Prof. Wheeler Ruml
TA Bence Cserna

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

6 handouts:
course info, schedule, programming tips, formulas, slides, asst 1
- web: search, caching, crypto
- networking: load balancing, synchronization, failover
- machine learning: data mining, recommendation
- 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 (e.g., Amazon.com)

- CPUs aren’t getting faster
- Devices are getting smaller
- Software is the differentiator
- 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
<table>
<thead>
<tr>
<th>Algorithms</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>This Class</strong></td>
<td>■ Relations  ■ Topics  ■ Course Mechanics</td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
</tr>
</tbody>
</table>

**This Class**
requires 531/659 (formal thinking), 515 (data structures, C)
good mid-curriculum foundation for later advanced courses
UNH: one of 3 ‘theory’ courses (compilers, formal spec)
many schools: required
continuous improvement!
‘Greatest Hits’

1. data structures: trees, heaps, hashing
2. algorithms: 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 programming
6. geometric algorithms
7. on-line or ‘run forever’ algorithms
8. fancy analysis
names → faces

sign up sheet

General information

- contact, books, C, due dates, collaboration, piazza.com

Schedule

- first week squeezed
- wildcard slot

Feedback is greatly needed and appreciated.

- eg, EOLQs. Try coming to my office hours!
Complexity

Algorithms

This Class

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

How to sort one billion 16-bit integers?
How to sort one million records?

How to sort one billion 16-bit integers?

How to sort one trillion 4-bit integers?
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 \( x \)
Counting Sort

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

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

Correctness? Yes.

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

\[
O(k + n + k + n) = O(2n + 2k) = O(n + k) \neq O(n \log n)
\]
### Order Notation

ignore constant factors
ignore ‘start-up costs’
upper bound
Ignore constant factors, ignore 'start-up costs', upper bound

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

e.g., 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 cg(n) \text{ for all } n \geq n_0 \}\]

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 \( n \)?”
0.002x^2 - 35, 456x + 2^{80}
\(O(n^2)\) vs \(O(n^3)\)
And Friends

Upper bound (‘order of’):
\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \]

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

Tight bound:
\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, n_0 \text{ such that } c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0 \} \]
Asymptotics

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!*