CS3000: Algorithms & Data
Jonathan Ullman

Lecture 3:
- Divide and Conquer: Mergesort
- Asymptotic Analysis

Sep 14, 2018

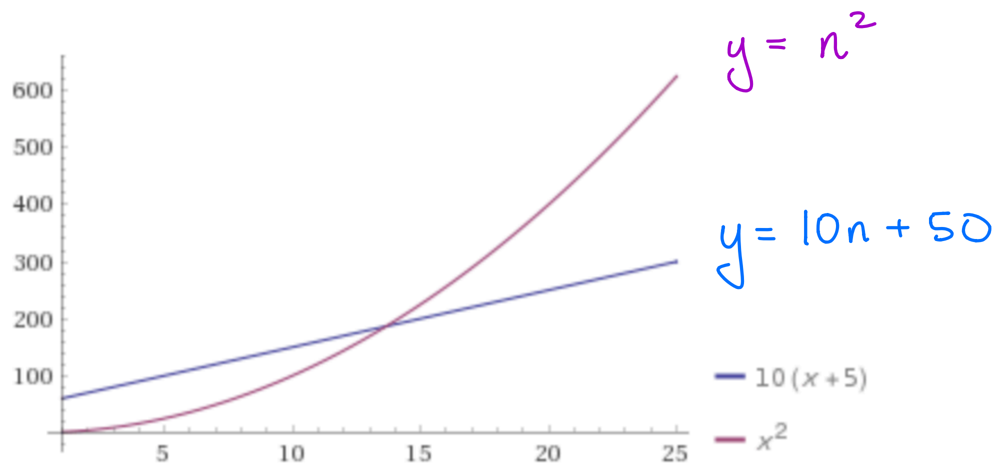
Asymptotic Analysis

Asymptotic Order Of Growth

- Predicting the wall-clock time of an algorithm is nigh impossible.
  - What machine will actually run the algorithm?
  - Impossible to exactly 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

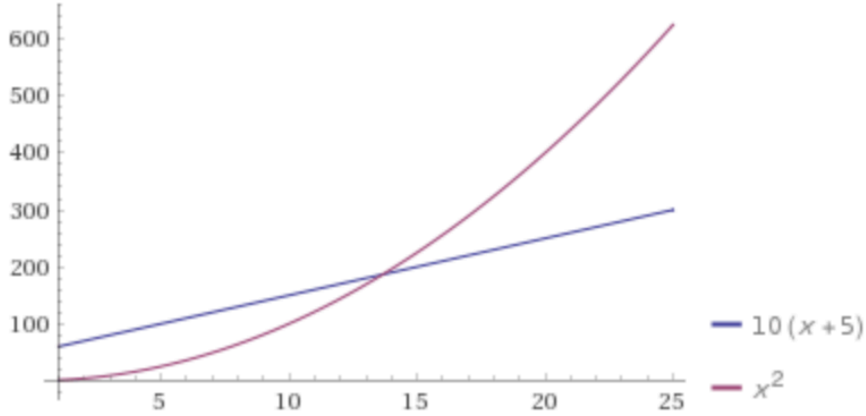


Asymptotic Order Of Growth

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

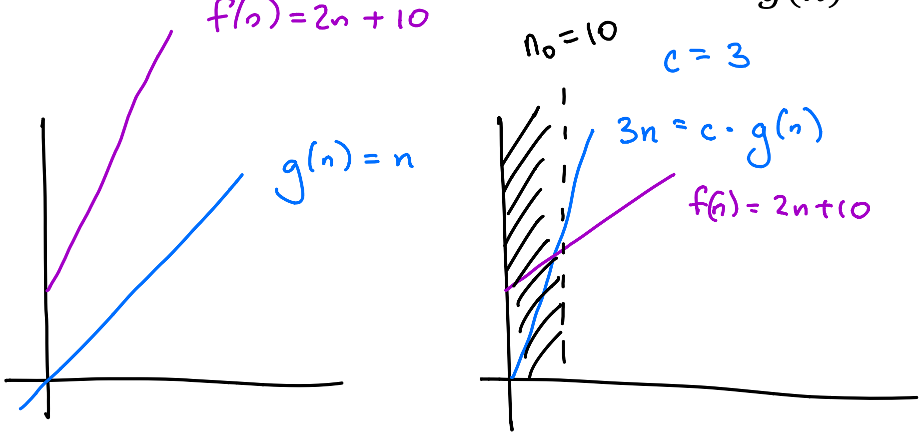
$T(n)$ = exact running time

$g(n)$ = nice function



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 to $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$



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?
  - ✓ $3n^2 + n = O(n^2)$
  - ✗ $n^3 = O(n^2)$ $\qquad \lim_{n\to\infty} \frac{n^3}{n^2} = \infty$
  - ✓ $10n^4 = O(n^5)$ $\qquad$ also $10n^4 = O(n^4)$
  - ✓ $\log_2 n = O(\log_{16} n)$

$$\log_{16} n = \frac{\log_2 n}{4}$$

Clm: $3n^2 + n = O(n^2)$

$c = 4$
$n_0 = 1$

$\forall n \geq 1 \quad f(n) = 3n^2 + n \leq 4n^2 = c \cdot g(n)$ $\blacksquare$

Big-Oh Rules

$\log_2 n = O(n^{\varepsilon})$

$\log_2 n = O(n^{.59})$

$n \log_2 n = O(n \times n^{.59})$

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$ $\qquad C \cdot g(n) = O(g(n))$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^{\varepsilon})$ $\qquad \log_2^{1000} n = O(n^{.001})$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\, a > 0, b > 1 \quad n^a = O(b^n)$ $\qquad n^{1000} = O(1.0001^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

$f_1 = O(g) \quad f_2 = O(g) \implies f_1 + f_2 = O(g)$

A Word of Caution

should be $f \in O(g)$

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

$n = O(n^2) \qquad n = O(n^3)$

$n^3 \neq O(n^2)$

$$n = \underbrace{1 + \dots + 1}_{n \text{ times}} = \underbrace{O(1) + O(1) + \dots + O(1)}_{n \text{ times}}$$

$$= \underbrace{O(1) + \dots + O(1)}_{n-1 \text{ times}}$$

$$\vdots$$

$$= O(1)$$

Asymptotic Order Of Growth

$\frac{1}{3}n^2 = \Omega(n^2)$

- **"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 $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$
- **"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 $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty)$

$f = O(g)$

$f = \Omega(g)$

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:**
  - $30 \log_2 n + 45 = \Theta(\log n)$
  - $Cn \log_2 2n = \Theta(n \log n)$
  - $\sum_{i=1}^{n} i = \Theta(n^2)$

$$n \log_2 2n = n \log_2 n + n = \Theta(n \log n)$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} = \Theta(n^2)$$

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 $n \geq n_0$.

  $n^2 = o(n^3)$

  - 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 $f(n) > g(n)$

    $n^3 = \omega(n^2)$
  - Roughly equivalent to $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Ask the Audience!

$\log_2 n = O(n)$

$n \log_2 n = O(n^2)$

$\log_2 n = O(n^{.59})$

- 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
  - $100n$
  - $3^{\log_2 n}$

$100n,\ 3^{\log_2 n},\ n\log_2 n,\ n^2$

$3^{\log_2 n},\ n^2,\ 100n,\ n\log_2 n$

$100n,\ n\log_2 n,\ n^2,\ 3^{\log_2 n}$

$100n,\ n\log_2 n,\ 3^{\log_2 n},\ n^2$

$3^{\log_2 n}$

$= 2^{(\log_2 3)(\log_2 n)}$

$= n^{\log_2 3} = n^{1.59}$

$100n,\ n\log_2 n,\ n^{\log_2 3} \approx n^{1.59},\ n^2$

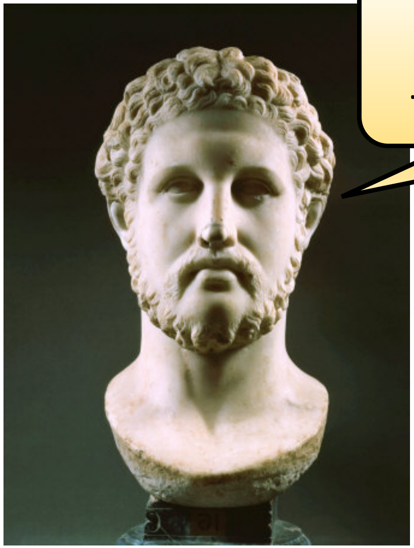
Why Asymptotics Matter

	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

- Exponential time bad / Polynomial time good
- Exponents matter

Divide and Conquer Algorithms

Divide and Conquer Algorithms



Divide et impera!
-Philip II of Macedon

- Split your problem into smaller subproblems
- Recursively solve each subproblem
- Combine the solutions to the subprobelms
- For many problems, combining is easier than solving

Divide and Conquer Algorithms

- **Examples:**
  - Mergesort: sorting a list
  - Binary Search: search in a sorted list
  - Karatsuba's Algorithm: integer multiplication
  - Fast Fourier Transform
  - …
- **Key Tools:**
  - Correctness: proof by induction
  - Running Time Analysis: recurrences
  - Asymptotic Analysis

Sorting

11	3	42	28	17	8	2	15

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

Any "comparable" items

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

2	3	8	11	15	17	28	42

A Simple Algorithm: Insertion Sort

Scan to find largest elt

11	3	42	28	17	8	2	15

11	3	15	28	17	8	2	42

repeat n-1 times

Running Time: $n + (n-1) + (n-2) + \ldots + 2$

$$= \left(\sum_{i=1}^{n} i\right) - 1 = \Theta(n^2)$$

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

11	3	15	2	17	8	28	42

Repeat $n - 1$ times.

2	3	8	11	15	17	28	42

A Simple Algorithm: Insertion Sort

Find the maximum

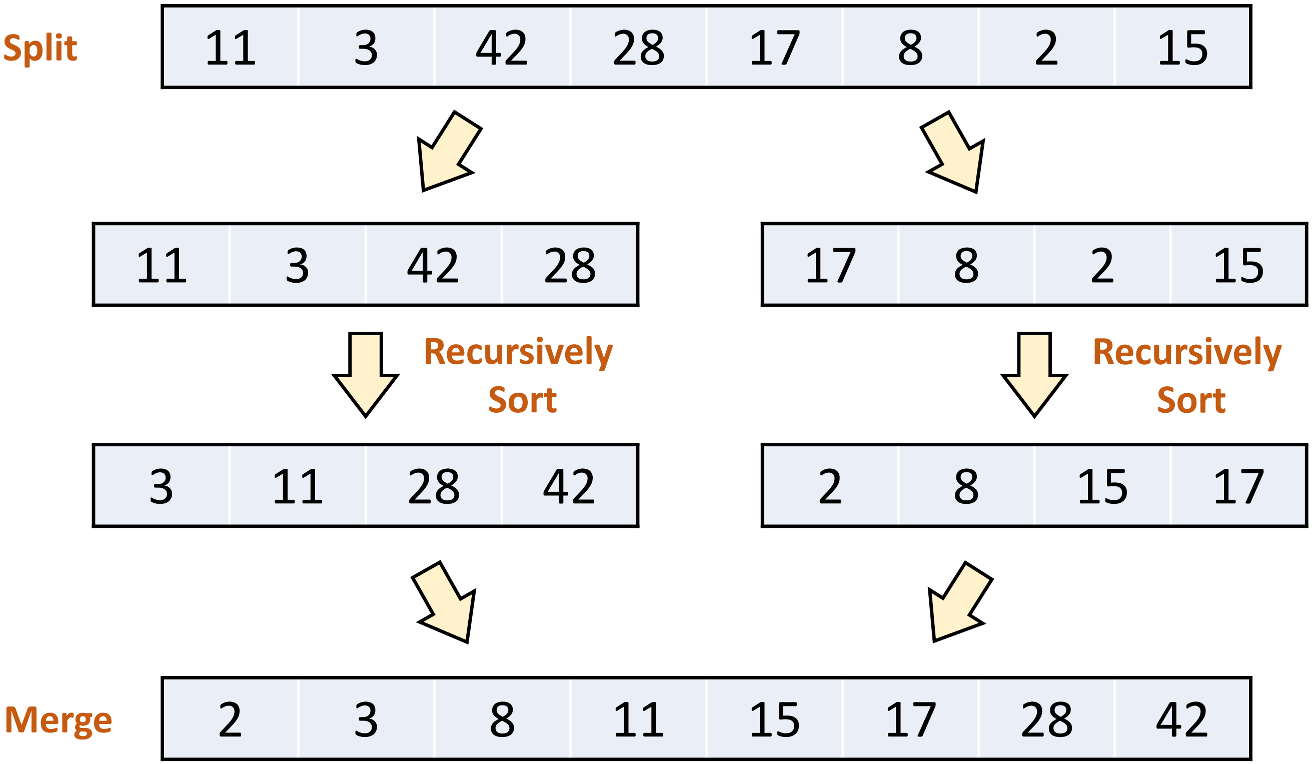
11	3	42	28	17	8	2	15

Swap it into place, repeat on the rest

11	3	15	28	17	8	2	42

Running Time:

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 $O(n)$
  - Merging two sorted lists is faster than sorting from scratch

$2n$ elements of A

$\times$ 2 ops per elem

$= O(n)$

3	11	28	42	$\boldsymbol{L}$

2	8	15	17	$\boldsymbol{R}$

2	3	8						$\boldsymbol{A}$

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):
  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

$\forall n \in \mathbb{N}$ $\forall$ list A of n numbers

MergeSort returns the list sorted.

Inductive Hypothesis:

$H(n)$: $\forall$ lists A of n numbers, MergeSort is correct.

Base Case: $H(1)$ is true, obviously

Inductive Step:

We will show that $H(1) \wedge H(2) \wedge \ldots \wedge H(n) \Rightarrow H(n+1)$

① Given any input A of size n+1, L and R have size $\lceil \frac{n+1}{2} \rceil \leq n$ and $\lfloor \frac{n+1}{2} \rfloor \leq n$

② By the IH, Mergesort sorts L, R correctly

$\boxed{\text{③ Since L, R are sorted, Merge(L, R) will be sorted}}$

④ Therefore MergeSort returns A in sorted order

Depends on the problem

Running Time of Mergesort

$T(n)$: running time on inputs of length n

$$T(n) = 2 \times T\left(\frac{n}{2}\right) + Cn$$

$$T(1) = C$$

$$T(n) = Cn\log_2 2n$$
$$= \Theta(n \log n)$$

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

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

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

  Return A
```

1

1

Cn

$2 \times T\left(\frac{n}{2}\right)$

Cn

1

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 (much) 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