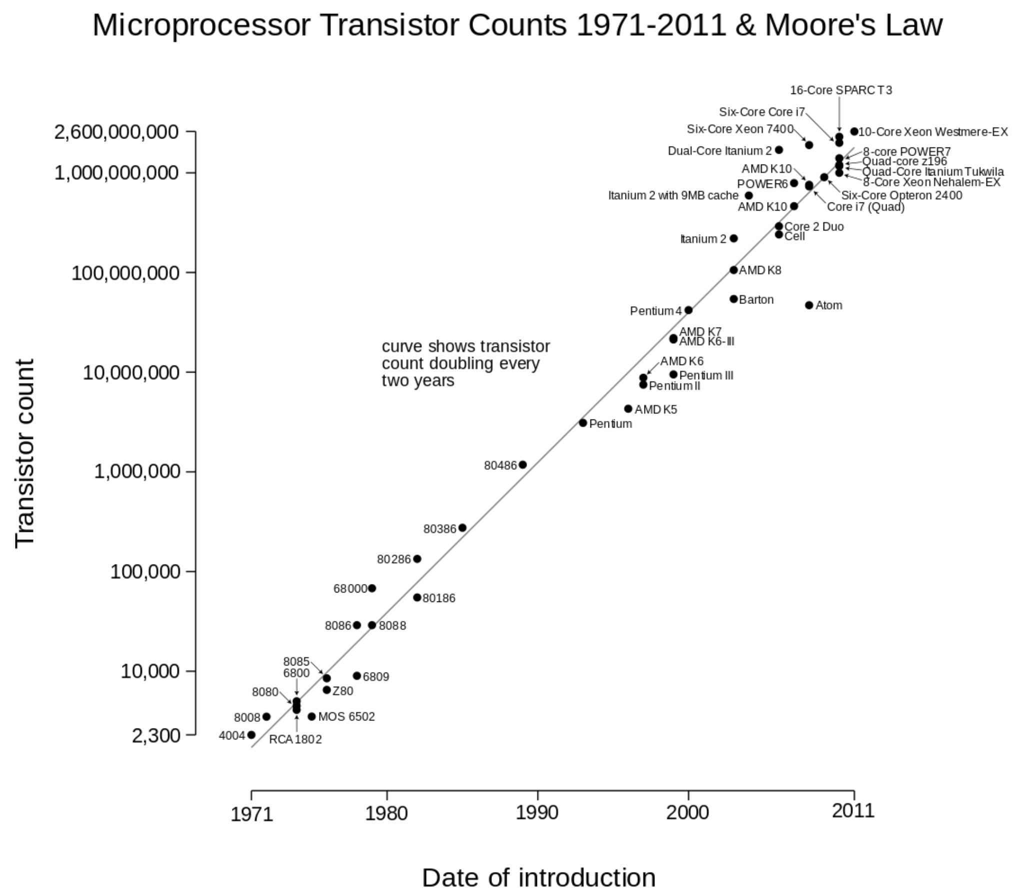
CS 4800: Algorithms & Data

Lecture 1

January 9, 2018

Huy L. Nguyen

- Email: hu.nguyen@northeastern.edu
- Office hours: Wednesday 2:00 – 4:00, WVH 358
- Research:
  - Algorithms for massive data sets (“big data”)
  - Theoretical aspects of machine learning


Microprocessor Transistor Counts 1971-2011 & Moore's Law
Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
Date of introduction
1971
1980
1990
2000
2011
curve shows transistor count doubling every two years
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
AMD K10
Quad-Core Itanium Tukwila
POWER6
8-Core Xeon Nehalem-EX
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Cell
Itanium 2
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K6-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
8080
Z80
8008
MOS 6502
4004
RCA 1802

Bits

Software Progress Beats Moore's Law

By STEVE LOHR MARCH 7, 2011 3:56 PM 21

… a study of progress over a 15-year span on a benchmark production-planning task. Over that time, the speed of completing the calculations improved by a factor of 43 million. Of the total, a factor of roughly 1,000 was attributable to faster processor speeds, according to the research by Martin Grotschel, a German scientist and mathematician. Yet a factor of 43,000 was due to improvements in the efficiency of software algorithms.

CS4800 syllabus

- [x] Algorithm analysis
- [x] Algorithm design

Course structure

- http://www.ccs.neu.edu/home/hlnguyen/cs4800/spring18
- Lectures: Tuesdays and Fridays 1:35pm – 3:15pm
- Homework: problem sets posted every week (50%)
  - Math proofs
  - Programming problems
- Tests: 2 midterms (15% each)
- Final exam (20%)

Recipe for success

HLN	lecture
staff	office hour
you	reading
you	homework
you	programming assignment
you	midterms
you	final exam

☞ Partnership!

Discussion forum

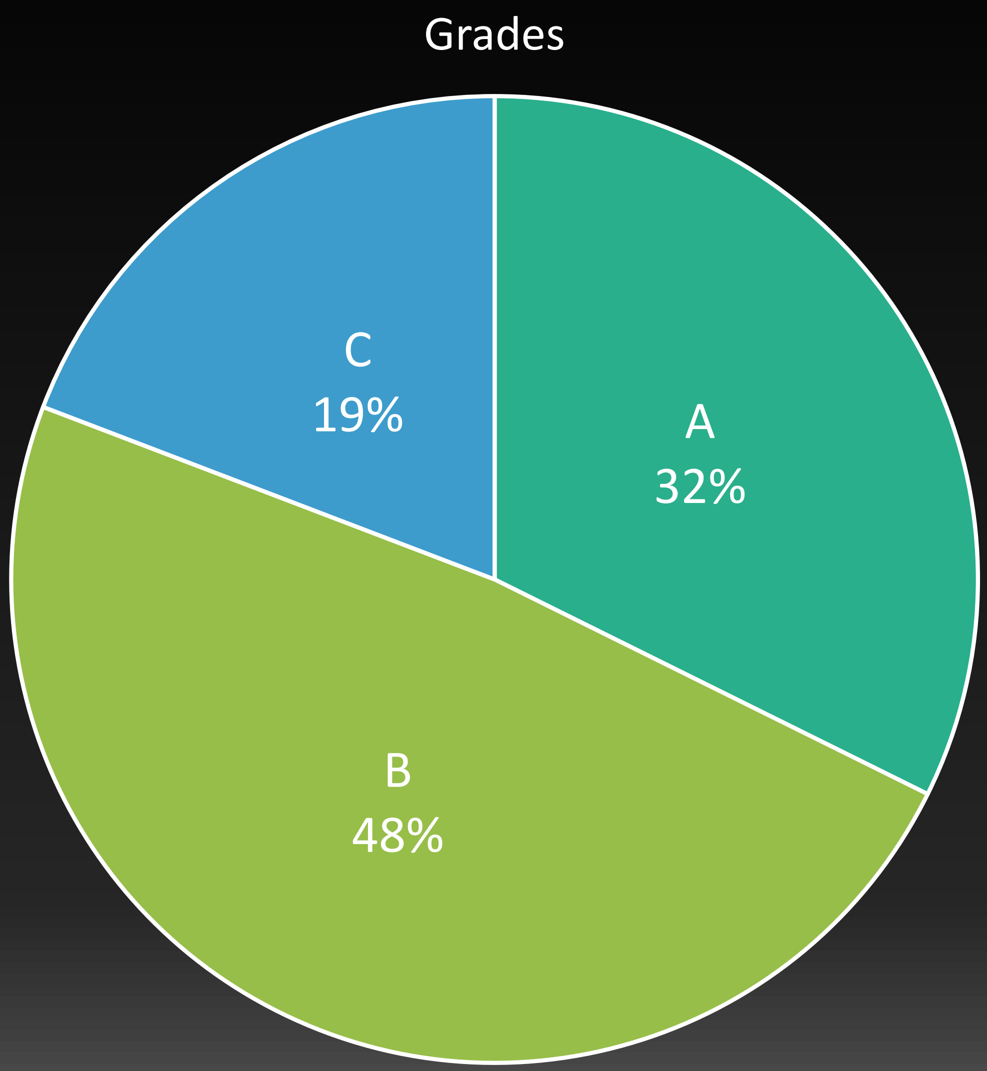
- https://piazza.com/northeastern/spring2018/cs4800
- Ask questions
- Help your peers

Homework submission

- Register at https://gradescope.com/courses/13862
- Use entry code: **9ERX47**

Topics

- Divide and conquer
- Dynamic programming
- Greedy algorithms
- Greedy in graphs
  - Minimum spanning trees
  - Shortest paths
- Shortest paths via dynamic programming
- Maximum flows, matching
- Hashing


Grades
C
19%
A
32%
B
48%

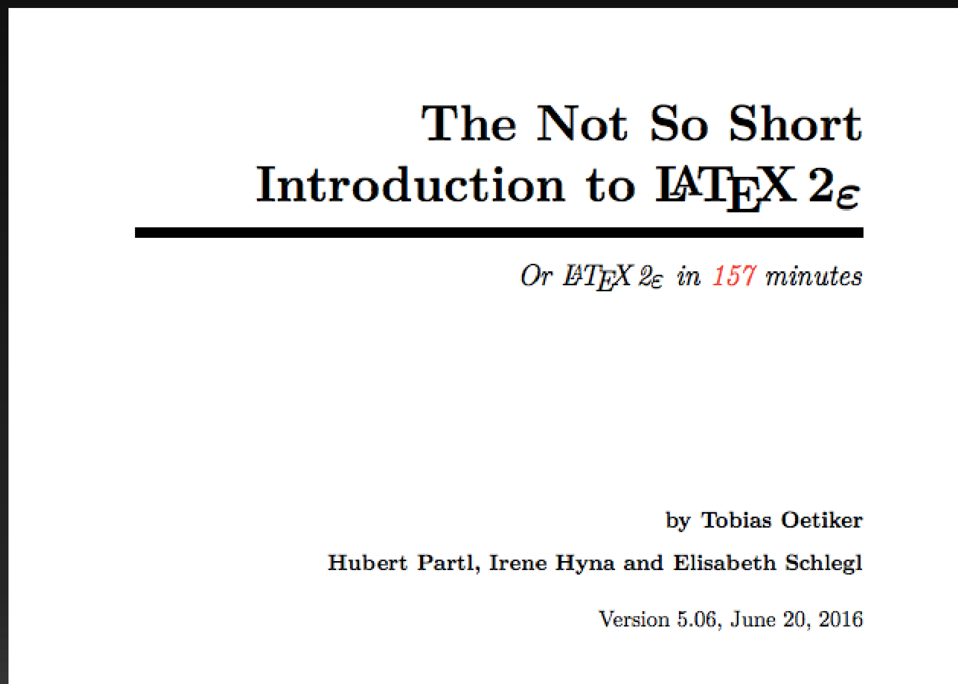
CS4800: Algorithms and Data

[Home] [Schedule]

The schedule is tentative and subjects to change (e.g. snow days)

Date	Topic	Reading	Problem sets
Jan 9	Introduction, induction	Lecture slides DPV Chapter 0 Erickson Appendix I	PS0 out PS0 source
Jan 12	asymptotic order of growth, Karatsuba	Lecture slides Karatsuba: Erickson 1.8, demo	PS0 due PS1 out PS1 source
Jan 16	recursion tree, mergesort	Lecture slides Erickson Appendix II.1-3 Mergesort: demo	
Jan 19	Master theorem, change of variable	Lecture slides Master theorem: Erickson Appendix II.3	PS1 due PS2 out PS2 source
Jan 23	deterministic median	Lecture slides Erickson 1.7	
		Lecture slides	PS2 due

LaTeX



The Not So Short
Introduction to $\LaTeX\ 2_\varepsilon$

Or $\LaTeX\ 2_\varepsilon$ in 157 minutes

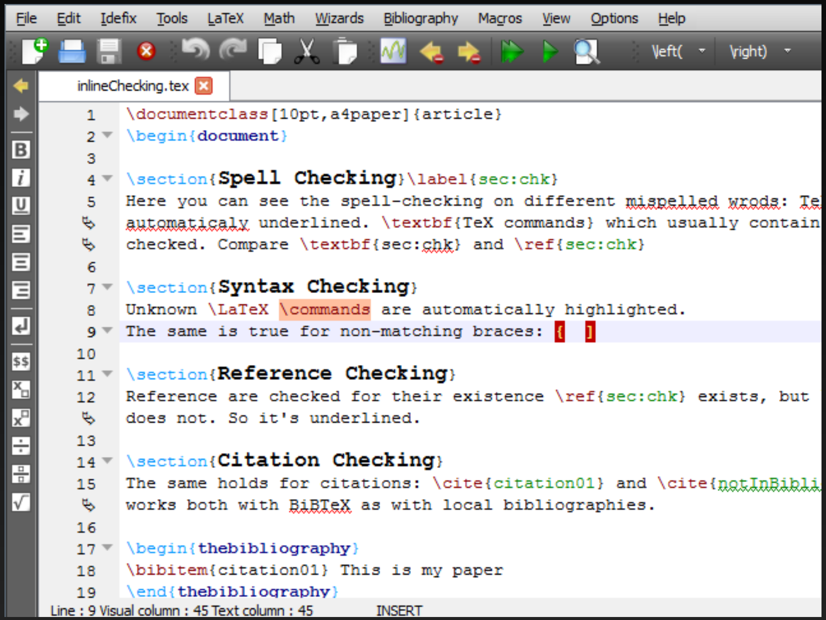
by Tobias Oetiker

Hubert Partl, Irene Hyna and Elisabeth Schlegl

Version 5.06, June 20, 2016

LaTeX

- Many editors: TeXShop, Texmaker, TeXstudio, …
- Homework template on course website



TeXstudio

Overleaf.com

Homework policies

- Discuss with peers (strongly encouraged!)
- Write up in your own words, acknowledge people you worked with
- Write your own codes
- Do not submit anything you cannot explain to me

Algorithms

- al-Khwārizmī (c. 780 – c. 850)

- The Compendious Book on Calculation by Completion and Balancing

Algorithms

- ~~Procedures~~ for solving linear and quadratic equations

- Introduce decimal numbers to Western world

Fibonacci (c. 1170 – c. 1250)

- Popularize decimal positional number system
- Fibonacci numbers

$$F_n = \begin{cases} F_{n-1} + F_{n-2} \; if \; n > 1 \\ 1 \; if \; n = 1 \\ 0 \; if \; n = 0 \end{cases}$$

- F_n grows very quickly, $F_n \approx 2^{0.694n}$

An algorithm for computing Fibonacci numbers

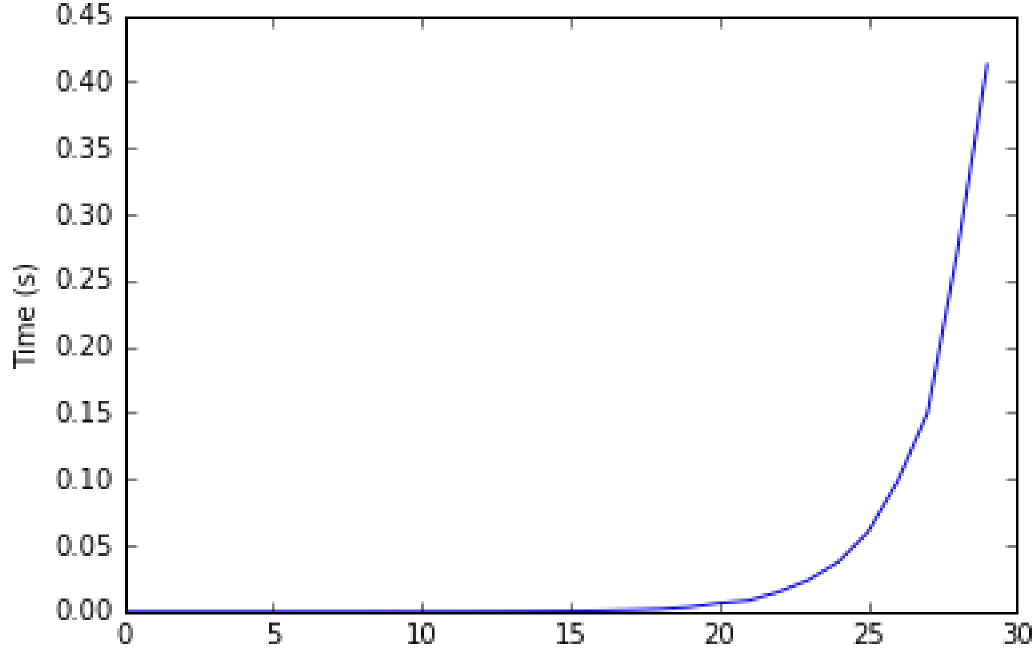
Pseudocode

```
function fib(n):
   if n = 0 then return 0
   else if n = 1 then return 1
   else return fib(n-1) + fib(n-2)
```

Python

```
def fib(n):
   if n == 0: return 0
   elif n == 1: return 1
   else: return fib(n-1) + fib(n-2)
```

How fast?



Running time analysis

function fib(n):
  if n = 0 then return 0
  else if n = 1 then return 1
  else return fib(n-1) + fib(n-2)

Function call   addition   Function call

$$T(n) = T(n-1) + T(n-2) + 3$$

$T(n)$ is larger than F_n

Induction

- Guess: computing F_n takes more than $2^{n/2}$ operations
- Verify: computing F_0, F_1 needs > $2^{1/2}$ operations
- Cannot verify all n=0,1,2,3,4,…
- How to prove for for all n?
- Induction: assume that the claim is true for all n<k, will prove it is true for n=k
- True for n=1 ⇨ True for n=2 ⇨ True for n=3 …
- True for all n!

An induction proof

- Claim: for all integer n, computing F_n needs at least $2^{n/2}$ operations
- Base case: computing F_0, F_1 needs at least $2^{1/2}$ operations
- Inductive step: assuming claim is true for all $n<k$
- To compute F_k
  - Make 2 recursive calls to compute F_{k-1} and F_{k-2}
  - By assumption, these calls require $2^{(k-1)/2}$ and $2^{(k-2)/2}$ operations, respectively
  - Thus, we need at least $2^{(k-1)/2} + 2^{(k-2)/2} > 2^{k/2}$ operations

```
function exponential(a, n):
        if n = 0 then return 1
        else if n = 1 then return a
        else return exponential(a, ⌈n/2⌉)*exponential(a, ⌊n/2⌋)
```

- What does this function compute?
- Prove that exponential(a,n) needs n-1 multiplications for n>=1

YOUR PROOF: fill in ???

- Claim: for all integer n, ???
- Base case: computing exponential(a,0) and exponential(a,1) needs ???
- Inductive step: assuming claim is true for all n<k, will show the claim is true for n=k
- ???

SAMPLE PROOF:

- Claim: for all n, computing F_n needs $2^{n/2}$ operations
- Base case: computing F_0, F_1 needs $2^{1/2}$ operations
- Inductive step: assume claim is true for all n<k, will prove it for n=k
- To compute F_k, we make 2 recursive calls to compute F_{k-1} and F_{k-2}
- By assumption, these calls require $2^{(k-1)/2}$ and $2^{(k-2)/2}$ operations, respectively
- We need $2^{(k-1)/2} + 2^{(k-2)/2} > 2^{k/2}$ operations

function exponential(a, n):

if n = 0 then return 1

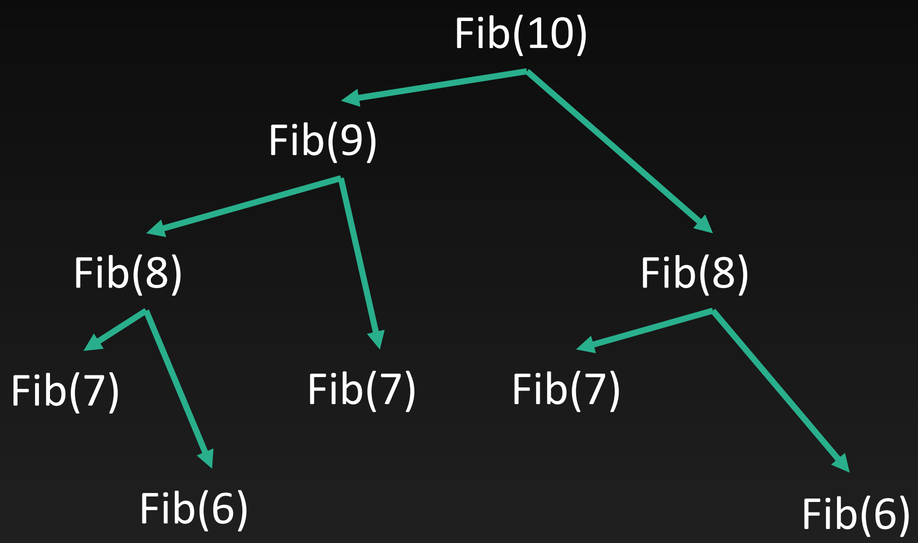
else if n = 1 then return a

else return exponential(a, $\lceil n/2 \rceil$)*exponential(a, $\lfloor n/2 \rfloor$)

- What does this function compute?
- Prove that exponential(a,n) needs n-1 multiplications for n>=1

- Claim: for all integer n>0, exponential(a,n) needs n-1 multiplications
- Base case: computing exponential(a,1) needs 0 multiplication
- Inductive step: assuming claim is true for all n<k, will show the claim is true for n=k
- exponential(a,k) makes 2 recursive calls to exponential(a, $\lceil k/2 \rceil$) and exponential(a, $\lfloor k/2 \rfloor$)
- By assumption, they require $\lceil k/2 \rceil - 1$ and $\lfloor k/2 \rfloor - 1$ multiplications
- On top of these 2 calls, we perform 1 more multiplication
- Thus, in total, we need $\left\lceil \frac{k}{2} \right\rceil - 1 + \left\lfloor \frac{k}{2} \right\rfloor - 1 + 1 = k - 1$ multiplications

Why so slow?



$$F_n = \begin{cases} F_{n-1} + F_{n-2} \; if \; n > 1 \\ 1 \; if \; n = 1 \\ 0 \; if \; n = 0 \end{cases}$$

Store intermediate results

- Create array fib[0..n]
- $fib[0] \leftarrow 0$
- $fib[1] \leftarrow 1$
- For i from 2 to n:
  - $fib[i] \leftarrow fib[i-1] + fib[i-2]$

Python:

```
fib = [0] * (n+1)
fib[0] = 0
fib[1] = 1
for i in range(2,n+1):
   fib[i] = fib[i-1] + fib[i-2]
```

- New total time: 0.000385 second!
- Speedup by 1000 fold!

Running time analysis

- Single for loop with one addition inside the loop
- Total time: n
- Inaccuracy: F_n grows quickly, each addition is not a single operation

Goal

- Formal framework for analyzing running times
- Accurate enough to describe general behaviors of algorithms
- Imprecise enough to avoid intricacy in processor types, programming languages