Intro to Algorithms

Asymptotic Notation

Sasha Golovnev

September 3, 2024

Algorithms

- Correctness
- Running Time

Algorithms

- Correctness
- Running Time

Today's Lecture

- Issues with Computing Runtime

Today's Lecture

- Issues with Computing Runtime
- Asymptotic Runtimes: Advantages and Disadvantages

Today's Lecture

- Issues with Computing Runtime
- Asymptotic Runtimes: Advantages and Disadvantages
- Big-*O* Expressions

Fibonacci numbers

Fibonacci numbers

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

Fibonacci numbers

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Exponential Growth

$$F_{20} = 6765$$

Exponential Growth

$$F_{20} = 6765$$
$$F_{50} = 12586269025$$

Exponential Growth

$$\begin{aligned} F_{20} &= 6765 \\ F_{50} &= 12586269025 \\ F_{100} &= 354224848179261915075 \end{aligned}$$

Exponential Growth

$$
\begin{aligned}
F_{20} &= 6765 \\
F_{50} &= 12586269025 \\
F_{100} &= 354224848179261915075 \\
F_{1000} &= 4346655768693745643568852767 \\
&\ 5040625802564660517371780402 \\
&\ 4817290895365554179490518904 \\
&\ 0387984007925516929592259308 \\
&\ 0322634775209689623239873322 \\
&\ 4711616429964409065331879382
\end{aligned}
$$

Exponential Growth

$$
\begin{aligned}
F_{20} &= 6765 \\
F_{50} &= 12586269025 \\
F_{100} &= 354224848179261915075 \\
F_{1000} &= 43466557686937456435688527675 \\
& 04062580256466051737178040 2 \\
& 48172908953655541794905189 04 \\
& 03879840079255169295922593 08 \\
& 03226347752096896232398733 22 \\
& \ldots 964409065331879382
\end{aligned}
$$

more than 200 digits long!

Exponential Growth

Lemma

$$2^{n/2} \leq F_n \leq 2^n \text{ for } n \geq 6\,.$$

Exponential Growth

Lemma

$$2^{n/2} \leq F_n \leq 2^n \text{ for } n \geq 6\,.$$

Proof

By Induction on n.

Computing Fibonacci numbers

Problem

Given $n \geq 1$, compute F_n.

Computing Fibonacci numbers

Problem

Given $n \geq 1$, compute F_n.

```
function FibRec(n)
```

Computing Fibonacci numbers

Problem

Given $n \geq 1$, compute F_n.

```
function FibRec(n)
if n ≤ 1:
  return n
```

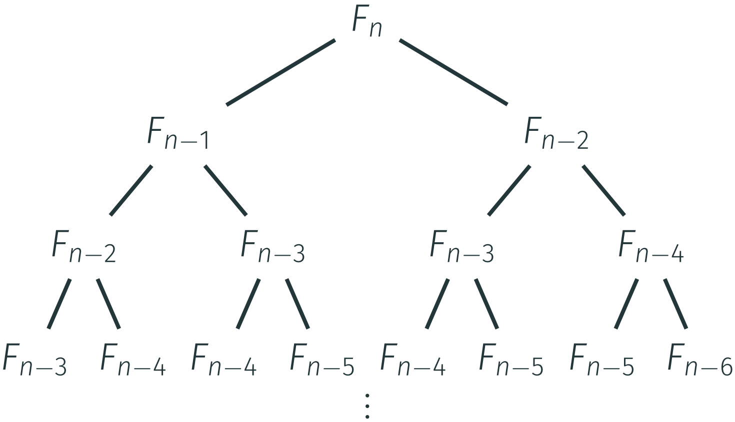
Computing Fibonacci numbers

Problem

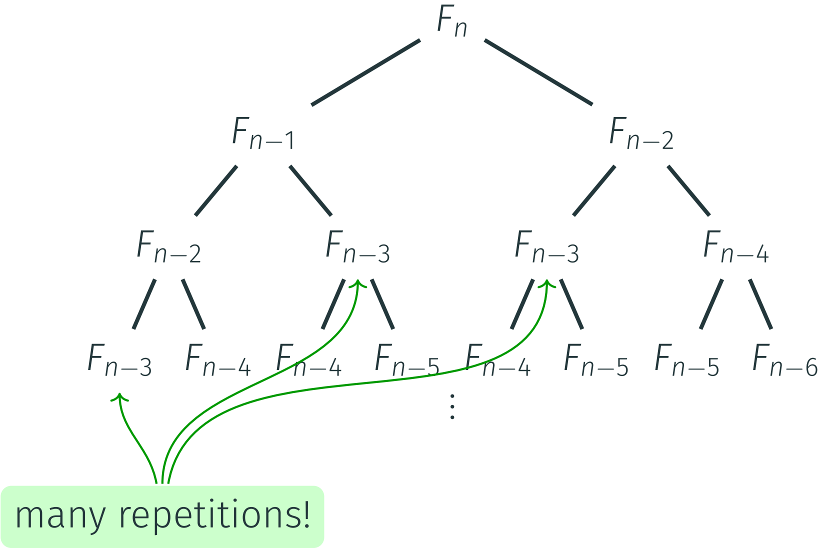
Given $n \geq 1$, compute F_n.

```
function FibRec(n)
if n ≤ 1:
  return n
return FibRec(n − 1) + FibRec(n − 2)
```

Running Time



Running Time



Running time

Let $T(n)$ denote the number of computer steps taken by `FibRec`(n). Then

$$T(n) = 2 \text{ for } n \leq 1$$

and

$$T(n) = T(n-1) + T(n-2) + 3 \text{ for } n > 1.$$

Running time

Let $T(n)$ denote the number of computer steps taken by `FibRec`(n). Then

$$T(n) = 2 \text{ for } n \leq 1$$

and

$$T(n) = T(n-1) + T(n-2) + 3 \text{ for } n > 1.$$

Hence $T(n) \geq F_n \geq 2^{n/2}$.

Running time

Let $T(n)$ denote the number of computer steps taken by `FibRec`(n). Then

$$T(n) = 2 \text{ for } n \leq 1$$

and

$$T(n) = T(n-1) + T(n-2) + 3 \text{ for } n > 1\,.$$

Hence $T(n) \geq F_n \geq 2^{n/2}$.

$$T(100) > 2.86 \cdot 10^{21}$$

Running time

Let $T(n)$ denote the number of computer steps taken by `FibRec`(n). Then

$$T(n) = 2 \text{ for } n \leq 1$$

and

$$T(n) = T(n-1) + T(n-2) + 3 \text{ for } n > 1\,.$$

Hence $T(n) \geq F_n \geq 2^{n/2}$.

$$T(100) > 2.86 \cdot 10^{21}$$

Takes more than 90,000 years to compute

FASTER ALGORITHM

```
function Fib(n)
create an array F[0 . . . n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Faster algorithm

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

$T(n) = 2n + 2$

Faster algorithm

```
function Fib(n)
create an array F[0 . . . n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
   F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

$T(n) = 2n + 2$ $\quad$ $T(100) = 202$

Visualization

http://www.cs.usfca.edu/~galles/visualization/DPFib.html

Fibonacci Numbers

- Naïve algorithm is ridiculously slow

Fibonacci Numbers

- Naïve algorithm is ridiculously slow
- Improved algorithm is so much faster

Fibonacci Numbers

- Naïve algorithm is ridiculously slow
- Improved algorithm is so much faster
- We want algorithms that are efficient for large inputs

Fibonacci Numbers

- Naïve algorithm is ridiculously slow
- Improved algorithm is so much faster
- We want algorithms that are efficient for large inputs
- The right algorithm makes all the difference

Running Time

Closer look

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Closer look

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Closer look

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i-1] + F[i-2]
return F[n]
```

Closer look

```
function Fib(n)
create an array F[0 . . . n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Closer look

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Closer look

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

Exact Running Time

- Processor's speed
- Compiler
- System Architecture
- Memory Hierarchy

Asymptotic Running Time

- All of that modifies runtime by a constant factor

Asymptotic Running Time

- All of that modifies runtime by a constant factor
- Measure runtime ignoring constants

Asymptotic Running Time

- All of that modifies runtime by a constant factor
- Measure runtime ignoring constants
- How runtime scales with n: $\sqrt{n}, n, n^2, 1.5^n, \ldots$

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 days

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 days
$n = 10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^{13}$ yr

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 days
$n = 10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^{13}$ yr
$n = 10^6$	1 sec	1 sec	17 min	

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 days
$n = 10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^{13}$ yr
$n = 10^6$	1 sec	1 sec	17 min	
$n = 10^9$	1 sec	30 sec	30 yr	

Comparing various running times and input sizes

	n	$n \log n$	n^2	2^n
$n = 20$	1 sec	1 sec	1 sec	1 sec
$n = 50$	1 sec	1 sec	1 sec	13 days
$n = 10^2$	1 sec	1 sec	1 sec	$4 \cdot 10^{13}$ yr
$n = 10^6$	1 sec	1 sec	17 min	
$n = 10^9$	1 sec	30 sec	30 yr	
max n for 1 sec	10^9	10^7	$10^{4.5}$	30

Asymptotic Running Time

- Allows us to ignore details of computer and program

Asymptotic Running Time

- Allows us to ignore details of computer and program
- Clears up notation for running time: n^3 instead of $7n^2 - 10n + \frac{n^3}{5} + \sin(x)$

Asymptotic Running Time

- Allows us to ignore details of computer and program
- Clears up notation for running time: n^3 instead of $7n^2 - 10n + \frac{n^3}{5} + \sin(x)$
- Loses (important) information about constant multiples

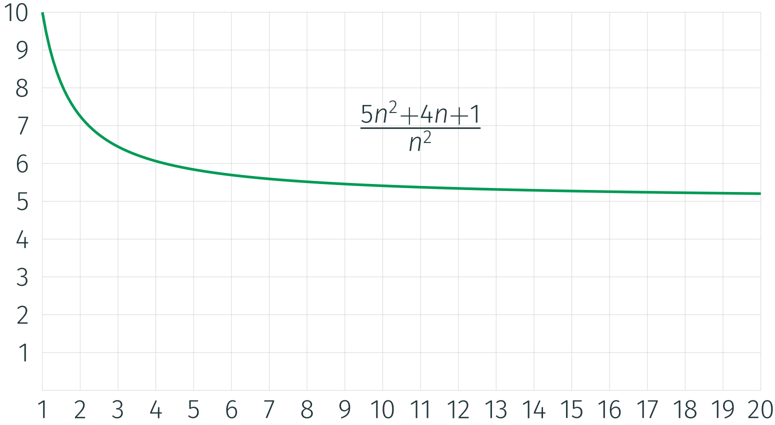
Asymptotic Running Time

- Allows us to ignore details of computer and program
- Clears up notation for running time: n^3 instead of $7n^2 - 10n + \frac{n^3}{5} + \sin(x)$
- Loses (important) information about constant multiples
- For large n, difference between n and $100n$ is insignificant comparing to difference between n and n^3

Asymptotic Notation

Growth rate

$5n^2 + 4n + 1$ has the same growth rate as n^2



Informal Definition

- $f, g\colon \mathbb{N} \to \mathbb{R}$

Informal Definition

- $f, g \colon \mathbb{N} \to \mathbb{R}$
- $f = O(g)$ (f grows no faster than g, $f \preceq g$), if there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \in \mathbb{N}$

Informal Definition

- $f, g\colon \mathbb{N} \to \mathbb{R}$
- $f = O(g)$ (f grows no faster than g, $f \preceq g$),
  if there is a constant $c > 0$ such that
  $f(n) \leq c \cdot g(n)$ for all $n \in \mathbb{N}$
- $5n^2 + 4n + 1 \leq 5n^2 + 4n^2 + n^2$

Informal Definition

- $f, g \colon \mathbb{N} \to \mathbb{R}$
- $f = O(g)$ (f grows no faster than g, $f \preceq g$),
  if there is a constant $c > 0$ such that
  $f(n) \leq c \cdot g(n)$ for all $n \in \mathbb{N}$
- $5n^2 + 4n + 1 \leq 5n^2 + 4n^2 + n^2 = 10n^2$

Informal Definition

- $f, g \colon \mathbb{N} \to \mathbb{R}$
- $f = O(g)$ (f grows no faster than g, $f \preceq g$), if there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \in \mathbb{N}$
- $5n^2 + 4n + 1 \leq 5n^2 + 4n^2 + n^2 = 10n^2 = O(n^2)$

Informal Definition

- $f, g \colon \mathbb{N} \to \mathbb{R}$
- $f = O(g)$ (f grows no faster than g, $f \preceq g$),
  if there is a constant $c > 0$ such that
  $f(n) \le c \cdot g(n)$ for all $n \in \mathbb{N}$
- $5n^2 + 4n + 1 \le 5n^2 + 4n^2 + n^2 = 10n^2 = O(n^2)$
- $5n^2 + 4n + 1 = O(n^2)$,
  $100n^2 - 5 = O(n^2)$,
  $n^2/50 - 70n = O(n^2)$

Quiz

- $n^3 + n^4 = O(n^3)$?

Quiz

- $n^3 + n^4 = O(n^3)$?
- $n^3 + n^4 = O(n^4)$?

Quiz

- $n^3 + n^4 = O(n^3)$?
- $n^3 + n^4 = O(n^4)$?
- $n^3 = O(n^4)$?

Quiz

- $n^3 + n^4 = O(n^3)$?
- $n^3 + n^4 = O(n^4)$?
- $n^3 = O(n^4)$?
- $\sqrt{n} = O(n^{6/7})$?

Quiz

- $n^3 + n^4 = O(n^3)$?
- $n^3 + n^4 = O(n^4)$?
- $n^3 = O(n^4)$?
- $\sqrt{n} = O(n^{6/7})$?
- $2^{n+1} = O(2^n)$?

Quiz

- $n^3 + n^4 = O(n^3)$?
- $n^3 + n^4 = O(n^4)$?
- $n^3 = O(n^4)$?
- $\sqrt{n} = O(n^{6/7})$?
- $2^{n+1} = O(2^n)$?
- $n \log_2 n = O(n)$?

Fibonacci Numebrs

function Fib(n)

create an array $F[0 \ldots n]$
$F[0] \leftarrow 0$, $F[1] \leftarrow 1$
for i from 2 to n:
  $F[i] \leftarrow F[i-1] + F[i-2]$
return F[n]

Fibonacci Numebrs

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
   F[i] ← F[i−1] + F[i−2]
return F[n]
```

$O(n)$ operations

Fibonacci Numebrs

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

$O(n)$ operations

$O(1)$ operations

Fibonacci Numebrs

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
  F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

$O(n)$ operations

$O(1)$ operations

$O(n)$ repetitions

Fibonacci Numebrs

```
function Fib(n)
create an array F[0 ... n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
    F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

$O(n)$ operations

$O(1)$ operations

$O(n)$ repetitions

$O(n)$ operations

Fibonacci Numebrs

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
   F[i] ← F[i−1] + F[i−2]
return F[n]
```

$O(n)$ operations

$O(1)$ operations

$O(n)$ repetitions

$O(n)$ operations

$O(1)$ operations

Fibonacci Numebrs

```
function Fib(n)
create an array F[0...n]
F[0] ← 0, F[1] ← 1
for i from 2 to n:
   F[i] ← F[i−1] + F[i−2]
return F[n]
```

$O(n)$ operations

$O(1)$ operations

$O(n)$ repetitions

$O(n)$ operations

$O(1)$ operations

Total: $O(n) + O(1) + O(n) \cdot O(n) + O(1) = O(n^2)$ operations

Missing Number

- Compute sum of all elements in stream:

$$s = x_1 + \ldots x_n$$

- Sum of all numbers in range $\{0, \ldots, n\}$ is $S = \frac{n(n+1)}{2}$
- Missing number is $S - s = \frac{n(n+1)}{2} - s$

Missing Number

- Compute sum of all elements in stream:

$$s = x_1 + \dots x_n$$

- Sum of all numbers in range $\{0, \dots, n\}$ is $S = \frac{n(n+1)}{2}$
- Missing number is $S - s = \frac{n(n+1)}{2} - s$
- Running time is $O(n)$

Majority Element

- $\mathsf{count} \leftarrow 0; \quad \mathsf{m} \leftarrow \perp$
- For each element x_i of Stream:
  - If $\mathsf{count} = 0$, then $\mathsf{m} \leftarrow x_i$ and $\mathsf{count} \leftarrow 1$
  - ElseIf $x_i = \mathsf{m}$, then count ++
  - Else count --
- Return m

Majority Element

- $\mathsf{count} \leftarrow 0$; $\mathsf{m} \leftarrow \perp$
- For each element x_i of Stream:
  - If $\mathsf{count} = 0$, then $\mathsf{m} \leftarrow x_i$ and $\mathsf{count} \leftarrow 1$
  - ElseIf $x_i = \mathsf{m}$, then $\mathsf{count}{+}{+}$
  - Else $\mathsf{count}{-}{-}$
- Return m
- Running time is $O(n)$

Common Rules

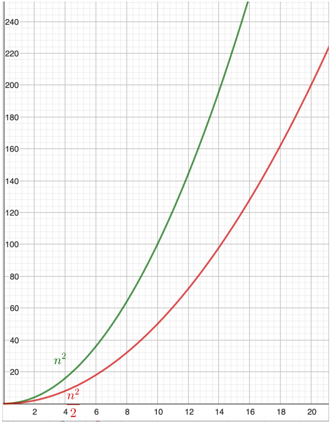
1. Multiplicative constant can be omitted:

Common Rules

1. Multiplicative constant can be omitted:
$\frac{n^2}{2} = O(n^2)$, $5n^3 = O(n^3)$

Common Rules

1. Multiplicative constant can be omitted:
$\frac{n^2}{2} = O(n^2)$, $5n^3 = O(n^3)$



Common Rules

2. Smaller terms can be omitted:

Common Rules

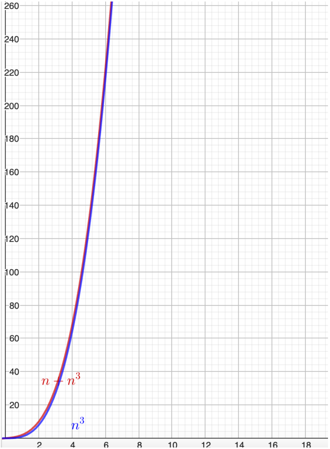
2. Smaller terms can be omitted:

$$n^3 + n = O(n^3),\ n^2 + 7n = O(n^2),\ 3n^2 + 2^n = O(2^n)$$

Common Rules

2. Smaller terms can be omitted:

$n^3 + n = O(n^3)$, $n^2 + 7n = O(n^2)$, $3n^2 + 2^n = O(2^n)$



Common Rules

3. Polynomials with larger degrees grow faster:
$$n^a \prec n^b \text{ for } 0 < a < b$$

Common Rules

3. Polynomials with larger degrees grow faster:

$$n^a \prec n^b \text{ for } 0 < a < b$$

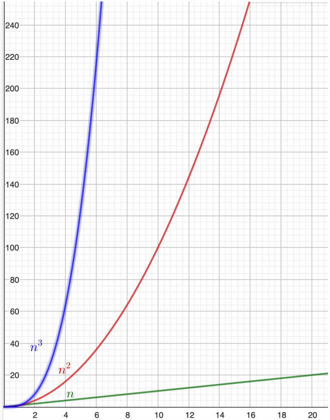
$$n = O(n^2),\ n^2 = O(n^3), \sqrt{n} = O(n)$$

Common Rules

3. Polynomials with larger degrees grow faster:

$$n^a \prec n^b \text{ for } 0 < a < b$$

$$n = O(n^2),\ n^2 = O(n^3), \sqrt{n} = O(n)$$



Common Rules

4. Polynomials grow faster than Polylogarithms: $(\log n)^a \prec n^b \ (a, b > 0)$

Common Rules

4. Polynomials grow faster than Polylogarithms:

$$(\log n)^a \prec n^b \ (a, b > 0)$$

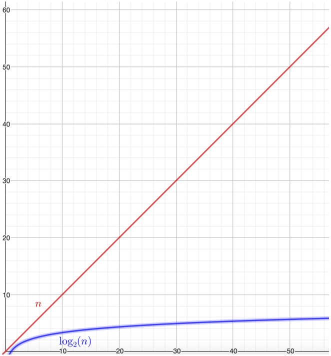
$$\log n = O(n), (\log n)^4 = O(\sqrt{n}), n \log n = O(n^2)$$

Common Rules

4. Polynomials grow faster than Polylogarithms:

$$(\log n)^a \prec n^b \ (a, b > 0)$$

$$\log n = O(n), (\log n)^4 = O(\sqrt{n}), n \log n = O(n^2)$$



Common Rules

5. Exponentials grow faster than Polynomials:
$$n^a \prec b^n \ (a > 0, b > 1)$$

Common Rules

5. Exponentials grow faster than Polynomials:

$$n^a \prec b^n \ (a > 0, b > 1)$$

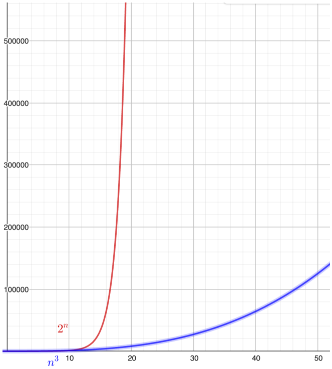
$$n^3 = O(2^n),\ n^4 = O(\sqrt{2}^n), n^{20} = O(1.1^n)$$

Common Rules

5. Exponentials grow faster than Polynomials:

$$n^a \prec b^n \ (a > 0, b > 1)$$

$$n^3 = O(2^n),\ n^4 = O(\sqrt{2}^n), n^{20} = O(1.1^n)$$



Quiz

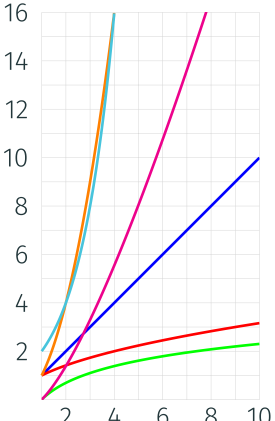
- Sort the following functions by their orders of growth
  - n^2
  - $\log n$
  - 2^n
  - $n \log n$
  - n
  - $\sqrt{n}$

Frequently used functions

$$\log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec 2^n$$

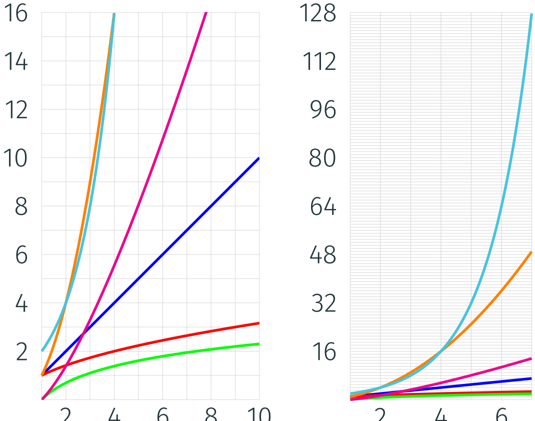
Frequently used functions

$$\log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec 2^n$$



Frequently used functions

$$\log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec 2^n$$



Formal Definition

- Is $n = O(n \log_2(n))$, is $n = O(n^2 - 1)$?

Formal Definition

- Is $n = O(n \log_2(n))$, is $n = O(n^2 - 1)$?

- $f = O(g)$ (f grows no faster than g, $f \preceq g$), if there are constants $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

Big-Omega

- $f = \Omega(g)$ (f grows no slower than g, $f \succeq g$), if $g = O(f)$

Big-Omega

- $f = \Omega(g)$ (f grows no slower than g, $f \succeq g$), if $g = O(f)$
- $f = \Omega(g)$ if there are constants $c, n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

Big-Omega

- $f = \Omega(g)$ (f grows no slower than g, $f \succeq g$), if $g = O(f)$
- $f = \Omega(g)$ if there are constants $c, n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$
- $5n^2 + 3n = \Omega(n)$, $\frac{n^3}{5} - 10n - 5 = \Omega(n^3)$

Asymptotically Tight Bounds

- $f = \Theta(g)$ (have the same growth rate, $f \asymp g$), if $f = O(g)$ and $g = O(f)$

Asymptotically Tight Bounds

- $f = \Theta(g)$ (have the same growth rate, $f \asymp g$), if $f = O(g)$ and $g = O(f)$
- $f = \Theta(g)$ if there are constants $c, C, n_0 > 0$ such that $c \cdot g(n) \leq f(n) \leq C \cdot g(n)$ for all $n \geq n_0$

Asymptotically Tight Bounds

- $f = \Theta(g)$ (have the same growth rate, $f \asymp g$), if $f = O(g)$ and $g = O(f)$
- $f = \Theta(g)$ if there are constants $c, C, n_0 > 0$ such that $c \cdot g(n) \leq f(n) \leq C \cdot g(n)$ for all $n \geq n_0$
- $17n^4 - 3n + 10 = \Theta(n^4)$

Other Notation

- $f = o(g)$ (f grows slower than g, $f \prec g$),
  if $f/g \to 0$

Other Notation

- $f = o(g)$ (f grows slower than g, $f \prec g$), if $f/g \to 0$
- $n = o(n^2)$, $n^2 = o(2^n)$

Other Notation

- $f = o(g)$ (f grows slower than g, $f \prec g$), if $f/g \to 0$
- $n = o(n^2)$, $n^2 = o(2^n)$
- $f = \omega(g)$ (f grows faster than g, $f \succ g$), if $f/g \to \infty$

Other Notation

- $f = o(g)$ (f grows slower than g, $f \prec g$), if $f/g \to 0$
- $n = o(n^2)$, $n^2 = o(2^n)$
- $f = \omega(g)$ (f grows faster than g, $f \succ g$), if $f/g \to \infty$
- $n^3 = \omega(n)$, $n = \omega(\log n)$

Order of Growth

- $\Theta(\log n)$ Logarithmic functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions
- $\Theta(n)$ Linear functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions
- $\Theta(n)$ Linear functions
- $\Theta(n \cdot \log n)$ Quasilinear functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions
- $\Theta(n)$ Linear functions
- $\Theta(n \cdot \log n)$ Quasilinear functions
- $\Theta(n^2)$ Quadratic functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions
- $\Theta(n)$ Linear functions
- $\Theta(n \cdot \log n)$ Quasilinear functions
- $\Theta(n^2)$ Quadratic functions
- $\Theta(n^c)$ for a constant $c > 0$ Polynomial functions

Order of Growth

- $\Theta(\log n)$ Logarithmic functions
- $\Theta((\log n)^c)$ Polylogarithmic functions
- $\Theta(n)$ Linear functions
- $\Theta(n \cdot \log n)$ Quasilinear functions
- $\Theta(n^2)$ Quadratic functions
- $\Theta(n^c)$ for a constant $c > 0$ Polynomial functions
- $\Theta(c^n)$ for a constant $c > 1$ Exponential functions