Asymptotic Notation: Conceptual

What is Asymptotic Notation?

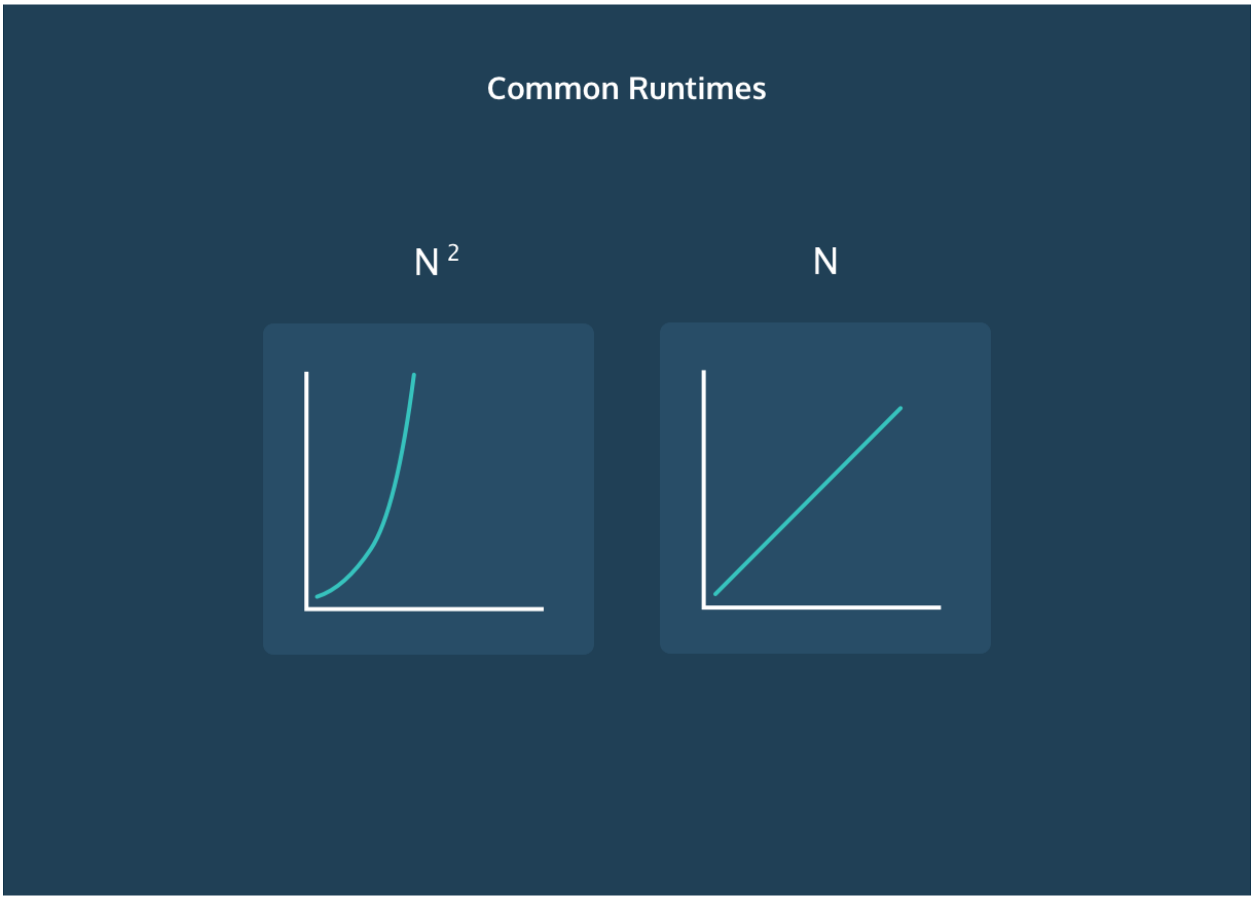
6 min

Cheetahs. Ferraris. Life. All are fast, but how do you know which one is the fastest? You can measure a cheetah's and a Ferrari's speed with a speedometer. You can measure life with years and months.

But what about computer programs? In fact, you *can* time a computer program, but different computers run at different speeds. For example, a program that takes 12 nanoseconds on one computer could take 45 milliseconds on another. Therefore, we need a more general way to gauge a program's runtime. We do this with *Asymptotic Notation*.

Instead of timing a program, through asymptotic notation, we can calculate a program's runtime by looking at how many instructions the computer has to perform based on the size of the program's input: N.

For instance, a program that has input of size N may tell the computer to run $5N^2+3N+2$ instructions. (We will get into how we get this kind of expression in future exercises.) Nevertheless, this is still a fairly messy and large expression. For asymptotic notation, we drop all of our constants (the numbers) because as N becomes extremely large, the constants will make minute differences. After changing our constants, we have N^2+N. If we take each of these terms in the expression and graph them, we see that the N^2 term grows faster than the N term.



For example, when N is 1000:

- the N^2 term is 1,000,000
- the N term is 1,000

As you can see, the N^2 term is much more significant than the N term. When N is larger than 1000, the difference becomes even more significant. Because the difference is so enormous, we don't even need to consider the N term when calculating the runtime. Thus, for this program, we would describe the runtime in terms of N^2. There are three different ways we could describe the runtime of this program: big Theta or $\Theta(N^2)$, big O or $O(N^2)$, big Omega or $\Omega(N^2)$. The difference between the three and when to use which one will be detailed in the next exercises.

You may see the term **execution count** used in evaluating algorithms. Execution count is more precise than Big O notation. The following method, addUpTo(), depending on how we count the number of operations, can be as low as 2N or as high as 5N + 2

```
public class Main() {
  void int addUpTo(int n) {
    int total = 0;
    for (int i = 1; i <= n; i++) {
      total += i;
    }
  return total;
  }
}
```

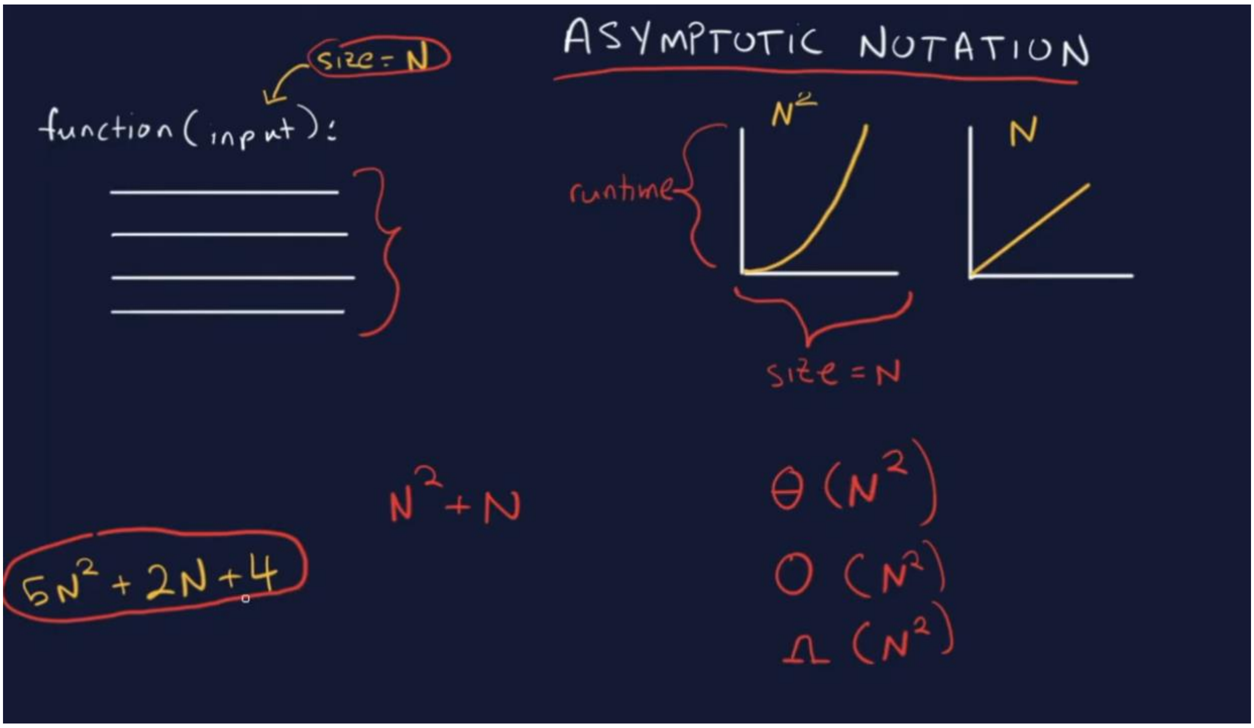
Determining execution count can increase in difficulty as our algorithms become even more sophisticated!

But regardless of the execution count, the number of operations grows roughly proportionally with n. If n doubles, the number of operations will also roughly double.

Big O Notation is a way to formalize fuzzy counting. It allows us to talk formally about how the runtime of an algorithm grows as the inputs grow. As we will see, Big O doesn't focus on the details, only the trends

Instructions

Play the video to learn about asymptotic notation in a more visual setting.


ASYMPTOTIC NOTATION
size = N
function(input):
runtime
N²
N
size = N
5N² + 2N + 4
N² + N
Θ(N²)
O(N²)
Ω(N²)