Binary Search

Definition:

Binary search is like playing a game of "Guess Who?" with a phone book 📖. Instead of flipping through every page to find someone, you start in the middle and say, "Are you here?" 🤔 If not, you cut the book in half ✂️ and keep guessing until you find your person. It's like finding a needle in a haystack, but with a haystack that keeps shrinking! 🎯

Example 1: Searching in Dictionary

Imagine you're looking for a name in a phone book or a word in a dictionary. Instead of starting from the first page and going one by one, you jump to the middle because you know the letter you're searching for, like "K" or "O," is likely to be near the middle. This smart way of searching is similar to how binary search works: you start in the middle and narrow down your search, making it faster.

Example 2: Searching in Dictionary

When you log on to Facebook, the site needs to find your username, like "Demogorgon" in its database to verify your account. Instead of starting from the beginning and searching through all the names, Facebook jumps to the middle of the list, knowing your name is more likely to be found near the middle. This method is quicker and is similar to how binary search works.

B**inary search only works when your list is in sorted order. For example, the names in a phone book are sorted in alphabetical order, so you can**

use binary search to look for a name. What would happen if the names weren’t sorted?

Binary search only works if your list is sorted. Imagine a phone book where the names aren't in alphabetical order—like finding "John" sandwiched between "Zara" and "Albert." In that case, using binary search would be like trying to find your way with a broken compass; you'd jump to the middle and quickly get lost because the names aren't where you'd expect them to be! You'd have no choice but to flip through every page, making the search slow and frustrating.

Suppose you have a sorted list of 128 names, and you’re searching through it using binary search. What’s the maximum number of steps it would take?

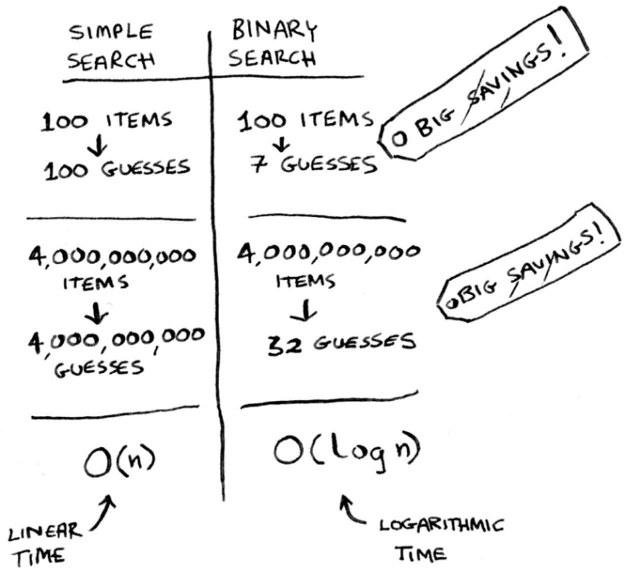
log(128) = 7

Suppose you double the size of the list. What’s the maximum number of steps now?

log(256) = 8

Running time

Binary search is different. If the list is 100 items long, it takes at most 7 guesses. If the list is 4 billion items, it takes at most 32 guesses. Powerful, eh?



Algorithm running times grow at different rates

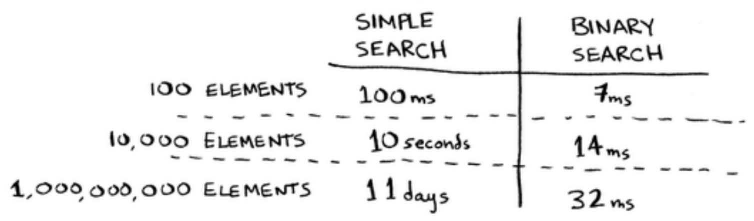
Simple Search:

Imagine you're on a rocket with a billion tiny buttons, and you need to find the right one. If you check each button one by one, it would take you **278 hours** to find the right one. 🚀🔘

Binary Search:

Now, if you can magically cut the number of buttons in half with each check, you'd find the right button in just **30 seconds**. 🚀✨

So, binary search is like having a rocket booster for finding buttons, while simple search is like crawling through space! 🪐



	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!

Some common Big O run times

1. **O(log n) (Log Time)**: **Binary Search** 🚀✨

   Imagine you're using a super-powered rocket that cuts your search space in half with each jump. Super fast!

2. **O(n) (Linear Time)**: **Simple Search** 🚀🔍

   You're searching through a billion buttons one by one. It's slow and tedious—like a long, slow spacewalk.

3. **O(n * log n) (Log-Linear Time)**: **Fast Sorting (e.g., Quicksort)** 🚀⚡

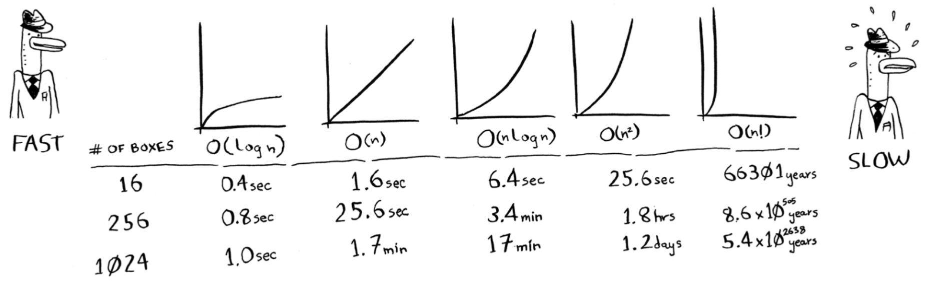
   You've got a rocket with a turbo boost that helps you sort things quickly, but it still takes a bit of time as the list grows.

4. **O(n²) (Quadratic Time)**: **Slow Sorting (e.g., Selection Sort)** 🚀🛠️

   Imagine you're doing a detailed space repair job where every part has to be checked against every other part. It takes a lot of time as the list grows.

5. **O(n!) (Factorial Time)**: **Traveling Salesperson Problem** 🚀💥

   You're trying every possible route to find the best one, which is like a never-ending, mind-boggling space mission! It's incredibly slow and overwhelming.



# OF BOXES	O(Log n)	O(n)	O(n Log n)	O(n^2)	O(n!)
16	0.4 sec	1.6 sec	6.4 sec	25.6 sec	66301 years
256	0.8 sec	25.6 sec	3.4 min	1.8 hrs	8.6×10^{505} years
1024	1.0 sec	1.7 min	17 min	1.2 days	5.4×10^{2638} years

1. Algorithm speed isn't measured in seconds, but in growth of the number of operations .Instead, we talk about how quickly the run time of an algorithm increases as the size of the input increases
2. O(log n) is faster than O(n), but it gets a lot faster as the list of items you're searching grows

Finding a person's phone number given their name in a sorted phone book:

O(log n)

If the phone book is sorted, you can use binary search to find the phone number efficiently.

Finding a person's name given their phone number in an unsorted phone book:

O(n)

You'll need to search through the entire phone book because the phone book isn't sorted by phone numbers.

Reading the phone numbers of every person in the phone book:

O(n)

You have to check each phone number, so it scales linearly with the number of people.

Reading the phone numbers of just the As:

O(n)

Even though it might seem like you only need to check a small portion of the phone book, you might still need to go through the whole book to find all the entries starting with A. If the phone book is sorted and you can quickly find the range of A's, it can be more efficient, but without additional information or preprocessing, it’s generally O(n).