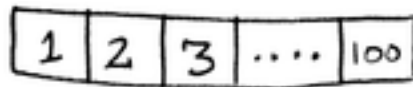


Algorithms

Analysis and Design

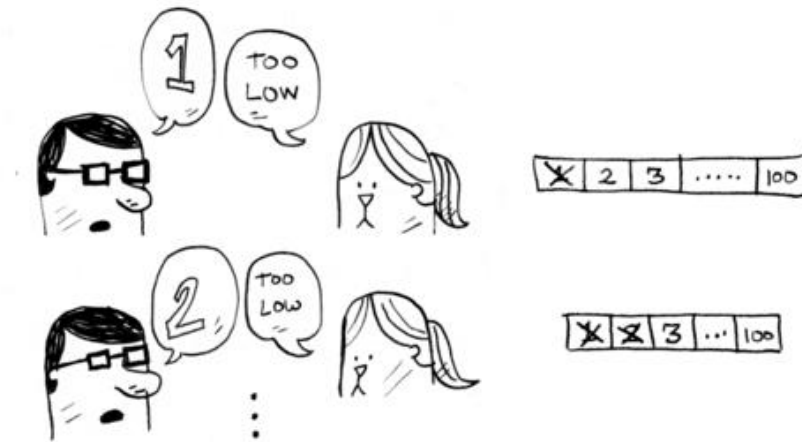
Introduction

- An *algorithm* is a set of instructions for accomplishing a task. Every piece of code could be called an algorithm.
- **What you need to know**
- You'll need to know basic algebra before starting this book. In particular, take this function: $f(x) = x \times 2$. What is $f(5)$? If you answered 10, you're set.
- **Binary search**
- Binary search is an algorithm; its input is a sorted list of elements. If an element you're looking for is in that list, binary search returns the position where it's located. Otherwise, binary search returns null.
- Here's an example of how binary search works. I'm thinking of a number between 1 and 100.



Introduction

- You have to try to guess my number in the fewest tries possible. With every guess, I'll tell you if your guess is too low, too high, or correct. Suppose you start guessing like this: 1, 2, 3, 4 Here's how it would go.



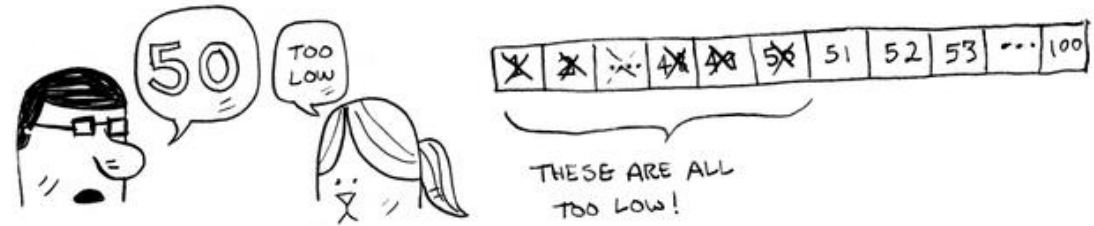
Introduction

This is *simple search* (maybe *stupid search* would be a better term). With each guess, you're eliminating only one number. If my number was 99, it could take you 99 guesses to get there!



A better way to search

Here's a better technique. Start with 50.



Too low, but you just eliminated *half* the numbers! Now you know that 1–50 are all too low. Next guess: 75.

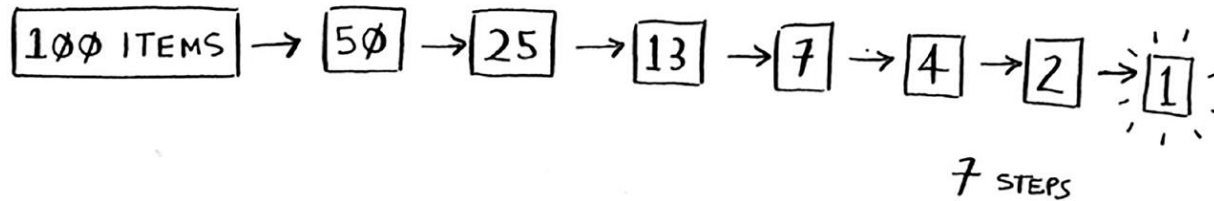


Introduction

- Too high, but again you cut down half the remaining numbers! *With binary search, you guess the middle number and eliminate half the remaining numbers every time.* Next is 63 (halfway between 50 and 75).



- This is binary search. You just learned your first algorithm! Here's how many numbers you can eliminate every time.



Introduction

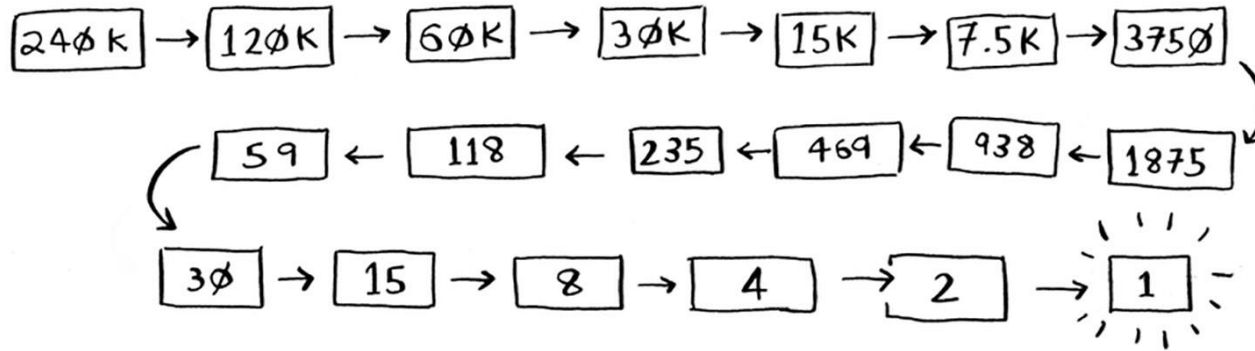
- Whatever number I'm thinking of, you can guess in a maximum of seven guesses—because you eliminate so many numbers with every guess!
- Suppose you're looking for a word in the dictionary. The dictionary has 240,000 words. *In the worst case*, how many steps do you think each search will take?

SIMPLE SEARCH: _____ STEPS

BINARY SEARCH: _____ STEPS

- Simple search could take 240,000 steps if the word you're looking for is the very last one in the book. With each step of binary search, you cut the number of words in half until you're left with only one word.

Introduction



18 STEPS

So binary search will take 18 steps—a big difference! In general, for any list of n , binary search will take $\log_2 n$ steps to run in the **worst case**, whereas simple search will take n steps.

You may not remember what logarithms are, but you probably know what exponentials are. $\log_{10} 100$ is like asking, "How many 10s do we multiply together to get 100?" The answer is 2: 10×10 . So $\log_{10} 100 = 2$. Logs are the flip of exponentials.

$$10^2 = 100 \iff \log_{10} 100 = 2$$

$$10^3 = 1000 \iff \log_{10} 1000 = 3$$

$$2^3 = 8 \iff \log_2 8 = 3$$

$$2^4 = 16 \iff \log_2 16 = 4$$

$$2^5 = 32 \iff \log_2 32 = 5$$

Big O notation

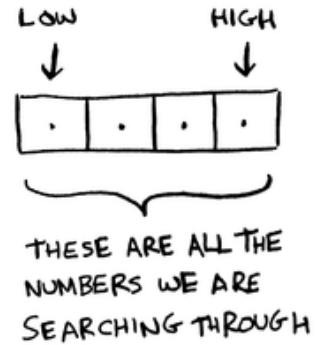
- We talk about running time in Big O notation, **log** always means **log₂**. When you search for an element using simple search, in the worst case you might have to look at every single element.
- So for a list of 8 numbers, you'd have to check 8 numbers at most.
- For binary search, you have to check **log *n*** elements in the worst case.
- For a list of 8 elements, $\log 8 == 3$, because $2^3 == 8$.
- So for a list of 8 numbers, you would have to check 3 numbers at most. For a list of 1,024 elements,
- $\log 1,024 = 10$, because $2^{10} == 1,024$. So for a list of 1,024 numbers, you'd have to check 10 numbers at most.

Binary Search

- **Note**
- Binary 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?
- Let's see how to write binary search in Python. The code sample here uses arrays. You just need to know that you can store a sequence of elements in a row of consecutive buckets called an **array**. The buckets are numbered starting with **0**: the first bucket is at position **#0**, the second is **#1**, the third is **#2**, and so on.
- The `binary_search` function takes a **sorted** array and an item. If the item is in the array, the function returns its position. You'll keep track of what part of the array you have to search through.
- At the beginning, this is the entire array:

Binary Search

- `low = 0`
- `high = len(list) - 1`

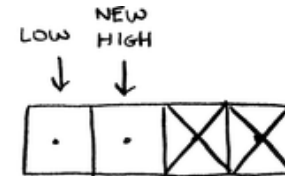


- Each time, you check the middle element:
- `mid = (low + high) / 2` ←
- `guess = list[mid]`

mid is rounded down by Python automatically if (low + high) isn't an even number.

Binary Search

- If the guess is too low, you update `low` accordingly:
- `if guess < item:`
- `low = mid + 1`
- And if the guess is too high, you update `high`. Here's the full code:



And if the guess is too high, you update `high`. Here's the full code:

```
def binary_search(list, item):  
    low = 0  
    high = len(list)-1  
  
    while low <= high:  
        mid = (low + high)  
        guess = list[mid]  
        if guess == item:  
            return mid  
        if guess > item:  
            high = mid - 1  
        else:  
            low = mid + 1  
    return None  
  
my_list = [1, 3, 5, 7, 9]  
  
print binary_search(my_list, 3) # => 1  
print binary_search(my_list, -1) # => None
```

low and high keep track of which part of the list you'll search in.

While you haven't narrowed it down to one element ...

... check the middle element.

Found the item.

The guess was too high.

The guess was too low.

The item doesn't exist.

Let's test it!

Remember, lists start at 0. The second slot has index 1.

"None" means nil in Python. It indicates that the item wasn't found.

Home works

- 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?

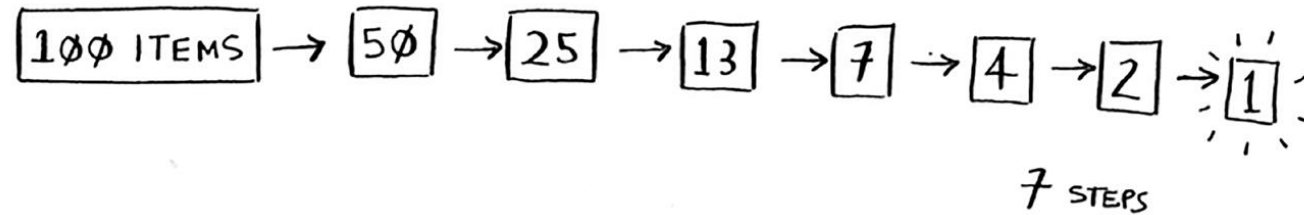
Running time

- Any time I talk about an algorithm, I'll discuss its running time. Generally you want to choose the most efficient algorithm— whether you're trying to optimize for time or space.
- Back to binary search. How much time do you save by using it?
- Well, the first approach was to check each number, one by one.
- If this is a list of 100 numbers, it takes up to 100 guesses.
- If it's a list of 4 billion numbers, it takes up to 4 billion guesses.
- So the maximum number of guesses is the same as the size of the list.
- This is called *linear time*.



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? Binary search runs in *logarithmic time*.

Running time

SIMPLE SEARCH	BINARY SEARCH
100 ITEMS ↓ 100 GUESSES	100 ITEMS ↓ 7 GUESSES
4,000,000,000 ITEMS ↓ 4,000,000,000 GUESSES	4,000,000,000 ITEMS ↓ 32 GUESSES
$O(n)$	$O(\log n)$
LINEAR TIME	LOGARITHMIC TIME

O BIG SAVINGS!

O BIG SAVINGS!

Big O notation

- Big O notation is a system for measuring the rate of growth of an algorithm. Big O notation mathematically describes the complexity of an algorithm in terms of time and space. We don't measure the speed of an algorithm in seconds (or minutes!). Instead, we measure the number of operations it takes to complete. The O is short for "Order of".

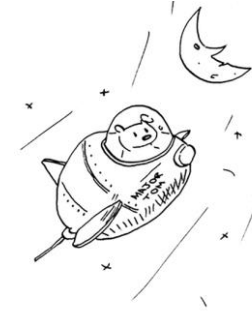
Big O notation

- **How Does Big O Work?**
- Big O notation measures the *worst-case scenario*.
- Why?
- **Because we don't know what we don't know.**
- We need to know just how poorly our algorithm will perform so we can evaluate other solutions.
- The worst-case scenario is also known as the **upper bound**. When we say upper bound, we mean the maximum number of operations performed by an algorithm.

Remember this table?

	Complexity	Rate of growth
O	constant	fast
$O(1)$	constant	
$O(\log n)$	logarithmic	
$O(n)$	linear time	
$O(n * \log n)$	log linear	
$O(n^2)$	quadratic	slow
$O(n^3)$	cubic	
$O(2^n)$	exponential	
$O(n!)$	factorial	

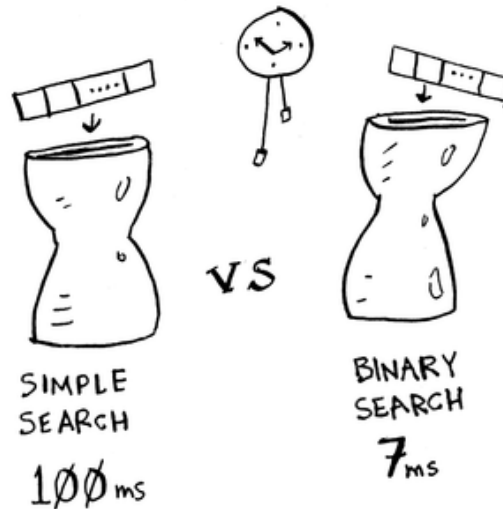
Big O notation



- *Big O* notation is special notation that tells you how fast an algorithm is.
- Who cares? Well, it turns out that you'll use other people's algorithms often—and when you do, it's nice to understand how fast or slow they are.
- 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.

Big O notation

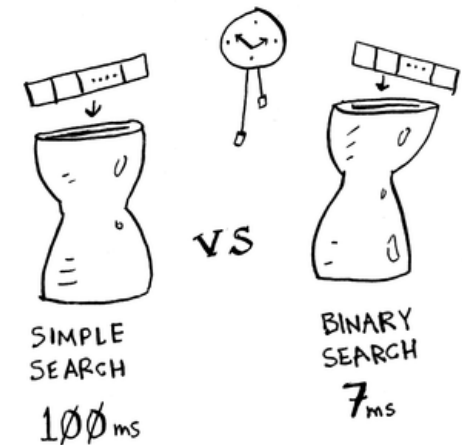
- Let's assume it takes **1 millisecond** to check one element. With simple search, Bob has to check 100 elements, so the search takes 100 ms to run.
- On the other hand, he only has to check 7 elements with binary search ($\log_2 100$ is roughly 7), so that search takes 7 ms to run.
- But realistically, 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.



Big O notation

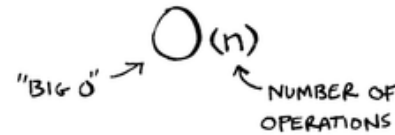
- Bob runs binary search with 1 billion elements, and it takes 30 ms ($\log_2 1,000,000,000$ is roughly 30). “32 ms!” he thinks. “Binary search is about 15 times faster than simple search, because simple search took 100 ms with 100 elements, and binary search took 7 ms. So simple search will take $30 \times 15 = 450$ ms, right? Way under my threshold of 10 seconds.” Bob decides to go with simple search. Is that the right choice?
- 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



Big O notation

- 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.

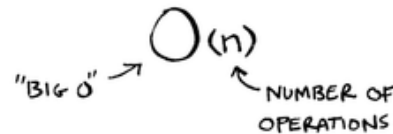


A hand-drawn diagram illustrating the components of Big O notation. It features a circle with '(n)' to its right. An arrow points from the text '"Big O"' to the circle. Another arrow points from the text 'NUMBER OF OPERATIONS' to the '(n)' part.

"Big O" → $O(n)$ ← NUMBER OF OPERATIONS

Big O notation

- 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.
- Here's another example. Binary search needs $\log n$ operations to check a list of size n . What's the running time in Big O notation? It's $O(\log n)$. In general, Big O notation is written as follows.



A hand-drawn diagram illustrating the components of Big O notation. It shows the expression $O(n)$. An arrow points from the text "BIG O" to the capital letter 'O'. Another arrow points from the text "NUMBER OF OPERATIONS" to the variable 'n' in parentheses.