Overleaf (web-based Latex editor)

CS3000: Algorithms & Data
Jonathan Ullman

Lecture 2:

- Divide and Conquer: Mergesort
- Asymptotic Analysis

Jan 8, 2020

Divide and Conquer Algorithms

Divide and Conquer Algorithms

διαίρει και βασίλευε!
-Philip II of Macedon

- Split your problem into smaller subproblems
- Recursively solve each subproblem
- Combine the solutions to the subprobelms

Divide and Conquer Algorithms

- **Examples:**
  - Mergesort: sorting a list
  - Binary Search: searching in a sorted list
  - Karatsuba's Algorithm: multiplying intergers
  - Finding the median of a list
  - Fast Fourier Transform
  - …

- **Key Tools:**
  - Correctness: proof by induction
  - Running Time Analysis: recurrences
  - Asymptotic Analysis

Sorting

11	3	42	28	17	8	2	15

$A[1]$ $A[n]$

Given a list of n numbers, put them in ascending order

2	3	8	11	15	17	28	42

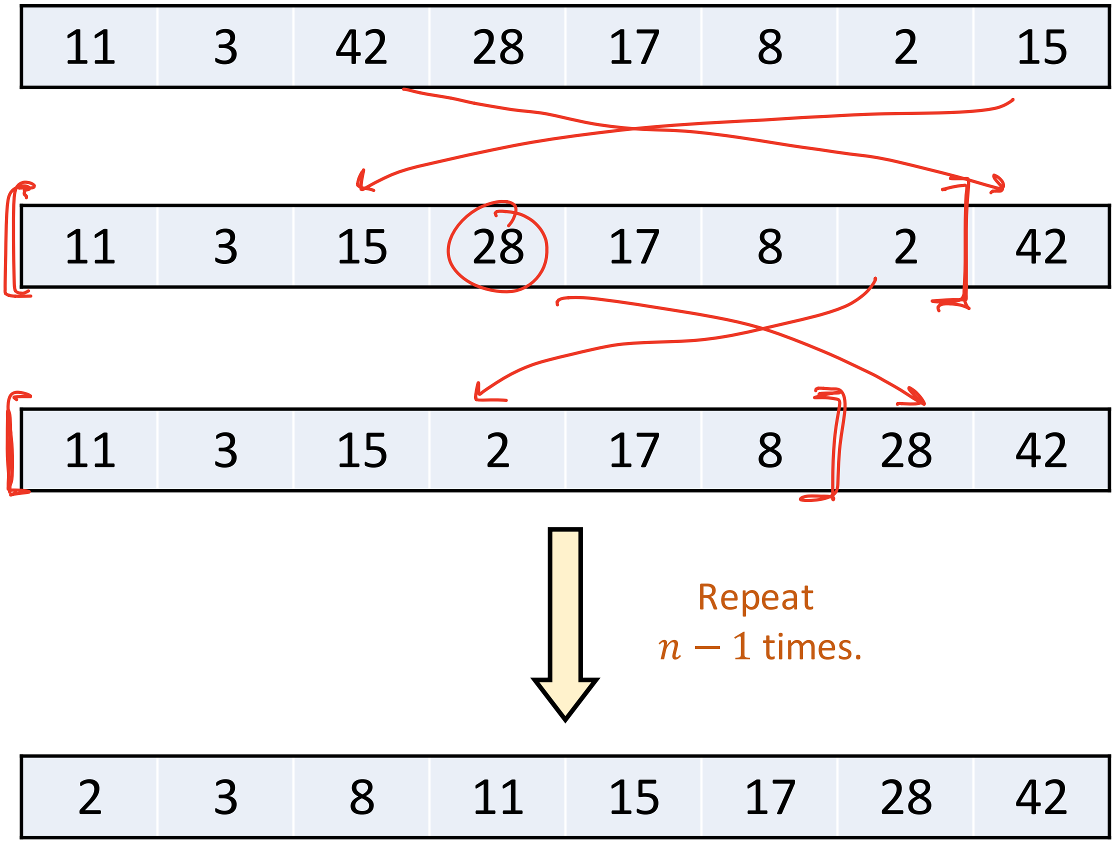
A Simple Algorithm

11	3	42	28	17	8	2	15

A Simple Algorithm: Insertion Sort

Find the maximum

Swap it into place, repeat on the rest



Repeat $n-1$ times.

A Simple Algorithm: Insertion Sort

Find the maximum

11	3	42	28	17	8	2	15

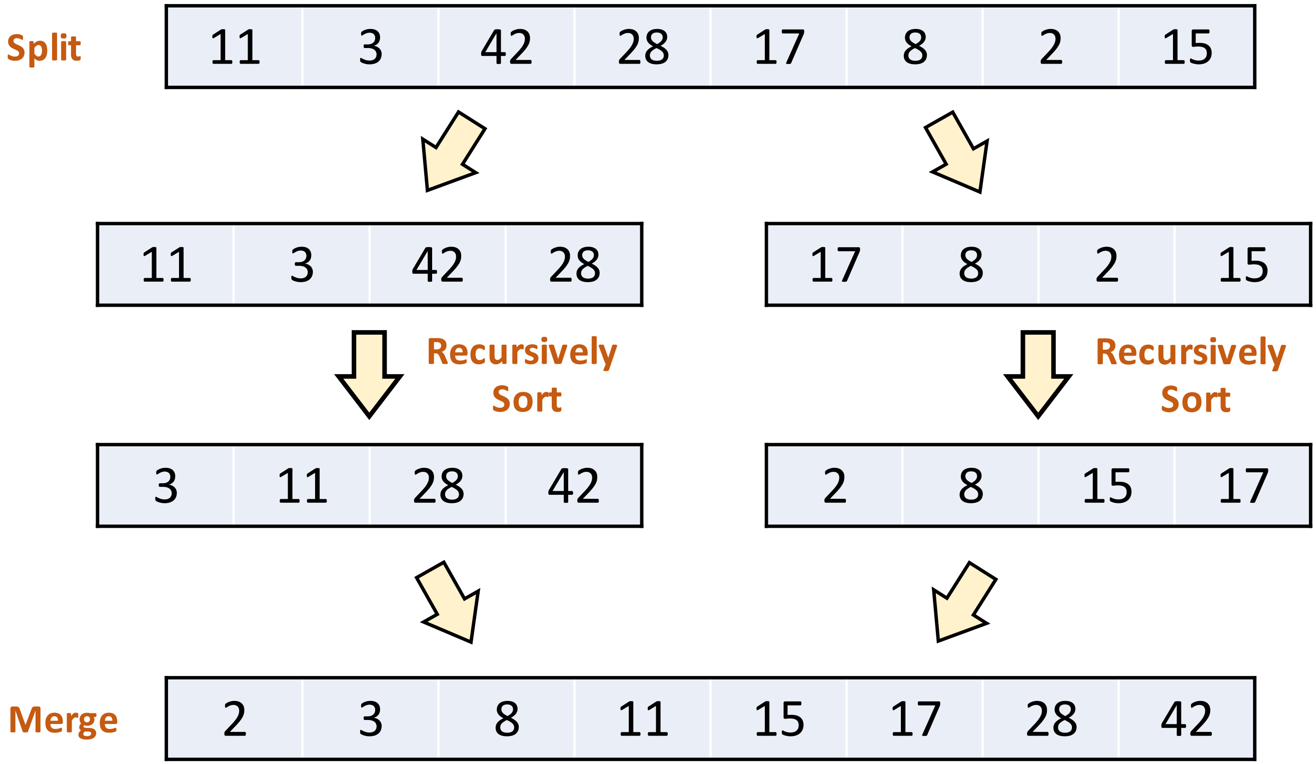
Swap it into place, repeat on the rest

11	3	15	28	17	8	2	42

Running Time:

$$\sum_{i=1}^{n-1} n-i+1 = \sum_{i=1}^{n-1} i \approx n^2$$

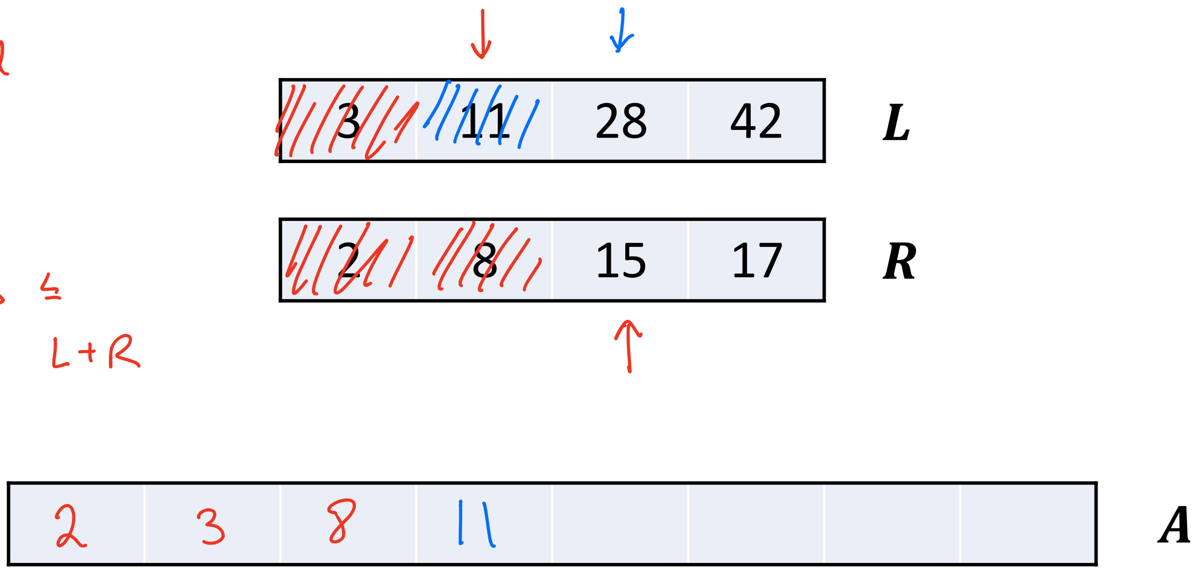
Divide and Conquer: Mergesort



Divide and Conquer: Mergesort

- **Key Idea:** If $\boldsymbol{L}, \boldsymbol{R}$ are sorted lists of length n, then we can merge them into a sorted list $\boldsymbol{A}$ of length $2n$ in time Cn
  - Merging two sorted lists is faster than sorting from scratch

- A is sorted
- L is sorted
- R is sorted
- All of A is ≤ than all of L+R



Merging

```
Merge(L,R):
  Let n ← len(L) + len(R)
  Let A be an array of length n
  j ← 1, k ← 1,

  For i = 1,…,n:
    If (j > len(L)):                // L is empty
      A[i] ← R[k], k ← k+1
    ElseIf (k > len(R)):            // R is empty
      A[i] ← L[j], j ← j+1
    ElseIf (L[j] <= R[k]):          // L is smallest
      A[i] ← L[j], j ← j+1
    Else:                           // R is smallest
      A[i] ← R[k], k ← k+1

  Return A
```

Merging

```
MergeSort(A):    Let n= len(A)
  If (len(A) = 1): Return A     // Base Case

  Let m ← ⌈len(A)/2⌉            // Split
  Let L ← A[1:m], R ← A[m+1:n]

  Let L ← MergeSort(L)          // Recurse
  Let R ← MergeSort(R)

  Let A ← Merge(L,R)            // Merge

  Return A
```

Correctness of Mergesort

- **Claim:** The algorithm **`Mergesort`** is correct

(a) Argue that Merge works. If L, R are sorted lists then Merge(L, R) are sorted.

...

(b) Prove that Mergesort works. For every list A, Mergesort(A) returns A in sorted order.

Stmt: For every $n \in \mathbb{N}$, for every list A of size n, Mergesort(A) works.

Inductive Hyp. : $H(n)$: $\forall$ list A of size $\leq n$, $MS(A)$ works

Base Case : $H(1)$ is true "obviously."

Inductive Step: $\forall\, n,\ H(n) \Rightarrow H(n+1)$

$$\begin{matrix} H(1) \\ H(2) \\ \vdots \\ H(n) \end{matrix} \Longrightarrow H(n+1)$$

Let A be a list of size $n+1$

L, R are lists of size $\leq n$

$L = MS(L)$ is sorted, $R = MS(R)$ is sorted (IH)

$A = \text{Merge}(L, R)$ is sorted (Part (a))

(a) We will argue that throughout the execution of Merge the following are all true:

. . .

- A, L, R are sorted
- A ≤ everything in L+R

At the start these are true because ...

At each step they remain true because ...

Running Time of Mergesort

$T(n)$: the running time on inputs of size n

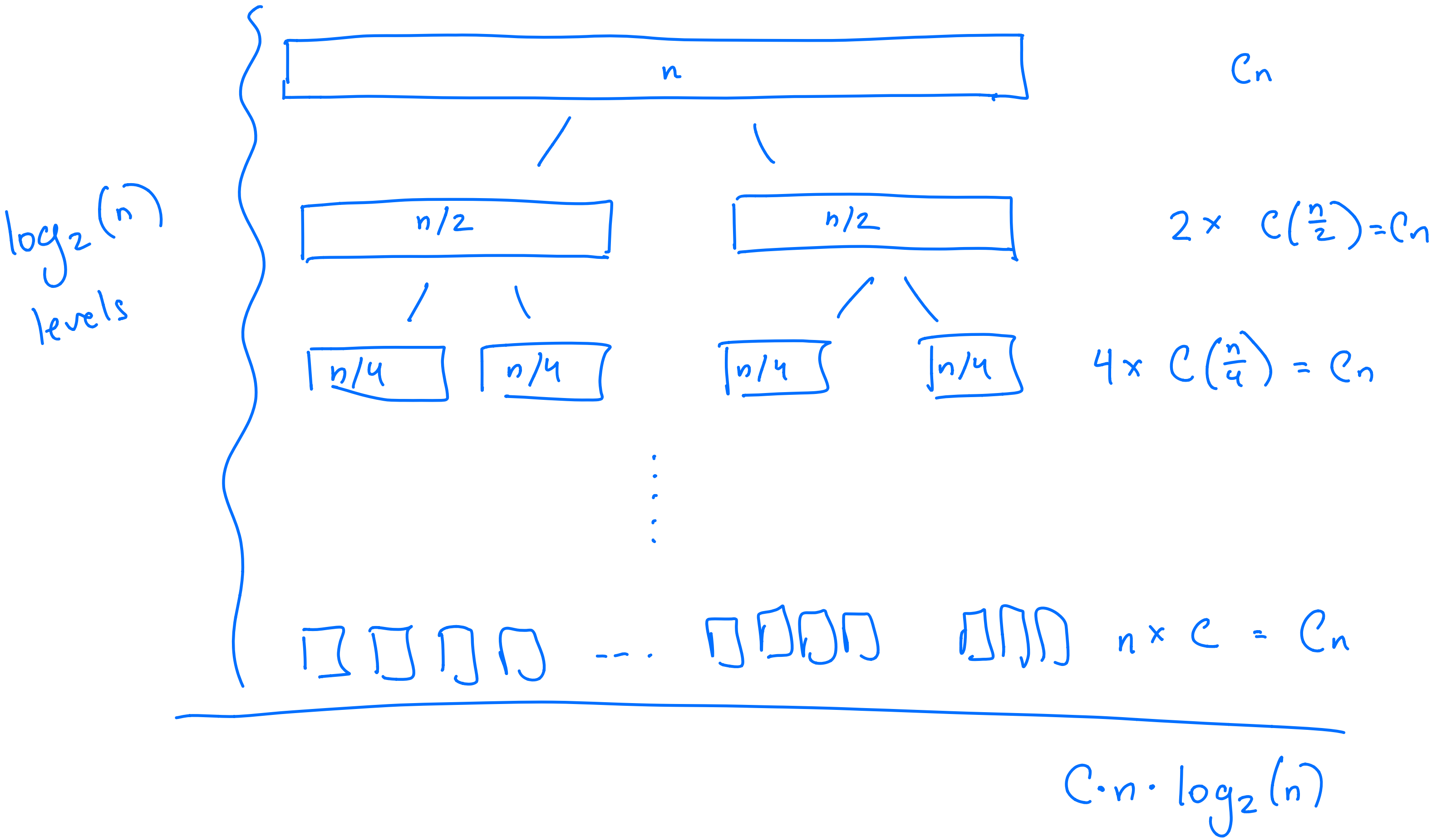
$$T(n) = 2 \cdot T\left(\left\lceil \frac{n}{2} \right\rceil\right) + Cn$$

$$T(1) = C$$

```
MergeSort(A):
  If (n = 1): Return A                 1

  Let m ← ⌈n/2⌉                        1
  Let   L ← A[1:m]                     Cn
        R ← A[m+1:n]

  Let L ← MergeSort(L)                 T(⌈n/2⌉)
  Let R ← MergeSort(R)                 T(⌈n/2⌉)
  Let A ← Merge(L,R)                   Cn

  Return A
```

Recursion Trees

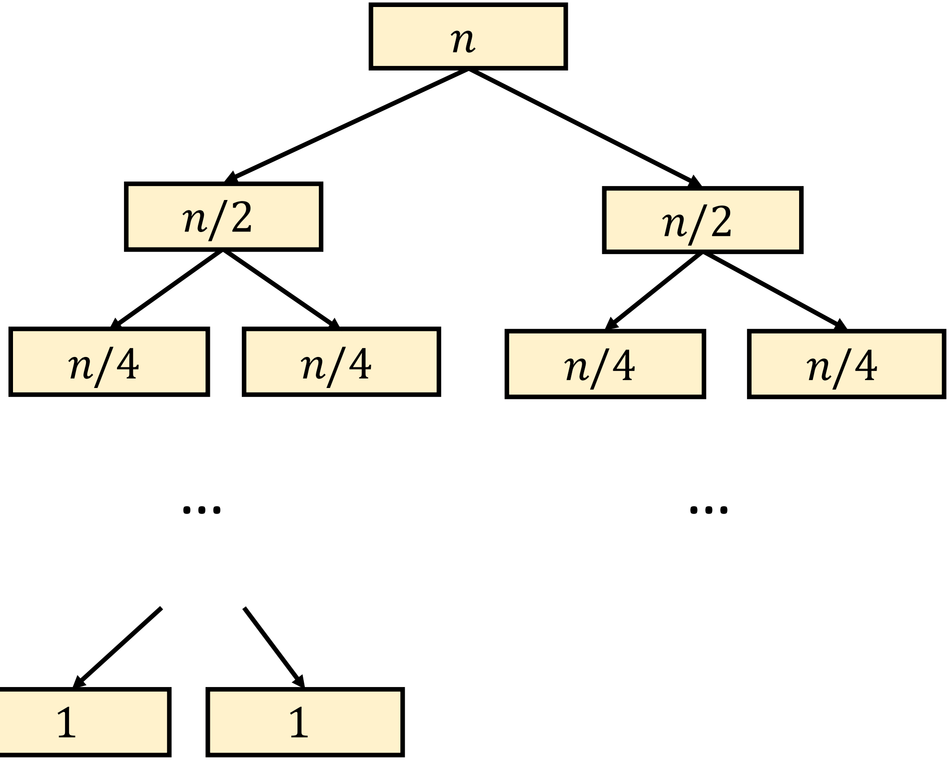
$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$



Recursion Trees

$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$

Input Size at this Level



Level	Work at this Level
0	Cn
1	$2 \cdot \left(\frac{Cn}{2}\right) = Cn$
2	$4 \cdot \left(\frac{Cn}{4}\right) = Cn$
i	$2^i \cdot \left(\frac{Cn}{2^i}\right) = Cn$
$\log_2 n$	$2^{\log_2 n} \cdot C = Cn$

Proof by Induction

$$T(n) = 2 \cdot T(n/2) + Cn$$
$$T(1) = C$$

- **Claim:** $T(n) = Cn \log_2 2n$

Mergesort Summary

- Sort a list of n numbers in $Cn \log_2 2n$ time
  - Can actually sort anything that allows comparisons
  - No comparison based algorithm can be faster
- Divide-and-conquer
  - Break the list into two halves, sort each one and merge
  - Key Fact: Merging is easier than sorting
- Proof of correctness
  - Proof by induction
- Analysis of running time
  - Recurrences

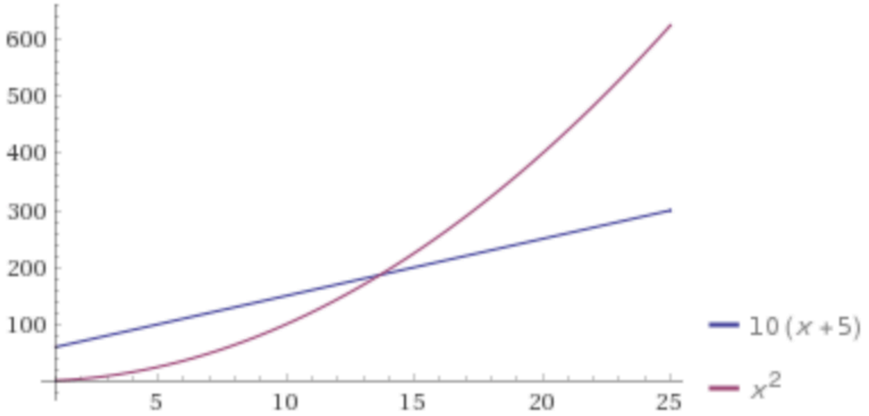
Asymptotic Analysis

Analyzing Running Time

- We don’t have enough information to precisely predict the running time of an algorithm
  - The running time will depend on the size of the input
  - The running time might depend on the input itself
  - Don’t know what machine will run the algorithm
  - Impractical to precisely count operations

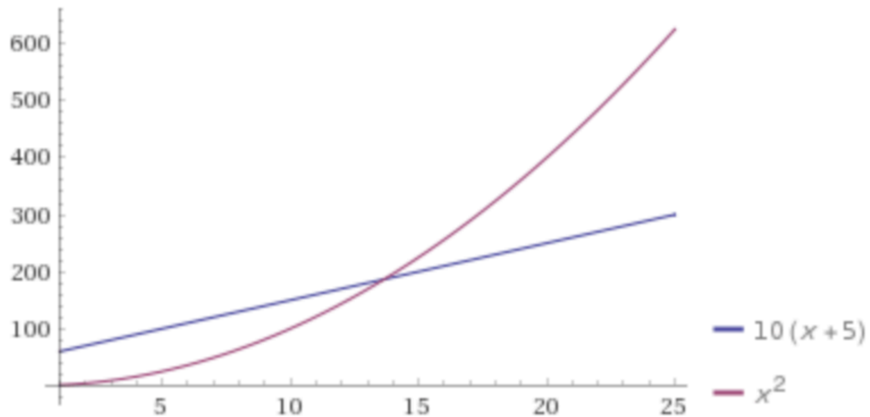
Asymptotic Order Of Growth

- Do we really need to worry about this problem?
  - Mostly we want to compare algorithms, so we can select the right one for the job
  - Mostly we don't care about small inputs, we care about how the algorithm will scale



Asymptotic Order Of Growth

- **Asymptotic Analysis:** How does the running time grow as the size of the input grows?



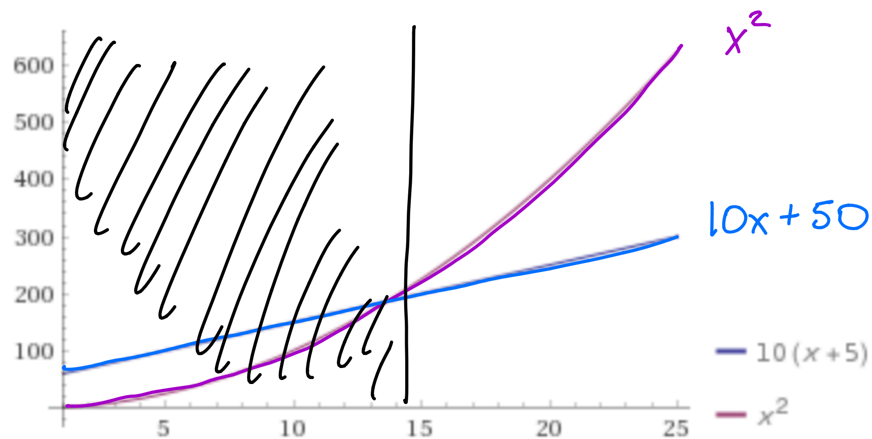
Asymptotic Order Of Growth

$n+1 = O(n)$

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$

$10x+50 = O(x^2)$

$n_0 = 15 \quad c = 1$



Asymptotic Order Of Growth

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$

- Roughly equivalent versions:

  - $\exists a, b \geq 0 \quad \forall n \in \mathbb{N} \quad f(n) \leq a \cdot g(n) + b$

  - $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$ $f(n) = 3n + 6$ $g(n) = n$

$\frac{3n+6}{n} = 3 + \frac{6}{n}$

Ask the Audience

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.

- Which of these statements are true?
  - $3n^2 + n = O(n^2)$
  - $n^3 = O(n^2)$
  - $10n^4 = O(n^5)$
  - $\log_2 n = O(\log_2(\sqrt{n}))$

Big-Oh Rules

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a < b \quad n^a = O\left(n^b\right)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\, a > 0, b > 1 \quad n^a = O(b^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

$n \cdot \log n = n \cdot O(n)$

$= O(n^2)$

A Word of Caution

- The notation $f(n) = O\big(g(n)\big)$ is weird—do not take it too literally

Asymptotic Order Of Growth

- **"Big-Omega" Notation:** $f(n) = \Omega\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \geq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \geq g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$

$f = O(g)$
$f = \Omega(g)$ $\Rightarrow$ $\Theta(g)$

- **"Big-Theta" Notation:** $f(n) = \Theta\big(g(n)\big)$ if there exists $c_1 \leq c_2 \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) = g(n)$
  - Roughly equivalent to $0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

Asymptotic Running Times

- **We usually write running time as a Big-Theta**
  - Exact time per operation doesn't appear
  - Constant factors do not appear
  - Lower order terms do not appear
- **Examples:**
  - $30\log_2 n + 45 = \Theta(\log n)$
  - $Cn\log_2 2n = \Theta(n\log n)$
  - $\sum_{i=1}^{n} Ci = \Theta(n^2)$
- **We usually "think asymptotically"**
  - We don't even discuss constants when they won't affect the asymptotic running time

Asymptotic Order Of Growth

- **"Little-Oh" Notation:** $f(n) = o\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) < g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
- **"Little-Omega" Notation:** $f(n) = \omega\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) > g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Ask the Audience

- **"Little-Oh" Notation:** $f(n) = o\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < c \cdot g(n)$ for every $n \geq n_0$.

- Which of these statements are true?
  - $3n^2 + n = o(n^2)$
  - $n^3 = o(n^2)$
  - $10n^4 = o(n^5)$
  - $\log_2 n = o(\log_2(\sqrt{n}))$

Ask the Audience!

- Rank the following functions in increasing order of growth (i.e. f_1, f_2, f_3, f_4 so that $f_i = O(f_{i+1})$)
  - $n \log_2 n$
  - n^2
  - $100n$
  - $3^{\log_2 n}$

Why Asymptotics Matter

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	4 sec
$n = 30$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	18 min	10^{25} years
$n = 50$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	11 min	36 years	very long
$n = 100$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	$<$ 1 sec	$<$ 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	$<$ 1 sec	$<$ 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	$<$ 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long