(record)
HW1 out 5/14 - due 5/21

CS3000: Algorithms & Data
Drew van der Poel

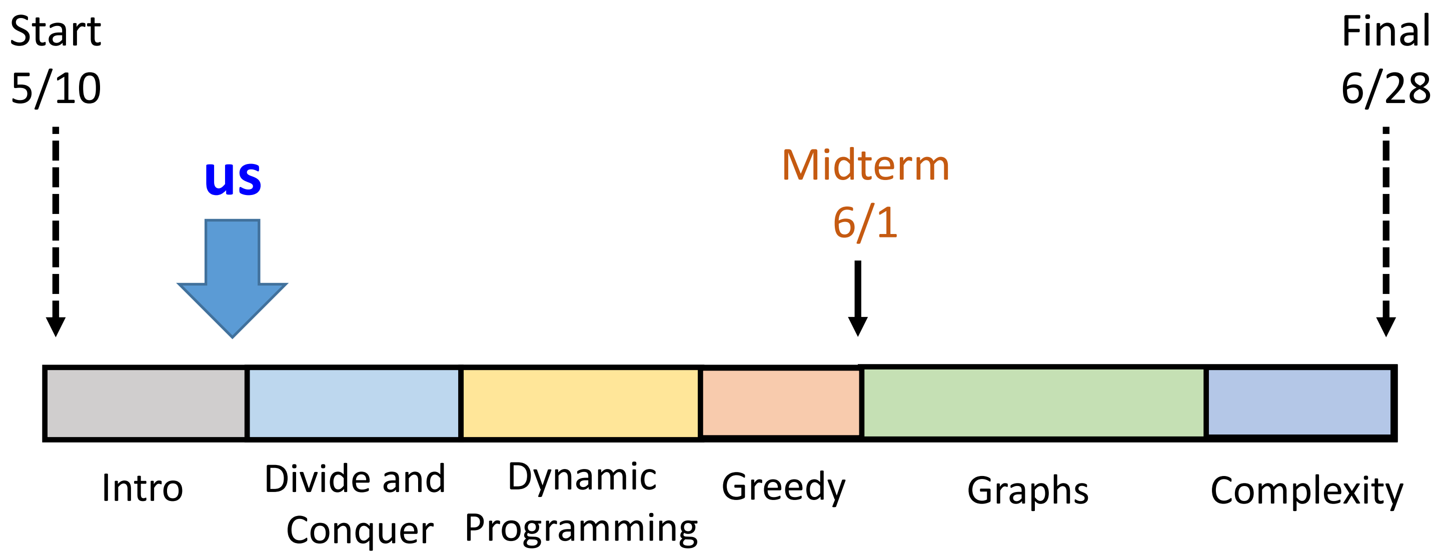
Quiz1 out 5/14 - due 5/17 (NOON)

Piazza

Lecture 4
- Finish Asymptotic Analysis
- Divide & Conquer: Merge Sort

May 13, 2021

Outline



Last class: stable matching, asymptotic analysis

Next class: divide and conquer: merge sort + Karatsuba's?

General Proof Example

- Prove that if $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then $f(n) + g(n) = O(h(n))$

$$f(n) \le c_1 h(n) \quad \forall n \ge n_0'$$

$$g(n) \le c_2 h(n) \quad \forall n \ge n_0''$$

$$f(n) + g(n) \le (c_1 + c_2) h(n) \quad \forall n \ge \max(n_0', n_0'')$$

$$c = c_1 + c_2 \qquad n_0 = \max(n_0', n_0'')$$

$$f(n) + g(n) \le c\, h(n) \quad \forall n \ge n_0$$

A Word of Caution

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

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

$$n = O(n)$$

$$n = O(10^n)$$

Asymptotic Analysis Rules

$70n^3 = O(n^3)$

order of growth

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$ $\qquad 2n = O(n)$
- **Lower order terms can be dropped**
  - E.g. $n^2 + \cancel{n^{3/2}} + \cancel{n} = O\left(n^2\right)$

ranking

- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$ $\qquad n^2 = O(n^{2.0001})$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$ $\qquad \log_2^{1000} n = O(n^{0.001})$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$ $\qquad n^{1000} = O(1.0001^n)$

Ask the Audience!

$x^{\log_b y} = y^{\log_b x}$ ‖ $(x^y)^2 = (x^2)^y$ ‖ $b^{\log_b a} = a$

- 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
  - $100n$
  - $3^{\log_2 n}$ → $\left(2^{\log_2 3}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 3} = n^{\log_2 3} = \boxed{n^{1.x}}$

slowest order of growth

1. $100n$
2. $n \log_2 n$
3. $3^{\log_2 n}$
4. n^2

fastest growth

$100n = O(n \log_2 n)$ $c = 100$

$100n \le 100 n \log_2 n$

$1 \le \log_2 n \quad \forall n \ge n_0 = 2$

$n \log n$ vs. $n^{1.x}$

$\log n$ vs. $n^{0.x}$

logs are Big-O of Polynomials

Asymptotic Order Of Growth

- **"Big-Omega" Notation:** $f(n) = \Omega(g(n))$ 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:** $f(n) = \Theta(g(n))$ 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 $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty)$
  - **Equivalent to:** $f(n) = O(g(n))$ AND $f(n) = \Omega(g(n))$

Asymptotic Running Times

- **It is *nice* to write running time as a Big-Theta**
  - **More precise than Oh or Omega**
  - Note: Sometimes it is simpler to just use Oh

needs to be as tight as possible

$f(n) = 3n^2 + n$

$f(n) = \Theta(n^2)$

$f(n) = O(n^2)$

$f(n) = O(n^3)$ not tight b/c $f(n) = O(n^2)$

More Examples

- $30\log_2 n + 45 = \Theta(\log_2 n)$

O: $c = 75$, $n_0 = 2$

$30\log_2 n + 45 \le 75\log_2 n \quad \forall n \ge 2$

$45 \le 45\log_2 n \quad \forall n \ge 2$ ✓

Ω: $c = 30$ $n_0 = 1$

$30\log_2 n + 45 \ge 30\log_2 n \quad \forall n \ge 1$

$45 \ge 0 \quad \forall n \ge 1$

$\boxed{\log_x ab = \log_x a + \log_x b}$

$4n\log_2 2n = 4n(\log_2 2 + \log_2 n)$

$= 4n(1 + \log_2 n) = 4n + 4n\log_2 n$

- $4n\log_2 2n = \Theta(n\log_2 n)$

O: $c = 8$ $n_0 = 2$

$4n + 4n\log_2 n \le 8n\log_2 n \quad \forall n \ge 2$

$4n \le 4n\log_2 n \quad \forall n \ge 2$ ✓

$1 \le \log_2 n$

Ω: $4n + 4n\log_2 n \ge 4n\log_2 n \quad \forall n \ge 1$

$c = 4$

$n_0 = 1$

$4n \ge 0 \quad \forall n \ge 1$ ✓

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

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

O: $c = 1$ $n_0 = 1$

$\frac{n^2}{2} + \frac{n}{2} \le 1n^2 \quad \forall n \ge 1$

$\frac{n}{2} \le \frac{n^2}{2}$

$1 \le n$ ✓

Ω: $c = \frac{1}{2}$ $n_0 = 1$

$\frac{n^2}{2} + \frac{n}{2} \ge \frac{1}{2}n^2 \quad \forall n \ge 1$

$\frac{n}{2} \ge 0 \quad \forall n \ge 1$

- Problems: counting students, stable matching

- Alg. techniques:

- Analysis: **asymptotic analysis**

- Proof techniques: (strong) induction, contradiction

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

Motivation: Differences across orders of growth

of operations

	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

Instance size

Running time when 1 op. takes ~1 microsecond

Observations:

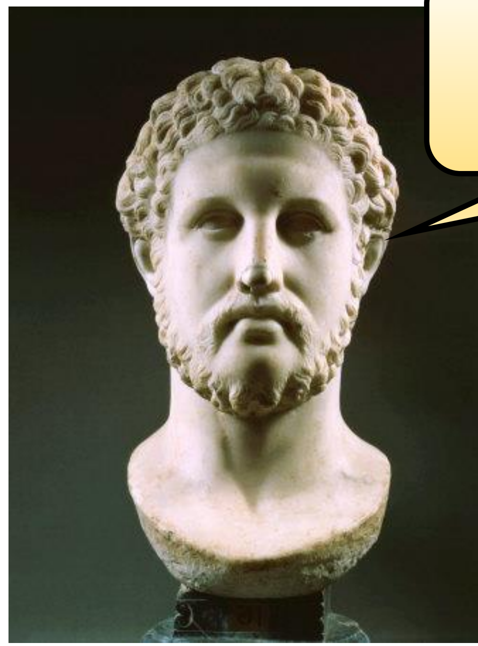
1. Different polynomials make a BIG difference! (e.g. n^2 vs. n^3 when n $\geq$ 1000)
2. Things go bad quickly (look down any polynomial or exponential column)! We want **scalability**!
3. Large instances are when there is a difference (log still good!)

- split (divide) a big problem into smaller instances
- solve the small problems
- combine (conquer) the smaller solns. to solve the original large instance!

Divide and Conquer Algorithms

What do you think of?

Divide and Conquer Algorithms



Divide et impera!
-Philip II of Macedon

D+C Recipe

same as the original problem

- Split your problem into smaller subproblems
- Recursively solve each subproblem

Keep splitting

- Combine the solutions to the subprobelms

- Problems: counting students, stable matching

- Alg. techniques: **divide & conquer**

- Analysis: asymptotic analysis

- Proof techniques: (strong) induction

Divide and Conquer Algorithms

- **Examples:**
  - Today Mergesort: sorting a list
  - Binary Search: search in a sorted list
  - Karatsuba's Algorithm: integer multiplication
  - Finding the k-th largest element in an array
  - …

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

Problem: Sorting a List

Input: A list of comparable elements (numbers, words, etc.)

Problem: Sort the list so that it is ordered

Output: The list of elements in ascending (or descending) order

- Problems: counting students, stable matching, **sorting**

- Alg. techniques: divide & conquer

- Analysis: asymptotic analysis

- Proof techniques: (strong) induction, contradiction

Sorting

Input

$n=8$

11	3	42	28	17	8	2	15

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

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

output:

2	3	8	11	15	17	28	42

A Simple Algorithm: Selection Sort

11	3	42	28	17	8	2	15

A[1] A[n]

~~11~~ ~~3~~ 2

3

2 | 3 42 28 17 8 11 15

~~42~~ ~~28~~ ~~17~~ 8

2 3 | 42 28 17 8 11 15

2 3 8 | 28 17 42 11 15

A Simple Algorithm: Selection Sort

Find the minimum

11	3	42	28	17	8	$\boxed{2}$	15

Swap it with end, repeat on first n-1

2	3	42	28	17	8	11	15

Repeat
$n - 1$ more times.

2	3	8	11	15	17	28	42

Counting Operations

Examples of what counts as an operation:

- Creating (and assigning a value to) a variable
- Assigning a value to an existing variable
- Looking up a value
- Arithmetic (+, -, *, /, %)
- Comparisons (<, >, =, ≤, ≥)
- Function calls

A Simple Algorithm: Selection Sort

Find the min. & swap with end

11	3	42	28	17	8	2	15

Repeat n-1 times

2	3	8	11	15	17	28	42

(asymptotic time complexity)

Running Time (# of operations):

Steps:

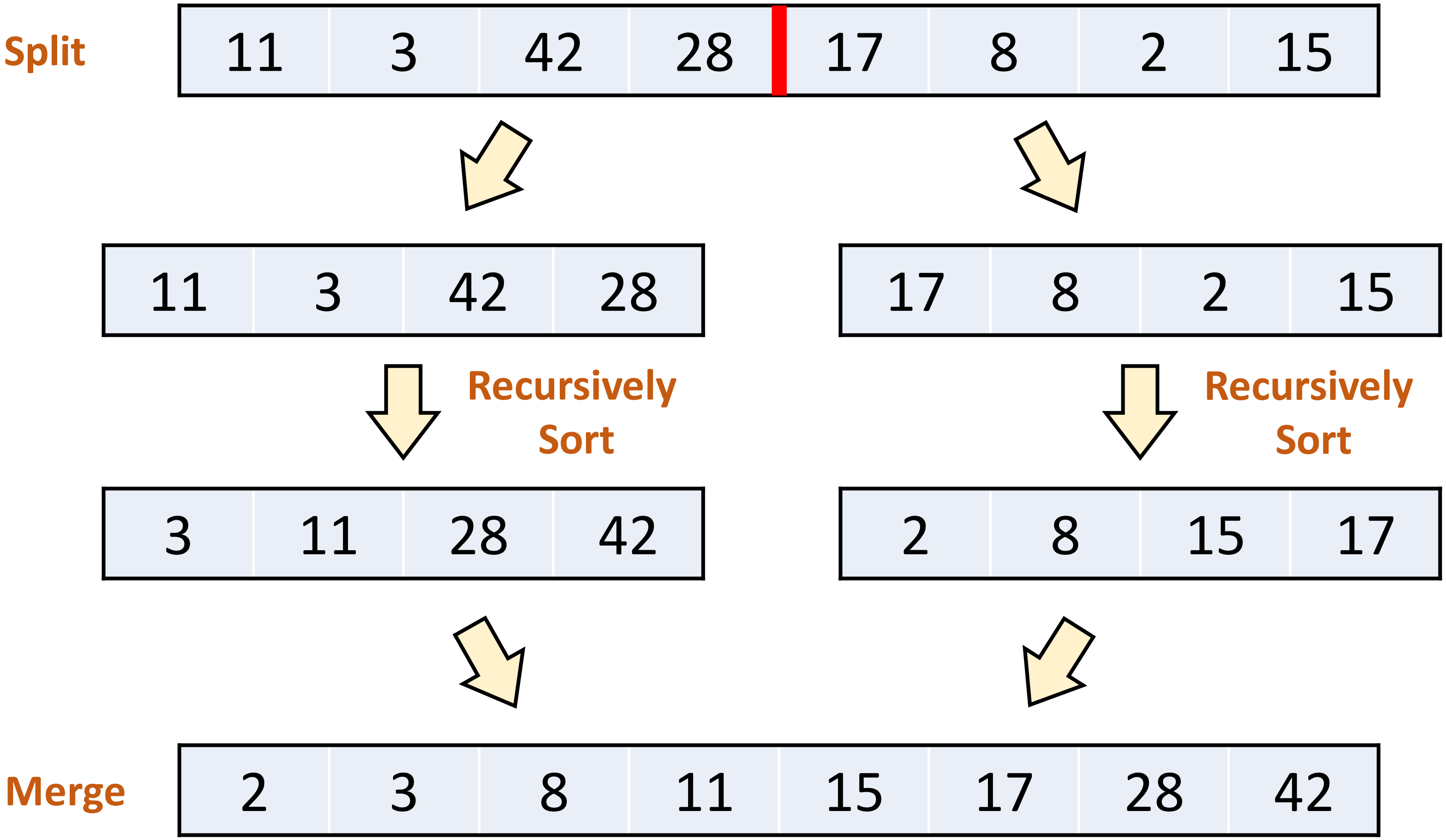
	# of ops.
scan	n
swap	1

total # of ops. n iterations

$$\underline{n} + 1 + \underline{n-1} + 1 + \underline{n-2} + 1 + \ldots \underline{1} + 1$$

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

Divide and Conquer: Mergesort



Divide and Conquer: Mergesort

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

3	11	28	42	$\boldsymbol{L}$

2	8	15	17	$\boldsymbol{R}$

								$\boldsymbol{A}$

Merging Pseudocode

```
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
```

MergeSort

```
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
```

Mergesort Demo

11	3	42	28

Correctness of Mergesort

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

- H(n):

- Base Case:

Correctness of Mergesort

- Inductive step:

```
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
```

Running Time of Mergesort

T(1) =

T(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
```

Recursion Trees

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

Running Time of Mergesort

Total work: $\sum_{i=0}^{last\ level} work\ at\ level\ i$

- Problems: counting students, stable matching, sorting

- Alg. techniques: divide & conquer

- Analysis: asymptotic analysis, **recursion trees**

- Proof techniques: (strong) induction, contradiction

Recursion Trees

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

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

Mergesort Summary

- Sort a list of n numbers in $\Theta(n \log n)$ 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 & recursion trees