CSE 250: Asymptotic Analysis

Lecture 4

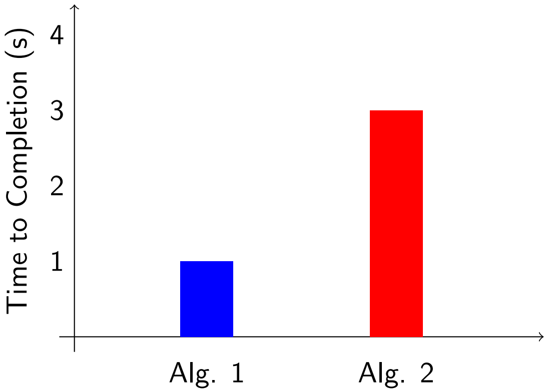
September 4, 2024

Reminders

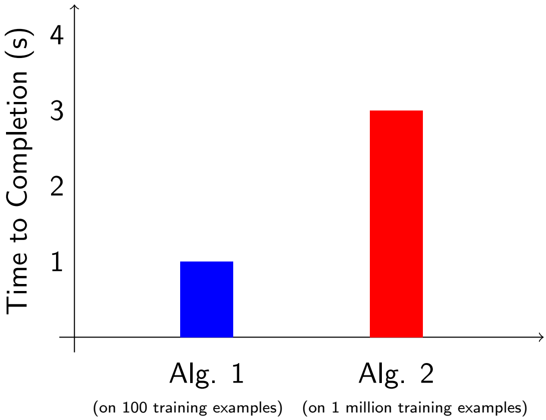
- AI Quiz due Sun, Sept 8 at 11:59 PM.
  - Your final submission must have a score of 1.0 to pass the class.
  - If you can't submit in autolab, let course staff know ASAP.
- PA0 due Sun, Sept 8 at 11:59 PM.
  - All you need to do is make sure you have a working environment.
  - If you can't submit in autolab, let course staff know ASAP.
- WA1 due Sun, Sept 8 at 11:59 PM.
  - Summations, Limits, Exponentials; Friday's Lecture

How "fast" is an algorithm?

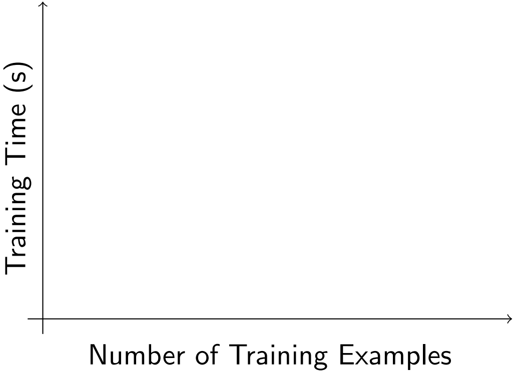
Runtime



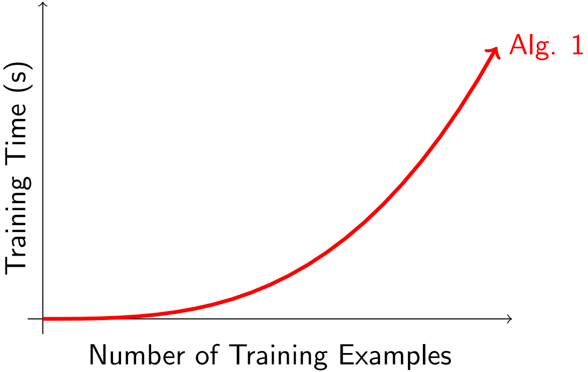
Runtime


Time to Completion (s)
4
3
2
1
Alg. 1
(on 100 training examples)
Alg. 2
(on 1 million training examples)

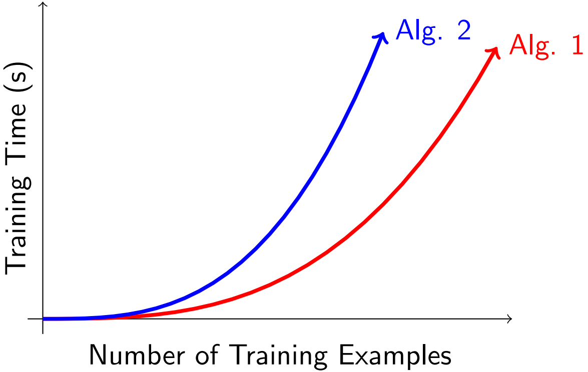
Runtime



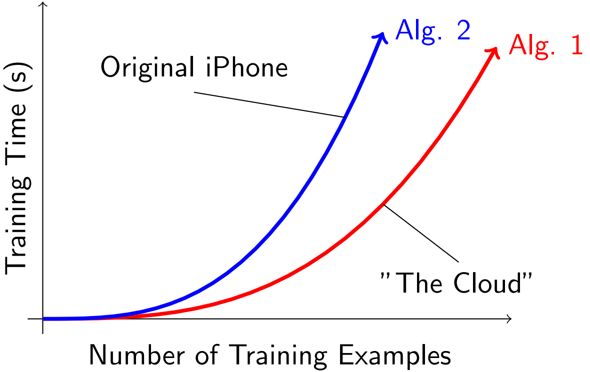
Runtime



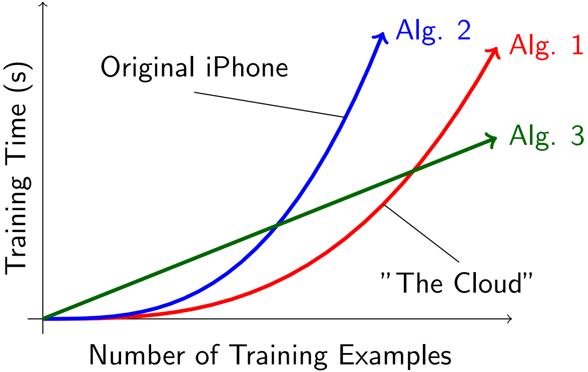
Runtime



Runtime


Alg. 2
Alg. 1
Original iPhone
Training Time (s)
"The Cloud"
Number of Training Examples

Runtime


Alg. 2
Alg. 1
Original iPhone
Alg. 3
Training Time (s)
"The Cloud"
Number of Training Examples

Implementation Variation

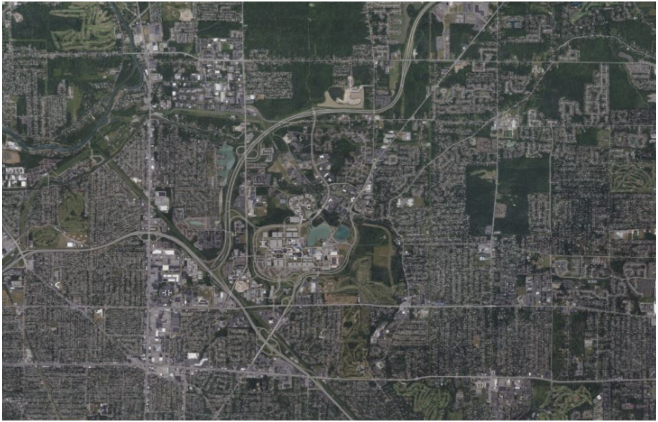
- How much data does it process?
- What hardware is it running on?
- How cleverly has the implementation been optimized?

Implementation Variation

- How much data does it process?
- What hardware is it running on?
- How cleverly has the implementation been optimized?

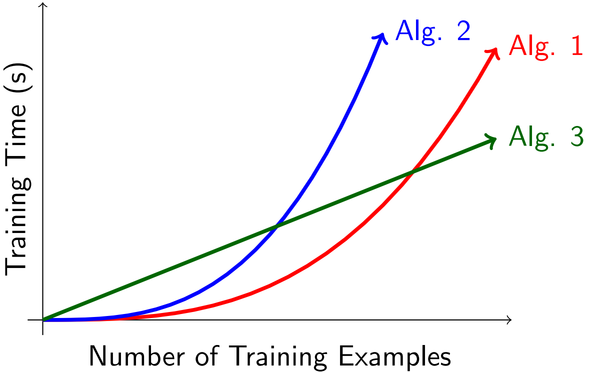
These are all (brittle) low-level details.

The Big Picture



©Earthstar Geographics SIO; via Bing Maps

Runtime



Scaling

Idea

Identify algorithms by their ???

Scaling

Idea

Identify algorithms by their "shape"

Scaling

Idea

Identify algorithms by their **"Complexity Class"**

Scaling

Idea

Identify algorithms by their **"Complexity Class"**

Quadratic is generally worse than *linear*.

- Algorithm 1 is quadratic
- Algorithm 3 is linear

Scaling

Idea

Identify algorithms by their **"Complexity Class"**

Quadratic is generally worse than *linear*.

- Algorithm 1 is quadratic
- Algorithm 3 is linear ✓

Some Notation

Some Notation

- N: The input "size"
  - How many students I have to email.
  - How many streets on a map.
  - How many key/value pairs in my dictionary

Some Notation

- N: The input "size"
  - How many students I have to email.
  - How many streets on a map.
  - How many key/value pairs in my dictionary
- $T(N)$: The runtime of 'some' implementation of the algorithm.

Some Notation

- N: The input "size"
  - How many students I have to email.
  - How many streets on a map.
  - How many key/value pairs in my dictionary
- $T(N)$: The runtime of 'some' implementation of the algorithm.
  - Some... correct implementation.

Some Notation

- N: The input "size"
  - How many students I have to email.
  - How many streets on a map.
  - How many key/value pairs in my dictionary
- $T(N)$: The runtime of 'some' implementation of the algorithm.
  - Some... correct implementation.

We care about the "shape" of $T(N)$ when you plot it.

Thinking in Steps

Instead of runtime, let's count the **'steps'**

Count the Steps

```
public void updateUsers(User[] users)
{
  x = 1;
  for(user : users)
  {
    user.id = x;
  }
}
```

Count the Steps

```
public void updateUsers(User[] users)
{
  x = 1;
  for(user : users)
  {
    user.id = x;
  }
}
```

1

Count the Steps

```java
public void updateUsers(User[] users)
{
  x = 1;
  for(user : users)
  {
    user.id = x;
  }
}
```

$$1 + \sum_{\texttt{user} \in \texttt{users}}$$

Count the Steps

```
public void updateUsers(User[] users)
{
  x = 1;
  for(user : users)
  {
    user.id = x;
  }
}
```

$$1 + \sum_{\texttt{user} \in \texttt{users}} 2 \text{ steps}$$

Count the Steps

```
public void updateUsers(User[] users)
{
  x = 1;
  for(user : users)
  {
    user.id = x;
  }
}
```

$$1 + \sum_{\texttt{user} \in \texttt{users}} 2 \text{ steps} = 1 + 2 \times |\texttt{users}|$$

... where $|\texttt{users}|$ means the size of the users array.

Count the Steps

```
public void userFullName(User[] users, int id)
{
  User user = users[id];
  String fullName = user.firstName + user.lastName;
  return fullName;
}
```

Count the Steps

```java
public void userFullName(User[] users, int id)
{
  User user = users[id];
  String fullName = user.firstName + user.lastName;
  return fullName;
}
```

3 steps

Count the Steps

```
public void userFullName(User[] users, int id)
{
  User user = users[id];
  String fullName = user.firstName + user.lastName;
  return fullName;
}
```

3 steps[1]

[1] This is actually a lie, but more on that in later lectures

Count the Steps

```
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts)
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

Count the Steps

```
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts)
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

1

Count the Steps

```
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts) ←
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

$$1 + \sum_{\texttt{post} \in \texttt{posts}}$$

Count the Steps

```java
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts)
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

$$1 + \sum_{\texttt{post} \in \texttt{posts}} \Bigg(3$$

Count the Steps

```
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts)
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

$$1 + \sum_{\texttt{post}\in\texttt{posts}} \left(3 + \sum_{\texttt{user}\in\texttt{users}} \right.$$

Count the Steps

```
public void totalReads(User[] users, Post[] posts)
{
  int totalReads = 0;
  for(post : posts)
  {
    int userReads = 0;
    for(user : users)
    {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

$$1 + \sum_{\texttt{post} \in \texttt{posts}} \left(3 + \sum_{\texttt{user} \in \texttt{users}} 2 \right)$$

Comparing Step Counts

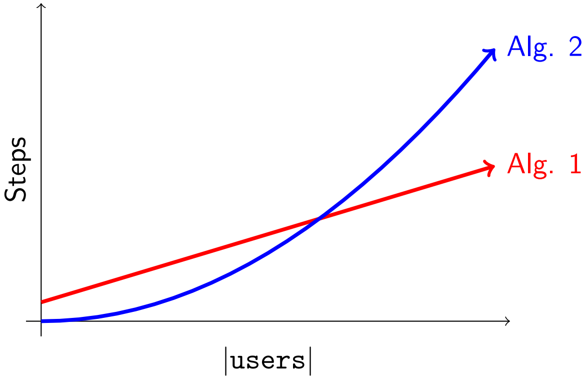
Which is better?

1. An algorithm that takes $5 + (|\texttt{users}| \times 3)$ steps
2. An algorithm that takes $\frac{1}{2}(|\texttt{users}|^2)$ steps

Comparing Step Counts

Which is better?

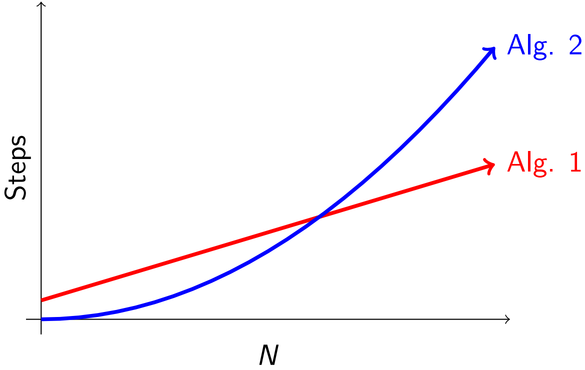
1. An algorithm that takes $5 + (|\texttt{users}| \times 3)$ steps
2. An algorithm that takes $\frac{1}{2}(|\texttt{users}|^2)$ steps



Comparing Step Counts

Which is better?

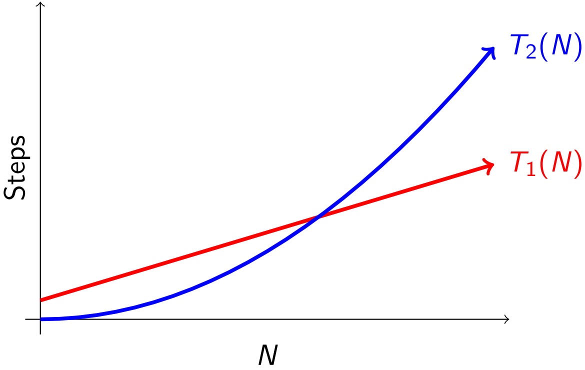
1. An algorithm that takes $5 + (N \times 3)$ steps
2. An algorithm that takes $\frac{1}{2}(N^2)$ steps



Comparing Step Counts

Which is better?

1. $T_1(N) = 5 + (N \times 3)$ steps
2. $T_2(N) = \frac{1}{2}(N^2)$ steps



Comparing Step Counts

$T_1(N) \ll T_2(N)$ (for "big enough" N).

Comparing Step Counts

$T_1(N) \ll T_2(N)$ (for "big enough" N).

So... <u>to us</u> an algorithm that takes $T_1(N)$ steps is better/faster/stronger than $T_2(N)$.

Additive Factors

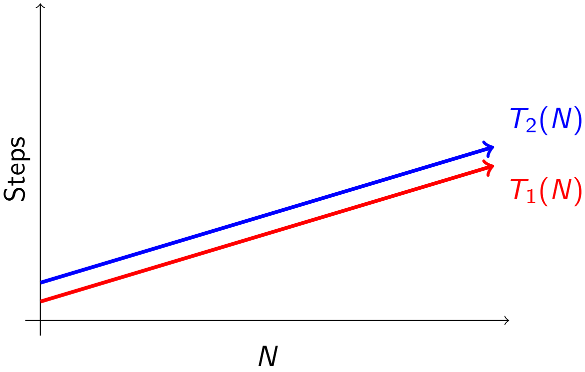
Which is better?

1. $T_1(N) = 5 + (N \times 3)$
2. $T_2(N) = 10 + (N \times 3)$

Additive Factors

Which is better?

1. $T_1(N) = 5 + (N \times 3)$
2. $T_2(N) = 10 + (N \times 3)$



Additive Factors

$T_1(N)$ is within a constant additive factor of $T_2(N)$
(i.e., $T_1(N) = T_2(N) + c$)

Additive Factors

$T_1(N)$ is within a constant <u>additive</u> factor of $T_2(N)$
(i.e., $T_1(N) = T_2(N) + c$)

In This Class

$T_1(N)$ and $T_2(N)$ are the same.

Multiplicative Factors

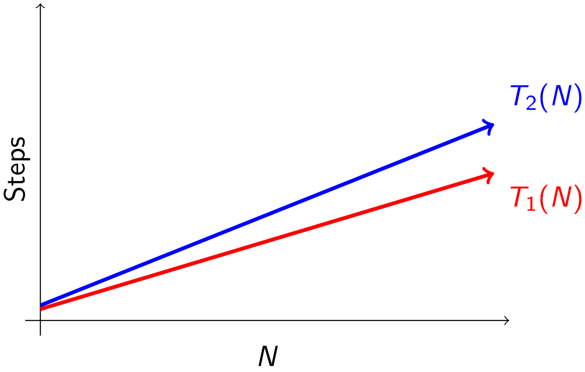
Which is better?

1. $T_1(N) = 3 + (N \times 3)$
2. $T_2(N) = 4 + (N \times 4)$

Multiplicative Factors

Which is better?

1. $T_1(N) = 3 + (N \times 3)$
2. $T_2(N) = 4 + (N \times 4)$



Multiplicative Factors

$T_1(N)$ is within a constant multiplicative factor of $T_2(N)$ (i.e., $T_1(N) = c \times T_2(N)$)

Multiplicative Factors

$T_1(N)$ is within a constant multiplicative factor of $T_2(N)$
(i.e., $T_1(N) = c \times T_2(N)$)

In This Class

$T_1(N)$ and $T_2(N)$ are the same.

Complexity

If there's a c_1 and c_2 so that $T_1(N) = c_2 + (c_1 \times T_2(N))$ then we say that T_1 is in the same **complexity class** as $T_2(N)$.

Complexity

If there's a c_1 and c_2 so that $T_1(N) = c_2 + (c_1 \times T_2(N))$ then we say that T_1 is in the same **complexity class** as $T_2(N)$[2].

[2] I'm lying to you again.

Complexity

If there's a c_1 and c_2 so that $T_1(N) = c_2 + (c_1 \times T_2(N))$ then we say that T_1 is in the same **complexity class** as $T_2(N)$[2].

[2] I'm lying to you again... slightly. More soon.

Growth Functions

"$T(N)$ is an algorithm's runtime" means:
On an input of size N the algorithm finishes in <u>exactly</u> $T(N)$ steps.

Growth Functions

"$T(N)$ is an algorithm's runtime" means:
On an input of size N the algorithm finishes in exactly $T(N)$ steps.

What is a step?

Growth Functions

"$T(N)$ is an algorithm's runtime" means:
On an input of size N the algorithm finishes in exactly $T(N)$ steps.

What is a step?

- An arithmetic operation
- Accessing a variable
- Printing a character

Growth Functions

" $T(N)$ is an algorithm's runtime" means:
On an input of size N the algorithm finishes in exactly $T(N)$ steps.

What is a step?

- An arithmetic operation
- Accessing a variable
- Printing a character

But...

How many Steps?

```
x = 10;
```

vs

```
x = 10;
y = 20;
```

How many Steps?

```
x = 10;
```

vs

```
x = 10;
y = 20;
```

1 and 2 are in the same complexity class $(2 = 1 + 1)$.

How many Steps?

```
x = 10;
```

vs

```
x = 10;
y = 20;
```

1 and 2 are in the same complexity class ($2 = 1 + 1$).

The exact number of steps doesn't matter.

Steps

A step is <u>any</u> computation that always has the same runtime.

Steps

A step is any computation that always[3] has the same runtime.

[3] Offer void where prohibited, some approximations may apply.

Growth Functions

We can make some assumptions about runtimes...

Growth Functions

We can make some assumptions about runtimes...

- The size of an input is never negative.
  $N \in \mathbb{Z}^+ \cup \{0\}$ (N is a positive integer or 0)

Growth Functions

We can make some assumptions about runtimes...

- The size of an input is never negative.
  $N \in \mathbb{Z}^+ \cup \{0\}$ (N is a positive integer or 0)
- Code never finishes before it starts.
  $T(N) \geq 0$

Growth Functions

We can make some assumptions about runtimes...

- The size of an input is never negative.
  $N \in \mathbb{Z}^+ \cup \{0\}$ (N is a positive integer or 0)
- Code never finishes before it starts.
  $T(N) \geq 0$
- Code never runs faster on bigger inputs.
  if $N_1 \leq N_2$, then $T(N_1) \leq T(N_2)$

Growth Functions

We can make some assumptions about runtimes...

- The size of an input is never negative.
  $N \in \mathbb{Z}^+ \cup \{0\}$ (N is a positive integer or 0)
- Code never finishes before it starts.
  $T(N) \geq 0$
- Code never runs faster on bigger inputs.
  if $N_1 \leq N_2$, then $T(N_1) \leq T(N_2)$
- We shouldn't allow fractional steps, but we want easy math.
  $T(N) \in \mathbb{R}^+ \cup \{0\}$ ($T(N)$ is a non-negative real.)

Growth Functions

We can make some assumptions about runtimes...

- The size of an input is never negative.
  $N \in \mathbb{Z}^+ \cup \{0\}$ (N is a positive integer or 0)
- Code never finishes before it starts.
  $T(N) \geq 0$
- Code never runs faster on bigger inputs.
  if $N_1 \leq N_2$, then $T(N_1) \leq T(N_2)$
- We shouldn't allow fractional steps, but we want easy math.
  $T(N) \in \mathbb{R}^+ \cup \{0\}$ ($T(N)$ is a non-negative real.)

We call any function T with these properties a **growth function**.

Growth Functions

When I say a **function**, I mean a mathematical expression like $1 + 2N$ (not a bit of code).

Shorthands

$$\theta(f(N))$$

Shorthands

$$\theta(f(N))$$

(all the mathematical functions in $f(N)$'s complexity class)

Shorthands

$$\theta(f(N))$$

(all the mathematical functions in $f(N)$'s complexity class)

$\theta(2 + (3 \times N)) = \{$

- $5 + (10 \times N)$
- N
- $2 \times N$
- $\ldots$

$\}$

Shorthands

$$\theta(f(N))$$

(all the mathematical functions in $f(N)$'s complexity class)

$\theta(2 + (3 \times N)) = \{$

- $5 + (10 \times N)$
- N
- $2 \times N$
- ...

$\}$

$g(N) \in \theta(f(N))$ means g and f are in the same complexity class

Shorthands

- $g(N) = \theta(f(N))$:
  Common shorthand for $g(N) \in \theta(f(N))$

Shorthands

- $g(N) = \theta(f(N))$:
  Common shorthand for $g(N) \in \theta(f(N))$
- $g(N)$ is in $\theta(f(N))$:
  Common shorthand for $g(N) \in \theta(f(N))$
- Algorithm Foo is in $\theta(f(N))$:
  Common shorthand for $T(N) \in \theta(f(N))$ where $T(N)$ is the <u>runtime</u> of Foo.

Class Names

- $\theta(1)$: Constant
- $\theta(\log(N))$: Logarithmic
- $\theta(N)$: Linear
- $\theta(N \log(N))$: Log-Linear
- $\theta(N^2)$: Quadratic
- $\theta(N^k)$ (for any $k \geq 1$): Polynomial
- $\theta(2^N)$: Exponential

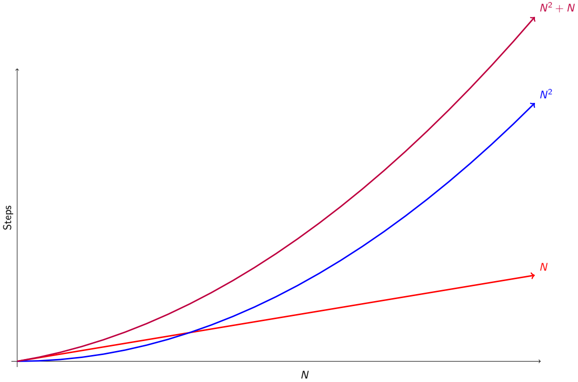
Moving forward:

- $f(N)$, $g(N)$, $f_1(N)$, $f_2(N)$, ...: Any mathematical function that's a growth function.
- $T(N)$: The growth function for a specific algorithm

Combining Classes

What class is $g(N) = N + N^2$ in?

Combining Classes



Combining Classes

For big N, $N + N^2$ looks a lot more like N^2 than N.
But it's not a constant factor different.

$$N + N^2 \neq c_1 + N^2 \times c_2$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$N^2 + N \overset{?}{\leq} 2N^2$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$
\begin{aligned}
N^2 + N &\overset{?}{\leq} 2N^2 \\
N &\overset{?}{\leq} N^2
\end{aligned}
$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$\begin{aligned} N^2 + N &\stackrel{?}{\leq} 2N^2 \\ N &\stackrel{?}{\leq} N^2 \\ 1 &\leq N \end{aligned}$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$\begin{aligned} N^2 + N &\overset{?}{\leq} 2N^2 \\ N &\overset{?}{\leq} N^2 \\ 1 &\leq N \end{aligned}$$

$$N^2 + N \overset{?}{\geq} N^2$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$
\begin{aligned}
N^2 + N &\overset{?}{\leq} 2N^2 \\
N &\overset{?}{\leq} N^2 \\
1 &\leq N
\end{aligned}
$$

$$
\begin{aligned}
N^2 + N &\overset{?}{\geq} N^2 \\
N &\geq 0
\end{aligned}
$$

Combining Classes

N^2 and $2N^2$ are in the same complexity class.

$$\begin{aligned} N^2 + N &\overset{?}{\leq} 2N^2 \\ N &\overset{?}{\leq} N^2 \\ 1 &\leq N \end{aligned}$$

$$\begin{aligned} N^2 + N &\overset{?}{\geq} N^2 \\ N &\geq 0 \end{aligned}$$

$$N^2 \leq N^2 + N \leq 2N^2$$

Complexity Bounds

$$N^2 \leq \quad N^2 + N \quad \leq 2N^2$$

Complexity Bounds

$$N^2 \leq \quad N^2 + N \quad \leq 2N^2$$

$N^2 + N$ should probably be in $\theta(N^2)$ too.

Complexity Bounds

f and g are in the same complexity class if:

Complexity Bounds

f and g are in the same complexity class if:

- g is bounded from above by something f-shaped

Complexity Bounds

f and g are in the same complexity class if:

- g is bounded from above by something f-shaped
- g is bounded from below by something f-shaped

Complexity Bounds

f and g are in the same complexity class if:

- g is bounded from above by something f-shaped
  $g(N) \in O(f(N))$
- g is bounded from below by something f-shaped

Complexity Bounds

f and g are in the same complexity class if:

- g is bounded from above by something f-shaped
  $g(N) \in O(f(N))$
- g is bounded from below by something f-shaped
  $g(N) \in \Omega(f(N))$