theta(\log_2 n)$ time in *all* cases. What if we find the target value upon the first guess? Then it runs in $\Theta(1)$ time. The running time of binary search is never worse than $\Theta(\log_2 n)$, but it's sometimes better.

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-O" notation for just such occasions.

If a running time is $O(f(n))$, then for large enough n , the running time is at most $k \cdot f(n)$ for some constant k . Here's how to think of a running time that is $O(f(n))$:



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

Now we have a way to characterize the running time of binary search in all cases. We can say that the running time of binary search is *always* $O(\log_2 n)$. We can make a stronger statement about the worst-case running time: it's $\Theta(\log_2 n)$. But for a blanket statement that covers all cases, the strongest statement we can make is that binary search runs in $O(\log_2 n)$ time.

If you go back to the definition of big- Θ notation, you'll notice that it looks a lot like big-O notation, except that big- Θ notation bounds the running time from both above and below, rather than just from above. If we say that a running time is $\Theta(f(n))$ in a particular situation, then it's also $O(f(n))$. For example, we can say that because the worst-case running time of binary search is $\Theta(\log_2 n)$, it's also $O(\log_2 n)$.

The converse is not necessarily true: as we've seen, we can say that binary search always runs in $O(\log_2 n)$ time but *not* that it always runs in $\Theta(\log_2 n)$ time.

Because big-O notation gives only an asymptotic upper bound, and not an asymptotically tight bound, we can make statements that at first glance seem incorrect, but are technically correct. For example, it is absolutely correct to say that binary search runs in $O(n)$ time. That's because the running time grows no faster than a constant times n . In fact, it grows slower.

Think of it this way. Suppose you have 10 dollars in your pocket. You go up to your friend and say, "I have an amount of money in my pocket, and I guarantee that it's no more than one million dollars." Your statement is absolutely true, though not terribly precise.

One million dollars is an upper bound on 10 dollars, just as $O(n)$ is an upper bound on the running time of binary search. Other, imprecise, upper bounds on binary search would be $O(n^2)$, $O(n^3)$, and $O(2^n)$. But none of $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, and $\Theta(2^n)$ would be correct to describe the running time of binary search in any case.

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?




[Log in](#)



Roman 5 years ago

[more](#) 

I didn't understand this sentence: "Other, imprecise, upper bounds on binary search would be $O(n^2)$, $O(n^3)$, and $O(2^n)$. But none of $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, and $\Theta(2^n)$ would be correct to describe the running time of binary search in any case." Can someone explain it to me please? Thank you!

Reply  **1** comment (17 votes)  Upvote  Downvote



Cameron 5 years ago    [m...](#) 

Here's a couple definitions to keep in mind:
 if $f(n)$ is $O(g(n))$ this means that $f(n)$ grows asymptotically no faster than $g(n)$
 if $f(n)$ is $\Theta(g(n))$ this means that $f(n)$ grows asymptotically at the same rate as $g(n)$

Let's call the running time of binary search $f(n)$.

$f(n)$ is $k * \log(n) + c$ (k and c are constants)

Asymptotically, $\log(n)$ grows no faster than $\log(n)$ (since it's the same), n , n^2 , n^3 or 2^n .

So we can say $f(n)$ is $O(\log(n))$, $O(n)$, $O(n^2)$, $O(n^3)$, and $O(2^n)$.

This is similar to having $x = 1$, and saying $x \leq 1$, $x \leq 10$, $x \leq 100$, $x \leq 1000$, $x \leq 1000000$.

All of these statements are true, but the most precise statement is $x \leq 1$. By precise, we mean that it gives us the best idea of what x actually is.

Asymptotically, $\log(n)$ grows at the same rate as $\log(n)$ (since it is the same).

So, we can say that $f(n)$ is $\Theta(\log(n))$

This would be similar to having $x=1$ and then saying $x = 1$, which would be a precise statement that tells us what x is.

However, asymptotically, $\log(n)$ grows slower than n , n^2 , n^3 or 2^n i.e. $\log(n)$ does not grow at the same rate as these functions.

So, we can not say $f(n)$ is $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, and $\Theta(2^n)$.

Similarly if $x = 1$, we can not say that $x = 10$, $x = 100$, $x = 1000$, or $x = 1000000$.

Hope this makes sense

6
comments (79 votes)  Upvote  Downvote

See 3 more replies

**William Chargin** 6 years ago

more ▾

If I'm not mistaken, the first paragraph is a bit misleading.

Before, we used big-Theta notation to describe the *worst case* running time of binary search, which is $\Theta(\lg n)$. The *best case* running time is a completely different matter, and it is $\Theta(1)$.

That is, there are (at least) three different types of running times that we generally consider: best case, average/expected case, and worst case. Usually it's the latter two that are the most useful. For binary search, the best case time is $\Theta(1)$, and the expected and worst case times are $\Theta(\lg n)$.

Reply • **Comment** (25 votes) Upvote Downvote

**Dominic Alonso** 3 years ago

m... ▾

I would agree with this.

Comment (2 votes) Upvote Downvote



☺ ☹ ☺ **Natth4545** ☺ ☹ ☺ 5 years ago

m... ▾

And I thought I could do this. :(Cant learn everything.

Reply • **1 comment** (4 votes) Upvote Downvote

**Yahaya Aluke** 5 years ago

m... ▾

If you think you can't, you're right. If you think you can you're also right. It's all in how you think about it. Stick for awhile till the function storm passes, it'll surprise you how

you don't even really need to know the math, just how fast some few functions growth because you have to compare the rate of growth of algorithms to them. Like knowing the order the alphabets come so you know where to place a stray alphabet.

Comment (20 votes)  Upvote  Downvote



See 4 more replies



c0ffeeartc 4 years ago

[more](#) 

The complexity of this article is (n^3) at least

Reply • Comment (14 votes)  Upvote  Downvote



purposenigeria 3 years ago

[m...](#) 

What is upper bound, lower bound and tight bound ?

Comment (4 votes)  Upvote  Downvote

See 2 more replies





Veronica 6 years ago

[more](#) 

I really don't get this paragraph, could someone please explain it to me in other words?? Thank you!

"Now we have a way to characterize the running time of binary search in all cases. We can say that the running time of binary search is always $O(\lg n)O(\lg n)$. We can make a stronger statement about the worst-case running time: it's $\Theta(\lg n)\Theta(\lg n)$. But for a blanket statement that covers all cases, the strongest statement we can make is that binary search runs in $O(\lg n)O(\lg n)$ time."

Reply • Comment (4 votes)  Upvote  Downvote



Enrico Susatyo 5 years ago m... ▾

It means that:

- You can find a scenario when binary search takes $\Theta(\lg n)$.
- Ignoring worst case scenario, binary search will generally take $\text{Big-O}(\lg n)$ time. Because not all of the binary search runs will take $\Theta(\lg n)$.

Comment (3 votes) Upvote Downvote

See 1 more reply



sparksro 6 years ago more ▾

I have a question on upper and lower bounds to make sure my understanding is spot on. Please tell me if I am correct or not and why if not. Given the set $S = \{2, 3, 5, 7, 9, 12, 17, 42\}$ A lower bound could be 2 or 3 for example but in set S only 2 is the tight lower bound and only 42 is the tight upper bound. Is this correct?

Reply • Comment (2 votes) Upvote Downvote



JaniceHolz 6 years ago mo... ▾

A lower bound has to be less than or equal to all members of the set. Therefore, here 3 is not a lower bound because it is greater than a member of the set (2). 1 is a lower bound, -3592 is a lower bound, 1.999 is a lower bound -- because each of those is less than every member of the set.

There is always only 1 tight lower bound: the greatest of all the lower bounds. Here, 2 is indeed the tight

lower bound.

Similarly, there is always only 1 tight upper bound: the least of all the upper bounds. Here, 42 is indeed the tight upper bound.

Try this example:

Given the set $S2 = \{-12, -5, 0, 1, 3, 3\}$, what is a lower bound? What is an upper bound?

3 (10 votes)  Upvote  Downvote

comments

See 3 more replies





mugnaio 5 years ago

more ▾

I didn't understand this: "If we say that a running time is $\Theta(f(n))$ in a particular situation, then it's also $O(f(n))$."

Binary search is $\Theta(1)$ in a particular situation (best case) but it is not $O(1)$.

Reply • Comment (3 votes)  Upvote  Downvote



Cameron 5 years ago

more ▾

It looks like you are confusing O and Ω with worst case and best case. They are not the same. First we specify the case (worst, best, average, etc.) and then we specify O , Ω (upper bound, lower bound) or Θ (tight bounds).

For Binary search:

In the best case scenario (our initial guess finds the target value):

- binary search is $\Theta(1)$ and as a result is also $\Omega(1)$ and $O(1)$.

In the worst case scenario (our target is not in the array)

- binary search is $\Theta(\log n)$ and as a result is also $\Omega(\log n)$ and $O(\log n)$.

4

(6

 Upvote  Downvote

comments votes)

▲ Upvote ▼ Downvote

See 2 more replies

**Harshit Vashistha** a year ago

more ▾

"the running time grows at most this much, but it could grow more slowly." what does this mean?

Reply • Comment

(4

votes)



Upvote



Downvote



5 months

harman052 ago

mo... ▾

It means this is the max rate of growth it can achieve but in some cases it can grow at smaller rate. For example, if the target element is not present in the input, it would be the worst case and so complexity will be maximum. But for the same input, if we find target in the very first attempt then it will be the best case and complexity will be minimum.

Big Θ is tight bound (i.e., both upper and lower bound) therefore it tells precisely about the complexity. Big O is upper bound i.e. it tells about the maximum complexity this algorithm can have which in other words means, this is the maximum growth rate, but it can grow at smaller rate in some cases.

Comment

(1

vote)



Upvote



Downvote



**Elvensong** a year ago

more ▾

These two statements seem contradictory. Earlier, the article says that the running time of binary search is always $O(\log(n))$ [Para. 6].

Then it says the running time of binary search is $O(n)$ [Para. 3 from bottom].

I've been taught that the running time of binary search is $O(\log(n))$ while linear search is $O(n)$. Confusing!



Reply • Comment (3 votes)  Upvote  Downvote



justinj1776 5 years ago

more ▾

Is it absolutely correct to say that binary search runs in $\text{Big-}O(n)$? Why couldn't we say it can run in $\text{Big-}O(n^2)$. Since the upperbound for binary search is $\text{Big-}O(\log(n))$ do you mean to say that we start from n and then go to n^2 for considering upperbound of an algorithm. For example if an algorithm runs in $\text{Big-}\Theta(n)$ can we say it runs in $\text{Big-}O(n^2)$?

Reply • Comment (2 votes)  Upvote  Downvote



Cameron 5 years ago

more ▾

Binary search is $\Theta(\log n)$ which means that it is $O(\log n)$ and $\Omega(\log n)$

Since binary search is $O(\log n)$ it is also $O(\text{any function larger than } \log n)$

i.e. binary search is $O(n)$, $O(n^2)$, $O(n^3)$, $O(e^n)$, $O(n!)$, etc,

Another way to express this is by saying:

Binary search doesn't run slower than really fast algorithms ($O(\log n)$), so:

Binary search doesn't run slower than fast algorithms ($O(n)$), so:

Binary search doesn't run slower than moderate to slow algorithms ($O(n^2)$, $O(n^3)$), so:

Binary search doesn't run slower

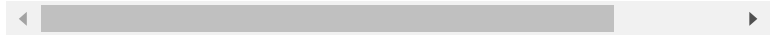
than horribly slow algorithms ($O(e^n)$, $O(n!)$)

Hope this make sense

7
comments (5 votes)  Upvote  Down

See 1 more reply

Show more...



[◀ Comparing function growth](#)

[Big-Ω \(Big-Omega\) notation ▶](#)