What does O(log n) mean exactly?

Asked 14 years, 3 months ago Modified 1 year, 8 months ago Viewed 1.6m times

2736

I am learning about Big O Notation running times and amortized times. I understand the notion of *O(n)* linear time, meaning that the size of the input affects the growth of the algorithm proportionally...and the same goes for, for example, quadratic time $O(n^2)$ etc..even algorithms, such as permutation generators, with *O(n!)* times, that grow by factorials.

For example, the following function is *O(n)* because the algorithm grows in proportion to its input *n*:

```
f(int n) {
  int i;
  for (i = 0; i < n; ++i)
    printf("%d", i);
}
```

Similarly, if there was a nested loop, the time would be $O(n^2)$.

But what exactly is *O(log n)*? For example, what does it mean to say that the height of a complete binary tree is *O(log n)*?

I do know (maybe not in great detail) what Logarithm is, in the sense that: $\log_{10} 100 = 2$, but I cannot understand how to identify a function with a logarithmic time.

algorithm **time-complexity** **big-o**

Share Edit Follow

edited Jul 28, 2020 at 21:38

Sisir
5,088 4 27 40

asked Feb 21, 2010 at 20:05

Andreas Grech
107k 100 301 362

89 A 1-node binary tree has height log2(1)+1 = 1, a 2-node tree has height log2(2)+1 = 2, a 4-node tree has height log2(4)+1 = 3, and so on. An n-node tree has height log2(n)+1, so adding nodes to the tree causes its average height to grow logarithmically. – David R Tribble Feb 21, 2010 at 22:40

46 One thing I'm seeing in most answers is that they essentially describe "O(something)" means the running time of the algorithm grows in proportion to "something". Given that you asked for "exact meaning" of "O(log n)", it's not true. That's the intuitive description of Big-Theta notation, not Big-O. O(log n) intuitively means the running time grows **at most** proportional to "log n": stackoverflow.com/questions/471199/... – Mehrdad Afshari Feb 22, 2010 at 10:42

51 I always remember divide and conquer as the example for O(log n) – RichardOD Feb 23, 2010 at 13:23

32 It's important to realize that its log base 2 (not base 10). This is because at each step in an algorithm, you remove half of your remaining choices. In computer science we almost always deal with log base

2 because we can ignore constants. However there are some exceptions (i.e. Quad Tree run times are log base 4) – Ethan May 27, 2013 at 1:33

22 @Ethan: It doesn't matter which base you are in, since base conversion is just a constant multiplication, The formula is log_b(x) = log_d(x) / log_d(b). Log_d(b) will just be a constant. – mindvirus May 27, 2013 at 1:46

32 Answers

Sorted by: Reset to default | Trending (recent votes count more)

3361

I cannot understand how to identify a function with a log time.

The most common attributes of logarithmic running-time function are that:

- the choice of the next element on which to perform some action is one of several possibilities, and
- only one will need to be chosen.

or

- the elements on which the action is performed are digits of n

This is why, for example, looking up people in a phone book is O(log n). You don't need to check *every* person in the phone book to find the right one; instead, you can simply divide-and-conquer by looking based on where their name is alphabetically, and in every section you only need to explore a subset of each section before you eventually find someone's phone number.

Of course, a bigger phone book will still take you a longer time, but it won't grow as quickly as the proportional increase in the additional size.

We can expand the phone book example to compare other kinds of operations and *their* running time. We will assume our phone book has *businesses* (the "Yellow Pages") which have unique names and *people* (the "White Pages") which may not have unique names. A phone number is assigned to at most one person or business. We will also assume that it takes constant time to flip to a specific page.

Here are the running times of some operations we might perform on the phone book, from fastest to slowest:

- **O(1) (in the worst case):** Given the page that a business's name is on and the business name, find the phone number.
- **O(1) (in the average case):** Given the page that a person's name is on and their name, find the phone number.

- **O(log n):** Given a person's name, find the phone number by picking a random point about halfway through the part of the book you haven't searched yet, then checking to see whether the person's name is at that point. Then repeat the process about halfway through the part of the book where the person's name lies. (This is a binary search for a person's name.)
- **O(n):** Find all people whose phone numbers contain the digit "5".
- **O(n):** Given a phone number, find the person or business with that number.
- **O(n log n):** There was a mix-up at the printer's office, and our phone book had all its pages inserted in a random order. Fix the ordering so that it's correct by looking at the first name on each page and then putting that page in the appropriate spot in a new, empty phone book.

For the below examples, we're now at the printer's office. Phone books are waiting to be mailed to each resident or business, and there's a sticker on each phone book identifying where it should be mailed to. Every person or business gets one phone book.

- **O(n log n):** We want to personalize the phone book, so we're going to find each person or business's name in their designated copy, then circle their name in the book and write a short thank-you note for their patronage.
- **O(n^2):** A mistake occurred at the office, and every entry in each of the phone books has an extra "0" at the end of the phone number. Take some white-out and remove each zero.
- **O(n · n!):** We're ready to load the phonebooks onto the shipping dock. Unfortunately, the robot that was supposed to load the books has gone haywire: it's putting the books onto the truck in a random order! Even worse, it loads all the books onto the truck, then checks to see if they're in the right order, and if not, it unloads them and starts over. (This is the dreaded **bogo sort**.)
- **O(n^n):** You fix the robot so that it's loading things correctly. The next day, one of your co-workers plays a prank on you and wires the loading dock robot to the automated printing systems. Every time the robot goes to load an original book, the factory printer makes a duplicate run of all the phonebooks! Fortunately, the robot's bug-detection systems are sophisticated enough that the robot doesn't try printing even more copies when it encounters a duplicate book for loading, but it still has to load every original and duplicate book that's been printed.

Share Edit Follow

edited Jan 1, 2020 at 13:03

answered Feb 21, 2010 at 20:14

John Feminella
308k 47 344 360

97 @cletus: Coincidental, I'm afraid. I picked it because phonebooks have a large N, people understand what they are and what they do, and because it's versatile as an example. Plus I got to use robots in my explanation! A win all-around. (Also, it looks like your answer was made before I was even a member on StackOverflow to begin with!) – John Feminella Feb 23, 2010 at 0:40

20 "A mistake occurred at the office, and every entry in each of the phone books has an extra "0" at the end of the phone number. Take some white-out and remove each zero." <-- this is not order N