Prof. Wheeler Ruml
TA William Doyle

http://www.cs.unh.edu/~ruml/cs758

4 handouts:
course info, schedule, slides, asst 1

2 online handouts:
programming tips, formulas
Algorithms
web: search, caching, crypto
networking: load balancing, synchronization, failover
machine learning: data mining, recommendation, prediction
bioinformatics: alignment, matching, clustering
hardware: design, simulation, verification
business: allocation, planning, scheduling
AI: robotics, games
Algorithm

- precisely defined
- mechanical steps
- terminates
- input and related output

What might we want to know about it?
**Why?**

- Computer scientist ≠ programmer
  - understand program behavior
  - have confidence in results, performance
  - know when optimality is abandoned
  - solve ‘impossible’ problems
  - sets you apart (eg, Amazon.com)

- CPUs aren't getting faster
- Devices are getting smaller
- Software is the differentiator
- ‘Software is eating the world’ — Marc Andreessen, 2011
- Everything is computation
780-850 AD
Born in Uzbekistan, worked in Baghdad.
Solution of linear and quadratic equations.
Founder of algebra.
Popularized arabic numerals, decimal positional numbers
→ algorism (manipulating digits)
→ algorithm.

The Compendious Book on Calculation by Completion and Balancing, 830.
invented algorithm analysis

*The Art of Computer Programming*, vol. 1, 1968

developed TeX,
literate programming

many famous students
published in MAD magazine
This Class
requires 531/659 (formal thinking), 515 (data structures, C)
some intentional overlap!
central, thus now required
same content, assignments, grading both semesters
continuous improvement!
‘Greatest Hits’

1. data structures: trees, tries, hashing
2. algorithms: divide-and-conquer, dynamic programming, greedy, graphs
3. correctness: invariants
4. complexity: time and space
5. NP-completeness: reductions

Not including

1. (much) computability
2. (many) randomized algorithms
3. parallel algorithms
4. distributed algorithms
5. numerical algorithms, eg: crypto, linear algebra
6. geometric algorithms
7. on-line or ‘run forever’ algorithms
8. fancy analysis
Course Mechanics

- names → faces
- sign up sheet
- General information
  - contact, books, C, due dates, collaboration, piazza.com
- Schedule
  - wildcard slot
- Expectations
  - $50/4 = 12.5; 50/3 = 16.7$
  - last year median 12, mean 12.8, stddev 5.5
- Feedback is always needed and appreciated.
  - eg, EOLQs. Try coming to my office hours!
Complexity

- Sorting
- Counting Sort
- Correctness
- Counting Sort
- Complexity
- Counting Sort
- Order Notation
- $O()$
- Examples
- And Friends
- Asymptotics
- EOLQs
How to sort one million records?
Sorting

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort
- Merge Sort
- Heap Sort
- Quick Sort

How to sort one million records?

How to sort one billion 16-bit integers?
Sorting

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort
- Merge Sort
- Heap Sort
- Quick Sort

How to sort one million records?

How to sort one billion 16-bit integers?

How to sort one trillion 4-bit integers?
For \( n \) numbers in the range 0 to \( k \):

1. for \( x \) from 0 to \( k \)
2. \( \text{count}[x] \leftarrow 0 \)
3. for each input number \( x \)
4. increment \( \text{count}[x] \)
5. for \( x \) from 0 to \( k \)
6. do \( \text{count}[x] \) times
7. emit \( x \)
For $n$ numbers in the range 0 to $k$:

1. for $x$ from 0 to $k$
2. $\text{count}[x] \leftarrow 0$
3. for each input number $x$
4. increment $\text{count}[x]$
5. for $x$ from 0 to $k$
6. do $\text{count}[x]$ times
7. emit $x$

Correctness?

Complexity?
Correctness

property 1: output is in sorted order
proof sketch: output loop increments $x$, never decrements
Correctness

property 1: output is in sorted order
proof sketch: output loop increments \( x \), never decrements

property 2: output contains same numbers as input
invariant:
Correctness

property 1: output is in sorted order
proof sketch: output loop increments $x$, never decrements

property 2: output contains same numbers as input
invariant: for each value,
\[ \text{remaining input} + \text{sum of counts} = \text{total} \]
proof sketch:
Correctness

property 1: output is in sorted order
proof sketch: output loop increments \( x \), never decrements

property 2: output contains same numbers as input
invariant: for each value,
\[
\text{remaining input} + \text{sum of counts} = \text{total}
\]
proof sketch:
initialized/established: before line 3
maintained: through lines 3–4
at termination: no remaining input
    each number printed count times
therefore, output has same numbers as input
Counting Sort

For \( n \) numbers in the range 0 to \( k \):

1. for \( x \) from 0 to \( k \)
2. \( \text{count}[x] \leftarrow 0 \)
3. for each input number \( x \)
4. increment \( \text{count}[x] \)
5. for \( x \) from 0 to \( k \)
6. do \( \text{count}[x] \) times
7. emit \( x \)

Correctness? Yes.

Complexity?
Complexity

RAM model: no cache
order of growth
worst-case

[ try with previous slide ]
Counting Sort

For $n$ numbers in the range 0 to $k$:

1. for $x$ from 0 to $k$ $\quad O(k)$
2. $\quad \text{count}[x] \leftarrow 0$
3. for each input number $x$ $\quad O(n)$
4. $\quad \text{increment count}[x]$
5. for $x$ from 0 to $k$ $\quad O(k + n)$
6. $\quad \text{do count}[x]$ times
7. $\quad \text{emit } x$

$O(k + n + k + n) = O(2n + 2k) = O(n + k) \neq O(n \lg n)$
ignore constant factors
ignore ‘start-up costs’
upper bound
ignore constant factors
ignore ‘start-up costs’
upper bound

\[ f(n) = O(g(n)) \]

eg, running time is \( O(n \log n) \)
\[
O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \text{ such that } f(n) \leq c g(n) \text{ for all } n \geq n_o \}\]

We can upper-bound (the tail of) \( f \) by scaling up \( g \).

Note non-transitive use of \( = \). Pronounced ‘is’.

Eg:

1. \( 0.002x^2 - 35, 456x + 2^{80} \)
2. \( O(n^2) \text{ vs } O(n^3) \)
3. \( O(2^n) \text{ vs } O(3^n) \)
4. \( O(2^n) \text{ vs } O(2^{n+2}) \text{ vs } O(2^{2n}) \text{ vs } O(n^n) \)

“What is \( n \)?:”
Examples

\[ n^3 = O(n^2) \]
\[ 0.2x^2 - 456x + 2^{20} \]
\[ 10n^2 + 5n \]
\[ O(n^2) \text{ vs } O(n^3) \]
Upper bound (‘order of’):
\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_o \} \]

Lower bound:
\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \text{ such that } cg(n) \leq f(n) \text{ for all } n \geq n_o \} \]

Tight bound:
\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, n_0 \text{ such that } c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_o \} \]
Upper bound ('dominated by'):
\[ o(g(n)) = \{ f(n) : \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \} \]

Lower bound ('dominates'):
\[ \omega(g(n)) = \{ f(n) : \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \} \]
What’s still confusing?
What question didn’t you get to ask today?
What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.

Thanks!