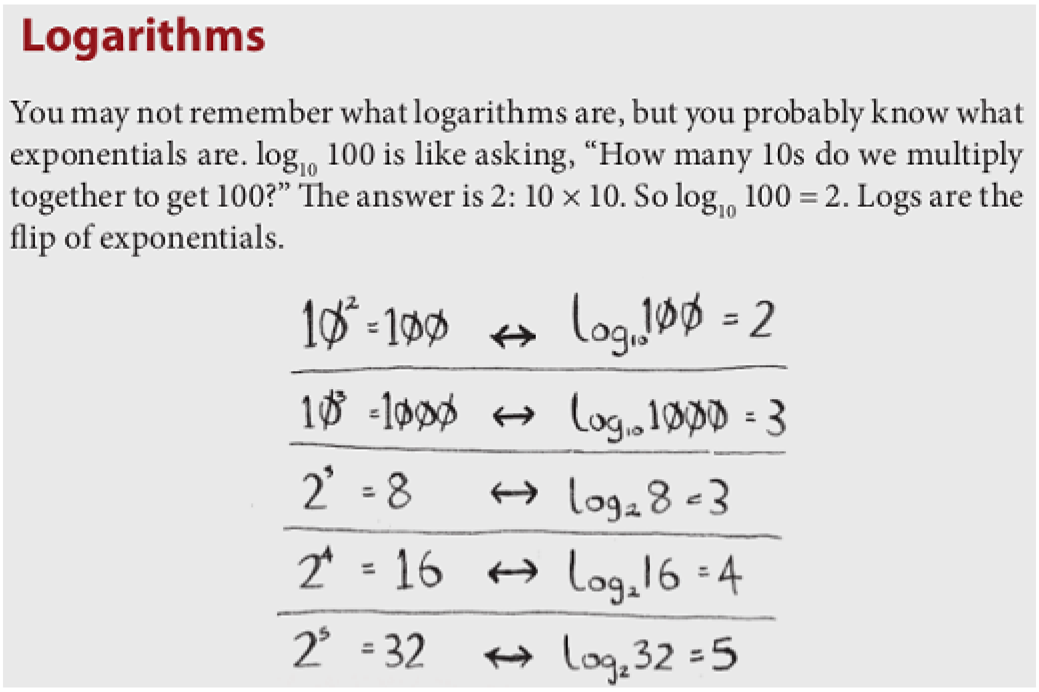
Binary Search

- Binary search is a divide-and-conquer 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**.
- Telefon rehberi, sıralı rakamlar vb. bunun örneği. Sırayla rakamları incelemektense her seferinde sıralı listenin ortasına giderek ileri mi geri mi gideceğimizi tahmin etmeliyiz.
- 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.



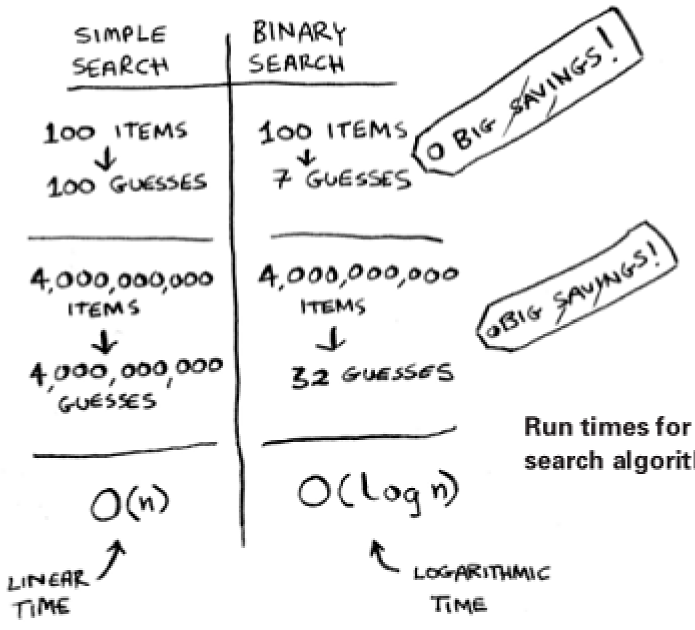
Logarithms

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.

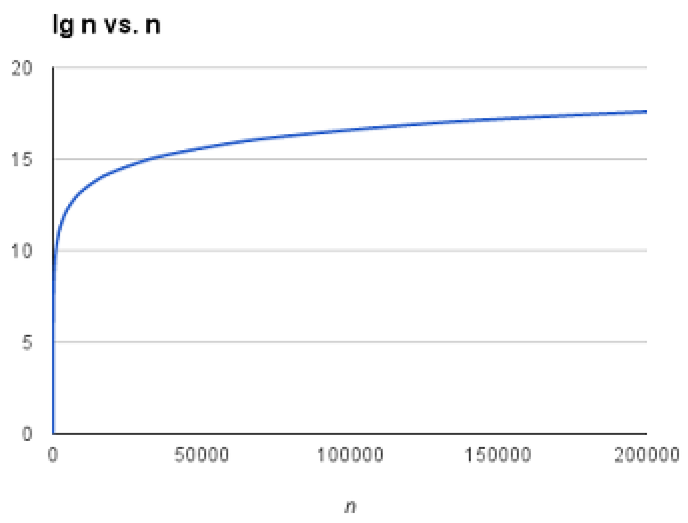
$$10^2 = 100 \leftrightarrow \log_{10} 100 = 2$$
$$10^3 = 1000 \leftrightarrow \log_{10} 1000 = 3$$
$$2^3 = 8 \leftrightarrow \log_2 8 = 3$$
$$2^4 = 16 \leftrightarrow \log_2 16 = 4$$
$$2^5 = 32 \leftrightarrow \log_2 32 = 5$$

In this book, when I talk about running time in Big O notation (explained a little later), log always means $\log_2$.

Running Time



Run times for search algorithms



The key idea is that when binary search makes an incorrect guess, the portion of the array that contains reasonable guesses is reduced by at least half. If the reasonable portion had 32 elements, then an incorrect guess cuts it down to have at most 16. Binary search halves the size of the reasonable portion upon every incorrect guess.

> According to 'Programming Pearls', only 10% of professional programmers are able to implement binary search in their code. They can explain it very well, but coding it is a challenge for them.