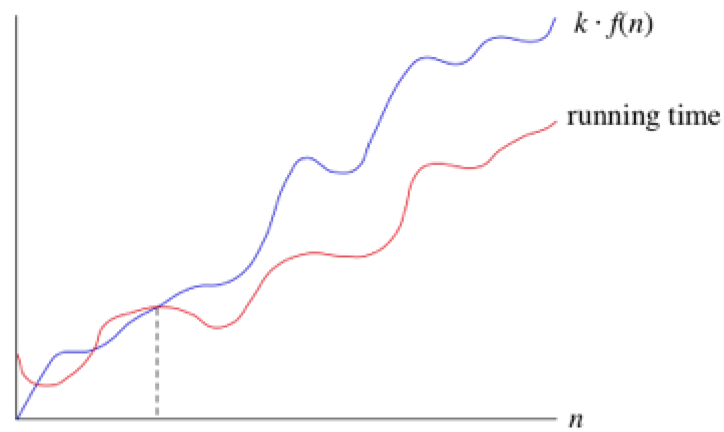
Big-O notation

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.

Sort by: Top Voted

Questions Tips & Thanks

Want to join the conversation?

You need at least 5000 energy points to get started.

jandatippetts 5 years ago

more

How does Big-Theta not describe all cases of running time? Both Big-O and Big-

Theta are limited to higher values of n, and they share the upper bound if I'm understanding correctly. How then does adding a lower bound become less precise, or in other words what makes adding a lower bound a less comprehensive description that does not cover all cases like Big-O does?

Edit: so reading the Big-Omega section I gather that Big-Theta does not refer the range between Big-O and Big-Omega. It seems that Big-Theta is the actual time (which falls between Big-O and Big-Omega), thus it only has 1 possible answer (the 'correct' answer) and does not account for all cases.

Sound good?

2 comments (47 votes) Flag more

jdsutton 5 years ago more

Big-Theta puts an upper and lower bound on all cases for n > some constant k. It describes the *rate* at which the time it takes for an algorithm to run will increase as n increases.

4 comments (23 votes) Flag more

See 8 more replies

Roman 4 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 Θ(n),Θ (n^2), Θ(n^3), and Θ (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!

1 comment

(10 votes) Flag more

Cameron 4 years ago

more

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 Θ(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 <= 1, x <= 10, x <= 100, x <= 1000, x <= 1000000.
All of these statements are true, but the most precise statement is x <= 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 Θ(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 Θ(n), Θ(n^2), Θ(n^3), and Θ(2^n).

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

Hope this makes sense

5 comments (45 votes) Flag more

See 3 more replies

William Chargin 5 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* Θ(lg n). The *best case* running time is a completely different matter, and it is Θ(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 Θ(1), and the expected and worst case times are Θ(lg n).

(20 votes) Flag more

Cameron 5 years ago

more

I would generally agree with this. I would also state that when the type of running time is not stated it is generally assumed that the type is worst case.

1 comment (15 votes) Flag more

See 2 more replies

☺ ☻ ☺Natth4545☺ ☻ ☺ 4 years ago

more

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

1 comment (2 votes) Flag more

Yahaya Aluke 4 years ago

more

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.

(15 votes) Flag more

See 4 more replies

Veronica 4 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(lgn). We can make a stronger statement about the worst-case running time: it's \Theta(\lg n)Θ(lgn). 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(lgn) time."

(3 votes) Flag more

Cameron 4 years ago

more

The worst case scenario will always take at least as long as the other scenarios e.g. worst case scenario takes at least as long as the best case scenario, and at least as long as the average case scenario.

Suppose we have an upper bound on the running times for our worst case scenario. There is no possible way for us to find a running time for any scenario (best case, average case, worst case,etc.) that exceeds that upper bound. Thus the upperbound on the running times for our worst case scenario is the upper bound on the running times for ALL scenarios.

We can say that binary search is Θ(log n) for the worst case scenario since the running time for the worst case scenario is both upper and lower bounded by log n.

We can also say that the running time is O(log n) for ALL scenarios, since the running time in the worst case scenario is O(log n). (Can't get worse than the worst)

We wouldn't be able to say Θ(log n) for ALL scenarios since the running time in the best case scenario is 1 guess, which is typically much less than log n.

Hope this makes sense

(9 votes) Flag more

See 2 more replies

c0ffeeartc 3 years ago

more

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

(6 votes) Flag more

purposenigeria 2 years ago

more

What is upper bound, lower bound and tight bound ?

(1 vote) Flag more

See 1 more reply

sparksro 4 years ago

more

I have a question on upper ane 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 examaple but in set S only 2 is the tight lower bound and only 42 is the tight upper bound. Is this correct?

(2 votes) Flag more

JaniceHolz 4 years ago

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 comments (8 votes) Flag more

See 3 more replies

mugnaio 4 years ago more

I didn't understand this: " If we say that a running time is Θ(f(n)) in a particular situation, then it's also O(f(n)). "
Binary search is Θ(1) in a particular situation (best case) but it is not O(1).

(2 votes) Flag more

Cameron 4 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 Θ(1) and as a result is also Ω(1) and O(1).

In the worst case scenario (our target is not in the array)
-binary search is Θ(log n) and as a result is also Ω(log n) and O(log n).

4 comments (5 votes) Flag more

See 2 more replies

justinj1776 4 years ago

more

Is it absolutely correct to say that binary search runs in Big-O(n)?Why couldn't we say it can run in Big-O(n^2).SInce the upperbound for binary search is 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 Big-Theta(n) can we say it runs in Big-O(n^2)?

(2 votes) Flag more

Cameron 4 years ago

more

Binary search is Θ(log n) which means that it is O(log n) and Ω(log n)

Since binary search is O(log n) it is also O(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) Flag more

See 1 more reply

William 4 years ago

more

Is Big-O also referred to as "big Omicron?" It would make sense, since we have Theta and Omega, but the text doesn't explicitly say so.

(2 votes) Flag more

Cameron 4 years ago

more

Donald Knuth called it Big Omicron in SIGACT News in 1976 when he wrote "BIG OMICRON AND BIG OMEGA AND BIG THETA", and he is a legend in computer science, but these days it is almost always referred to as Big-O or Big-Oh.

1 comment (3 votes) Flag more

See 2 more replies

Show more...

‹ Comparing function growth

Big-Ω (Big-Omega) notation ›