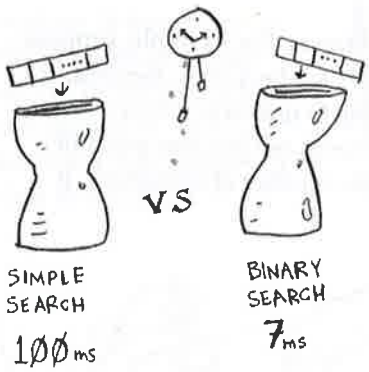
Algorithm running times grow at different rates

Bob is writing a search algorithm for NASA. His algorithm will kick in when a rocket is about to land on the Moon, and it will help calculate where to land.

This is an example of how the run time of two algorithms can grow at different rates. Bob is trying to decide between simple search and binary search. The algorithm needs to be both fast and correct. On one hand, binary search is faster. And Bob has only *10 seconds* to figure out where to land—otherwise, the rocket will be off course. On the other hand, simple search is easier to write, and there is less chance of bugs being introduced. And Bob *really* doesn't want bugs in the code to land a rocket! To be extra careful, Bob decides to time both algorithms with a list of 100 elements.

Let's assume it takes 1 millisecond to check one element. With simple search, Bob has to check 100 elements, so the search takes 100 ms to run. On the other hand, he only has to check 7 elements with binary search ($\log_2$ 100 is roughly 7), so that search takes 7 ms to run. But realistically, the list will have more like a billion elements. If it does, how long will simple search take? How long will binary search take? Make sure you have an answer for each question before reading on.



Running time for simple search vs. binary search, with a list of 100 elements

Bob runs binary search with 1 billion elements, and it takes 30 ms ($\log_2$ 1,000,000,000 is roughly 30). "30 ms!" he thinks. "Binary search is about 15 times faster than simple search, because simple search took 100 ms with 100 elements, and binary search took 7 ms. So simple search will take 30 × 15 = 450 ms, right? Way under my threshold of 10 seconds." Bob decides to go with simple search. Is that the right choice?

No. Turns out, Bob is wrong. Dead wrong. The run time for simple search with 1 billion items will be 1 billion ms, which is 11 days! The problem is, the run times for binary search and simple search *don't grow at the same rate.*

	SIMPLE SEARCH	BINARY SEARCH
100 ELEMENTS	100 ms	7 ms
10,000 ELEMENTS	10 seconds	14 ms
1,000,000,000 ELEMENTS	11 days	32 ms

Run times grow at very different speeds!

That is, as the number of items increases, binary search takes a little more time to run. But simple search takes a *lot* more time to run. So as the list of numbers gets bigger, binary search suddenly becomes a *lot* faster than simple search. Bob thought binary search was 15 times faster than simple search, but that's not correct. If the list has 1 billion items, it's more like 33 million times faster. That's why it's not enough to know how long an algorithm takes to run—you need to know how the running time increases as the list size increases. That's where Big O notation comes in.

Big O notation tells you how fast an algorithm is. For example, suppose you have a list of size n. Simple search needs to check each element, so it will take n operations. The run time in Big O notation is $O(n)$. Where are the seconds? There are none—Big O doesn't tell you the speed in seconds. *Big O notation lets you compare the number of operations.* It tells you how fast the algorithm grows.