THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

COMPLEXITY

AN INTRODUCTION

CS 124 / Department of Computer Science

What is complexity?

We write computer programs to perform calculations and solve problems.

But what is the use of a program that takes a year to perform a calculation?

How does execution time vary with the size of the input?

How do we compare the efficiency of two algorithms?

Is there a theoretical limit to the complexity of a problem?

Can we do better?

What is complexity?

Is the run time of our program simply due to the way we've written our code?

Or is there something in the nature of the problem that places limits on the run time?

Is the performance of our code close to some theoretical limit?

What is complexity?

Is the problem we're trying to solve *tractable*? That is, can it be solved in some "reasonable" amount of time?

If not, we call the problem "*intractable*", and we must resort to approximations, heuristics, or restrict the problem to special cases that are tractable.

Algorithm analysis

We attempt to answer these questions through a process called algorithm analysis — whether by calculation, estimation, or theoretical consideration.

Algorithm analysis

As you can see, complexity and algorithm analysis are at the heart of computer science.

But we'll need some tools to formalize the notion of complexity.

Importantly, we'll need ways to establish *bounds* on computation resources — whether these are time or space (memory or storage).

Describing bounds

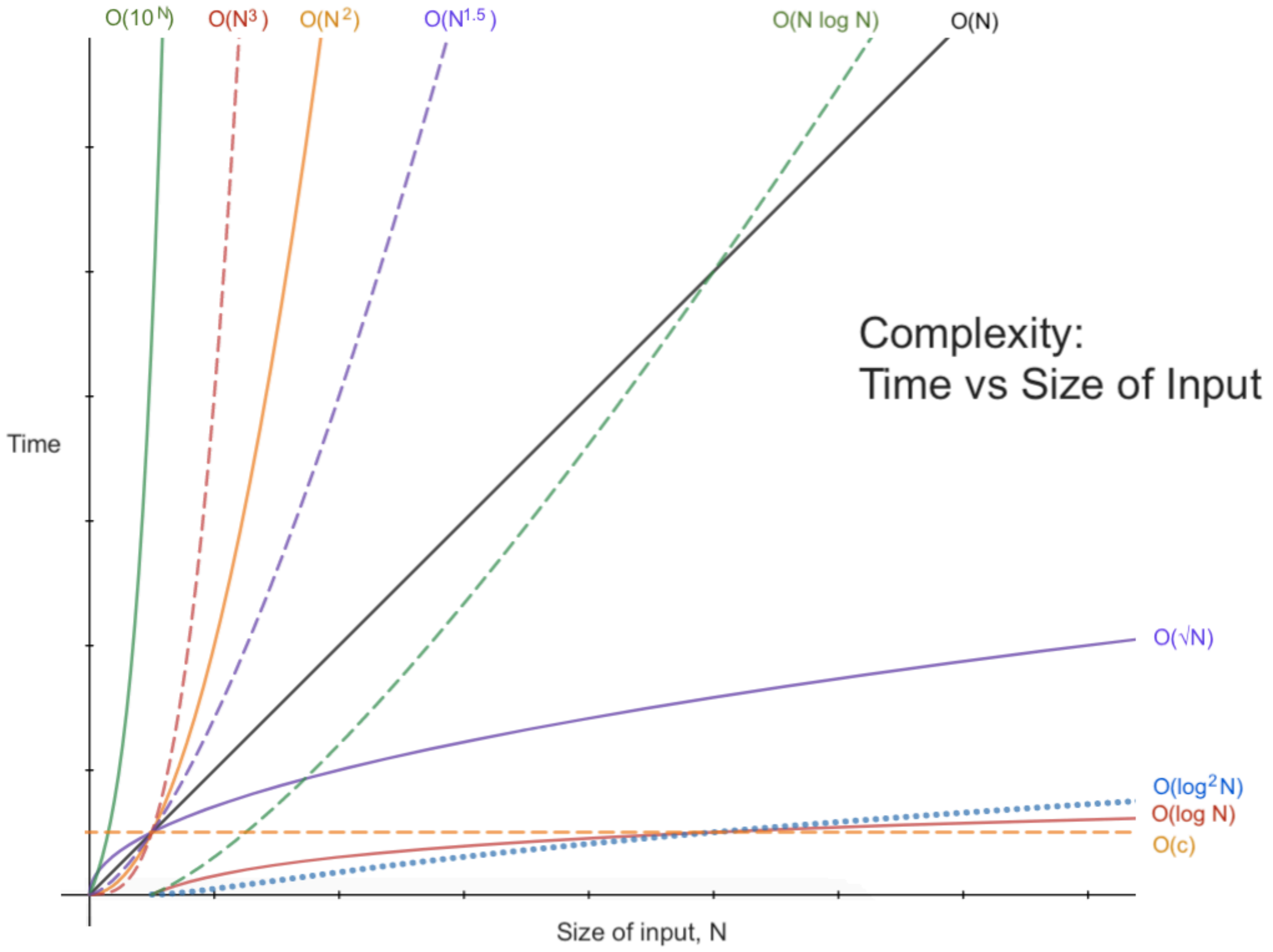
We may ask ourselves:

As the size of the input to a problem grows, what is the upper limit, or *bound*, on the time it takes to calculate a solution?

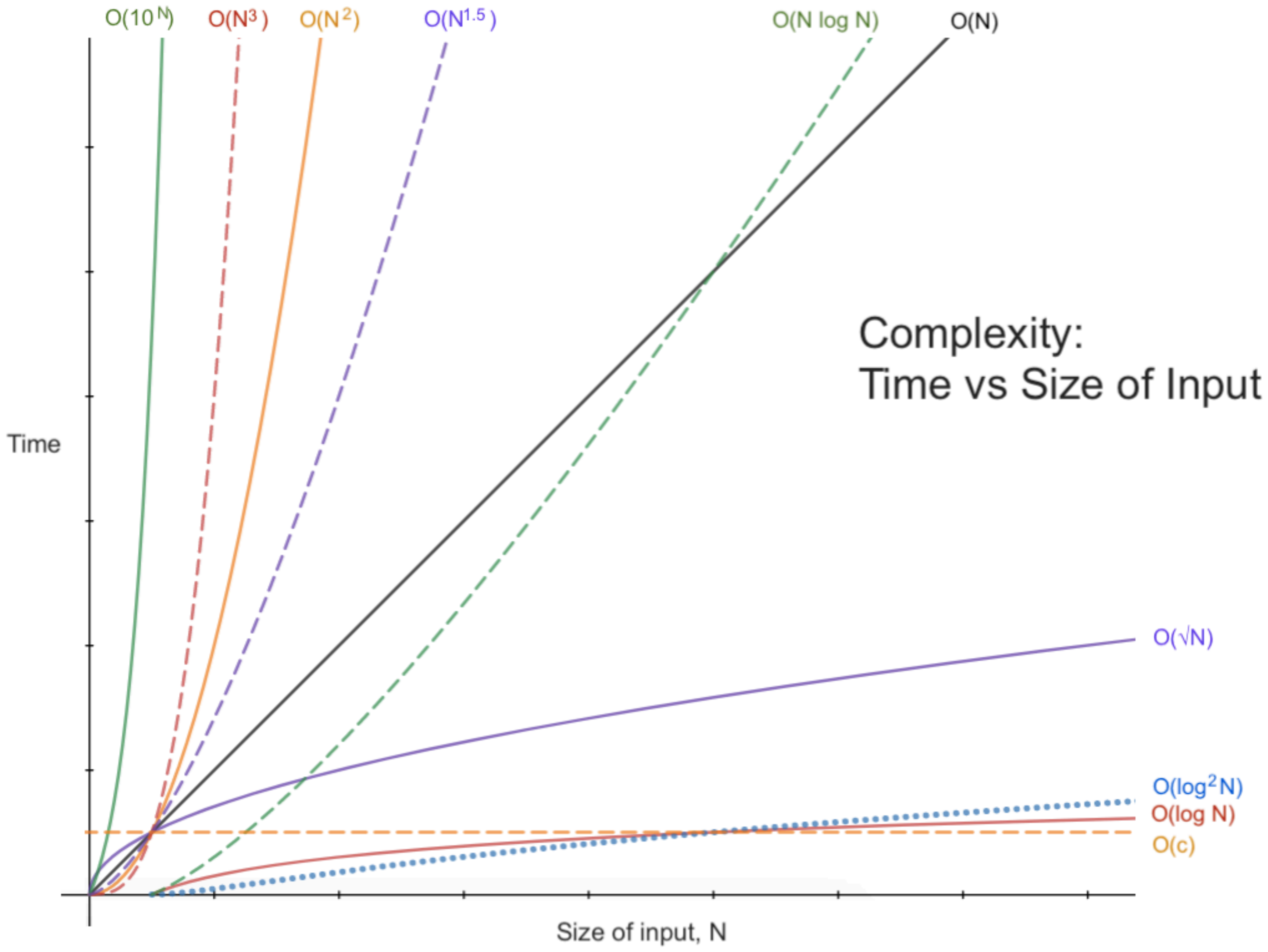
For this we use what is called *Big-O notation* — also called *asymptotic notation.*

If we say the complexity of a problem is $O(N^2)$, it means that the maximum time it takes to calculate a solution grows with the square of the size of the input.

Describing bounds



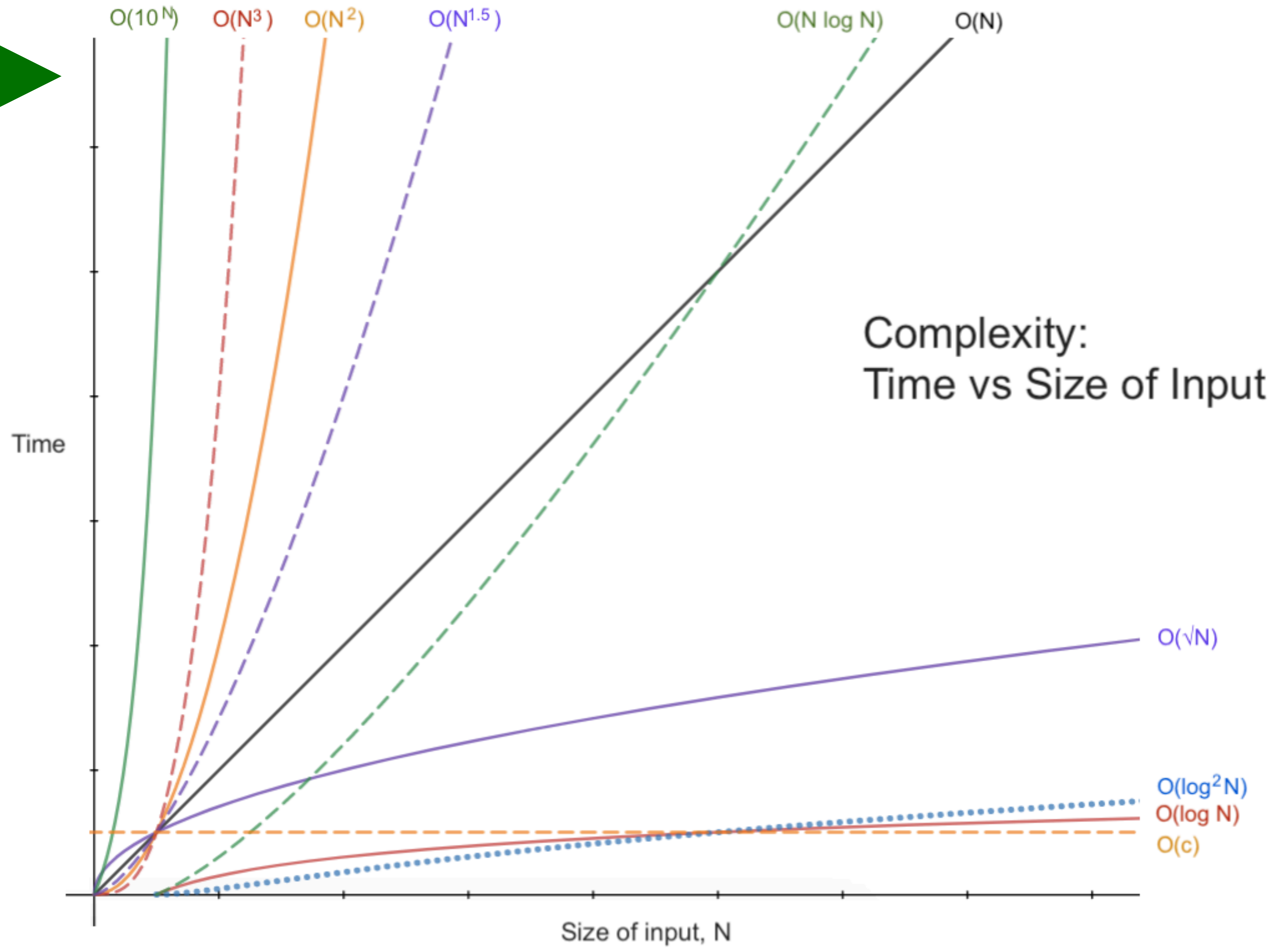
Describing bounds



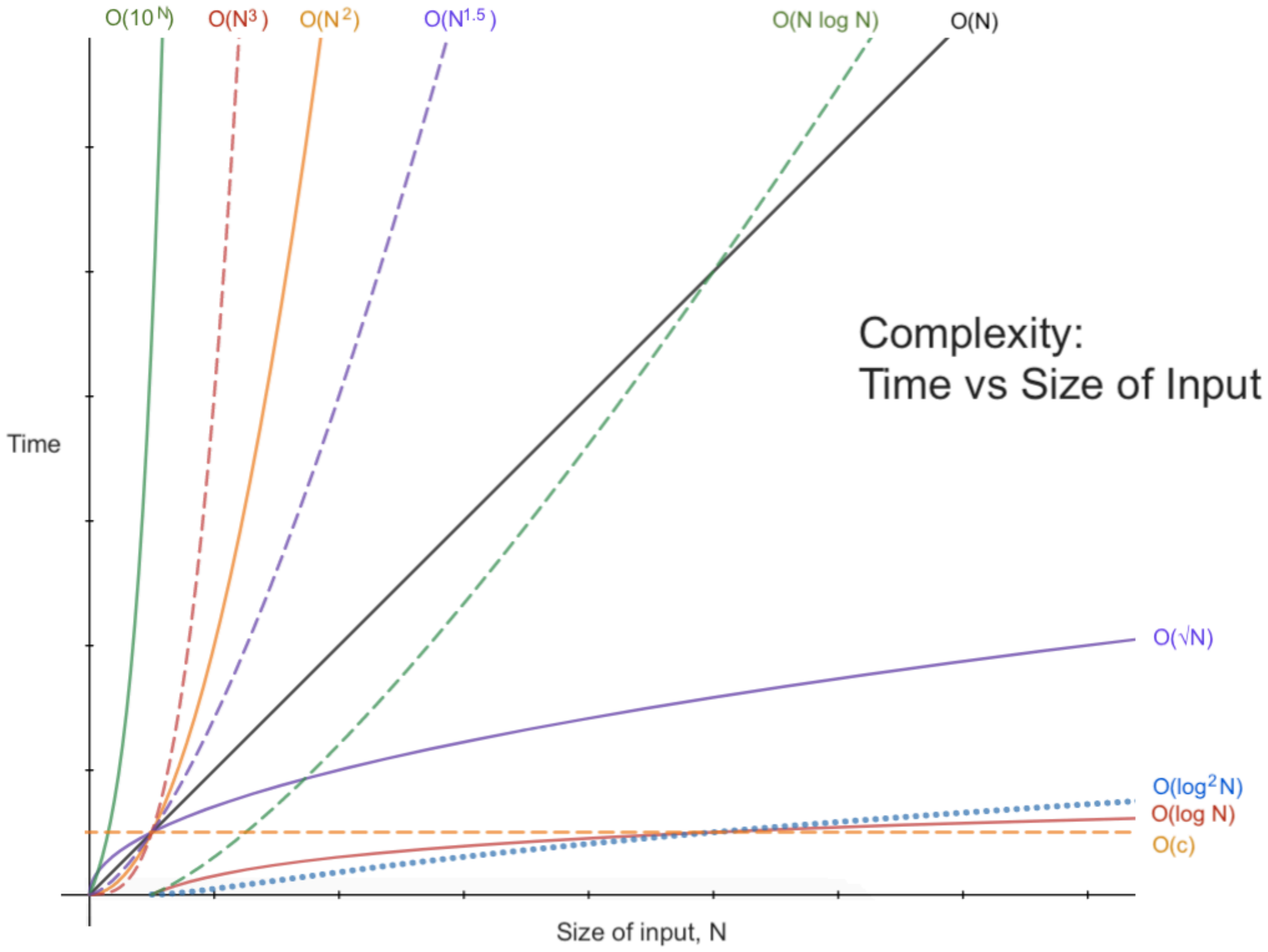
O(c)

Describing bounds

exponential



Describing bounds



Describing bounds

Some common terminology

<table>
<tr><td>2^N</td><td colspan="2">Exponential</td></tr>
<tr><td>N^3</td><td>Cubic</td><td rowspan="2">Polynomial</td></tr>
<tr><td>N^2</td><td>Quadratic</td></tr>
<tr><td>N log N</td><td colspan="2"></td></tr>
<tr><td>N</td><td colspan="2">Linear</td></tr>
<tr><td>log2 N</td><td>Log squared</td><td rowspan="3">Sublinear</td></tr>
<tr><td>log N</td><td>Log</td></tr>
<tr><td>c</td><td>Constant</td></tr>
</table>

A little formalization

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

A little formalization

Let $f(N) = N^2$.

$T(N) = O(N^2)$ if there are positive constants c and n_0 such that $T(N) \leq cN^2$ when $N \geq n_0$.

A little formalization

$T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$.

Other kinds of bounds

$T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq cg(N)$ when $N \geq n_0$.

$T(N) = \Theta(h(N))$ if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

$T(N) = o(p(N))$ if for all positive constants c there exists some n_0 such that $T(N) < cp(N)$ when $N > n_0$.

Other kinds of bounds

$T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq cg(N)$ when $N \geq n_0$.

$T(N) = \Theta(h(N))$ if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

$T(N) = o(p(N))$ if for all positive constants c there exists some n_0 such that $T(N) < cp(N)$ when $N > n_0$.

Other kinds of bounds

$T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq cg(N)$ when $N \geq n_0$.

$T(N) = \Theta(h(N))$ if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

$T(N) = o(p(N))$ if for all positive constants c there exists some n_0 such that $T(N) < cp(N)$ when $N > n_0$.

Other kinds of bounds

$T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq cg(N)$ when $N \geq n_0$.

$T(N) = \Theta(h(N))$ if and only if $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.

$T(N) = o(p(N))$ if for all positive constants c there exists some n_0 such that $T(N) < cp(N)$ when $N > n_0$.

Summary of kinds of bounds

$O(f(N))$	upper bound
$\Omega(g(N))$	lower bound
$\Theta(h(N))$	tight bound
$o(p(N))$	strict upper bound

More to follow...

In subsequent video lectures we'll present

- concrete examples of different algorithms with different run-time complexities,
- rules of thumb for calculating Big O for a given algorithm,
- rules for combining bounds, and
- a discussion of space complexity,

...and more.