Welcome!

CSCI 4041
Algorithms and Data Structures
Spring 2014

Instructor: Amy Larson

This course is about the science of computation.

You will learn to analyze and evaluate your problem solving approach in the context of a computational device (in other words, be critical of your programming solution).
Data + Algorithm = function

\[ f: \{0/1\}^* \rightarrow \{0/1\}^* \]
Think about representing the problem and deriving the solution. What are the space-time-accuracy trade-offs you would have to make (and why)?

- Google Maps (Fastest Route)
- Predict a Brain-Drug Interaction
- RC Car Controller App for iPhone (using gyros)
What is Computer Science?

"Computer science is no more about computers than astronomy is about telescopes." Edsger Dijkstra
You Have a Job Interview
An Optimization Problem

Brute Force

How will you rate these?
Search Space
Maximization Problem
(The solution is the one with the highest possible score.)

Greedy
best local part of best global

X-axis
1000 shirts, sorted by rank

Y-axis
1000 suits, sorted by rank

Z-axis
Score of shirt/suit combo

THE solution (i.e. max score)
Search Spaces
One Solution but Not the "Greedy" One

- Anomalous solution
- Follow the gradient
- Local maxima
Search Spaces
Multiple Solutions

all viable, equally good, solutions
Approaches

• Brute Force
  – typically not viable for large datasets.

• Greedy

• Constraint Satisfaction
  – narrow choices by eliminating those not meeting criteria.

• Decision Tree or Recommender System
  – answer yes/no questions or rate other items to narrow choices.

• Search (AI Techniques)
  – explore decision space in informed (efficient) way.

• Heuristics
  – use human knowledge to guide search.
This Course

• Sorting
  – A necessary tool for managing and navigating a variety of datasets.
  – A great context in which to learn computer science
    • Analyze Complexity
    • Prove Correctness

• Data Structures
  – Explore variety of structures beyond arrays and linked lists.
  – How you store data affects how easily you can manage and navigate it.

• Optimization Problems
  – Learn sophisticated methods that are alternates to exponential brute force approach.

• Graph Problems
  – Applicable to wide range of problems (many seemingly unrelated problems map to a graph problem).
  – Another great context in which to learn computer science.
Mechanics of Course

• Who
  – Bobby Davis, Agoritsa Polyzou, Fenggang Wu

• When and Where
  – LECTURE: TuTh 1:00pm-2:15pm in KHK 3-210 (i.e. Here)
  – RECITATION: Fri (10:10a,11:15,12:20,1:25,2:30)
  – FINAL EXAM: 10:30am-12:30pm, Friday May 16th (KHK 3-210)

• Textbook

• Websites
  – CSELABS: www.cselabs.umn.edu/classes/Spring-2014/csci4041
Schedule


• Schedule is subject to change, particularly with respect to topic content. Check often.

• Midterm schedule will not change.

• Final date and time will not change.

    Check your schedule immediately for conflicts.
    Let me know as soon as possible if you have one.
56% = 12 Assignments and Quizzes
  - not weighted equally
  - no programming

24% = 2 Midterms (open book)

20% = Final (open book)

* There will be no rounding of percentages. “[“ is inclusive and “)“ is exclusive.
• Policies are detailed on the syllabus and Moodle quiz.

• The Moodle quiz on policies is required.
  – Available on Moodle site
  – Due: Tuesday, Jan 28th at 11:55pm. (takes about 30 minutes)

• Please ask if you have any questions!
Policies

(These policies assure fairness across all students.)

- No Late Submissions. *Exceptions
- No Make-Up Exams. *Exceptions
- Demonstration of autonomous proficiency is required (a passing grade of >60% on all exams meets this requirement).
  - A low exam grade will require an evaluation by instructor to determine proficiency. (If this happens, we will meet to discuss.)
- Reasonable effort on all homework is required. If effort is questionable, we will meet to discuss.
- Regrades within 10 days of availability of graded item, except on final, which has no time limit.*

*Exceptions under special circumstances with as much advance notice as possible. In case of illness, stay home please.
Expectations

• Make reasonable effort in coursework.

• Take responsibility for due dates and requirements.

• Take responsibility for your work.

• You read and understood the policies of the course.

• Be respectful during class by not preventing your neighbors from hearing or concentrating (i.e. don’t talk, excessively type, watch distracting videos, play video games, ... You get the idea! ...)

• Respect the educational process, your classmates, and teaching staff by maintaining academic integrity.
Academic Integrity

• Maintain academic integrity by working independently when required.
  – Do your own work.
  – Do not share your work (until after everyone has handed it in).
  – Don’t use the web for homework.

• With classmates, you can discuss
  – course concepts and material.
  – the Problem but not the solution.

The Regent's Policy on Student Conduct, specifically Section IV, Subd. 1. Scholastic Dishonesty, addresses these issues and can be found at http://www1.umn.edu/regents/policies/academic/Student_Conduct_Code.pdf.

Incidents of cheating will be reported to the Director of Graduate or Undergraduate Studies in the department and to the appropriate parties at the college and university levels.
Education is not an easy endeavor, but it can be engaging.

Please start early and ask questions often.
Weekly assignments should keep you on track.

You can be confused in my office!
You can start with

“I'm so confused,
I don't even know what question to ask!”
Correctness and Asymptotic Growth in The Sorting Problem
Analyzing Your Solution

• In Context of Insertion Sort and Merge Sort

• Proving Correctness
  – Loop Invariants
  – Induction

• Bounding Asymptotic Growth (i.e. efficiency)
  – Big-Theta, Big-O, and Big-Omega
  – Growth and Recursion
    • Substitution Method
    • Recursion Tree
    • Master Theorem
**INSERTION SORT**

**INSERTION-SORT(A) {**
  
  for j = 2 to A.length
  
  key = A[j];
  
  i = j - 1;
  
  while i > 0 and A[i] > key
  
  A[i + 1] = A[i];
  
  i = i - 1;
  
  A[i + 1] = key;

**}**


Take a minute to Try It On: [35, 23, 28, 5, 14, 50]
INSERTION-SORT(A) { 
    for j = 2 to A.length
        key = A[j];
        i = j-1;
        while i>0 and A[i]>key
            A[i+1] = A[i];
            i = i - 1;
        A[i+1] = key;
    }

Sorting on a *key*: satellite data attached to that record.


Why are we swapping every time?

How could we store this data to eliminate swapping?

SORTING IN PLACE
Correct?

INPUT: A sequence of n numbers \( \{ a_1, a_2, a_3, \ldots, a_n \} \)

OUTPUT: A permutation \( \{ a'1, a'2, a'3, \ldots, a'n \} \) such that
\[
a'1 \leq a'2 \leq a'3 \leq \ldots \leq a'n
\]

PROOF OF CORRECTNESS using \textit{Loop Invariants} and \textit{Induction}.

**Loop Invariant**

A condition that is true at the start of each iteration.
(It ultimately defines the correctness of the proof).

**Loop Invariant of INSERTION SORT**

At start of iteration \( j \),
Using Loop Invariants and Induction

- **START with IMPLICATION ("Maintenance")**
  
  - Loop Invariant at Iteration $i$
  - IF or WHEN this is TRUE
  - Assume it is TRUE for iteration $i$
  - THEN this is TRUE
  - Show iteration $i+1$ is TRUE when iteration $i$ is TRUE.

- **Show loop invariant is true at iteration 1 ("Initialization")**

- **Describe loop invariant in terms of final iteration to show correctness ("Termination").**

  *Note that we know the loop invariant is TRUE at the final iteration because of the above 2 steps.*
INPUT: A sequence of \( n \) numbers \( \{ a_1, a_2, a_3, \ldots, a_n \} \)

OUTPUT: A permutation \( \{ a'_1, a'_2, a'_3, \ldots, a'_n \} \) such that
\[
a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n
\]
( for all \( i \neq j, \ i \leq j \rightarrow A[i] \leq A[j] \) )

**Loop Invariant of INSERTION SORT**
At start of iteration \( j \), \( A[1]..A[j-1] \) contain original elements in sorted order.

1. INSERTION-SORT(A) {
2. for \( j = 2 \) to \( A.length \)
3. key = \( A[j] \);
4. \( i = j - 1 \);
5. while \( i > 0 \) and \( A[i] > key \)
7. \( i = i - 1 ; \)
8. \( A[i+1] = key ; \)
9. }

**Develop Proof of Correctness** *(if you like, consult neighbor)*
- **Maintenance**
  - Define implication to prove.
  - Prove it is TRUE.
- **Initialization**
  - Determine initial value.
  - SHOW it is true.
- **Termination**
  - Discuss values at termination.
Analyzing Your Solution

- In Context of Insertion Sort and Merge Sort
- Proving Correctness
  - Loop Invariants
  - Induction
- Bounding Asymptotic Growth (i.e. efficiency)
  - Big-Theta, Big-O, and Big-Omega
  - Growth and Recursion
    - Substitution Method
    - Recursion Tree
    - Master Theorem
# Execution Time of Insertion Sort

<table>
<thead>
<tr>
<th></th>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{cost})</td>
<td>(c_1)</td>
<td>(n)</td>
<td>(n)</td>
</tr>
<tr>
<td>(\text{times})</td>
<td>(c_2)</td>
<td>(n - 1)</td>
<td>(n - 1)</td>
</tr>
<tr>
<td>(\text{sum}_{j=2}^{n} t_j)</td>
<td>(\sum_{j=2}^{n} t_j)</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
</tr>
</tbody>
</table>

---

**Asymptotic growth of number of times** while condition tested for that value of \(j\)

**Executed variable** number of times.

**Ames.**
Analysis of Running Time

How does the execution time increase with respect to an increase of input size $n$?

**f(n)** : function describing execution time (lines of code)

**$\Theta(g(n))$** : definition of a set of functions bounded by $g(n)$

$$f(n) \in \Theta(g(n))$$

$f(n)$ increases at a rate bounded above and below by $cg(n)$ for some constant $c$
Sets of Asymptotic Growth

If \( f(n) = \Omega(g(n)) \) AND \( f(n)=O(g(n)) \) then \( f(n) = \Theta(g(n)) \)
Analysis of Running Time

How does the execution time increase with respect to an increase of input size $n$?

$10n + 23 \in \Theta(n)$

$f(n) : \text{execution time (lines of code)}$
$
\Theta(g(n)) : \text{Set definition for function growth}$

$f(n) \in \Theta(g(n))$

$f(n)$ increases at a rate bounded above and below by $cg(n)$ for some constant $c$
Linear and Polynomial

$c^*g(n) > f(n)$ for all $n > 420$
Exponential Growth (Not Tractable)
Log VS Linear VS Polynomial
Log VS Linear VS Polynomial VS Exponential

\[ 1.4 \times 10^{29} \]

Graph showing the comparison of logarithmic, linear, polynomial, and exponential functions.
Analysis of Running Time

Things to Think About

• Computational Model
  – memory
  – multi-core systems – shared memory
  – parallel computing – each has own memory

• Variability due to starting configuration of input
  – Worst-Case ( produce tight upper bound )
  – Best-Case ( produce tight lower bound )
  – Average-Case ( produce tight upper and lower bound )
    • Requires understanding of Expected Value

2x core ≠ 2x as fast

If not THETA ...
Sets of Asymptotic Growth

If \( f(n) = \Omega(g(n)) \) AND \( f(n) = O(g(n)) \) then \( f(n) = \Theta(g(n)) \)

**Big-Omega**

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

\( c_1g(n) \leq f(n) \leq c_2g(n) \)

**Big-Theta**

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

\( c_1g(n) \leq f(n) \leq c_2g(n) \)

**Big-O**

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

\( f(n) \leq cg(n) \)
Tight Bounds

100n^2 + 2n + 20 ∈ Θ(n^2) which means
100n^2 + 2n + 20 ∈ O(n^2)
100n^2 + 2n + 20 ∈ Ω(n^2)

True, but useful?
100n^2 + 2n + 20 ∈ O(n^3)
100n^2 + 2n + 20 ∈ Ω(n)
Execution Time of Insertion Sort

<table>
<thead>
<tr>
<th>Best Case</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`INSERTION-SORT(A)`
1. for `j = 2` to `A.length`
2. `key = A[j]`
3. `// Insert A[j] into the sorted sequence A[1 .. j - 1]`
4. `i = j - 1`
5. while `i > 0` and `A[i] < key`
7. `i = i - 1`
8. `A[i + 1] = key`

- `c1` = `n`
- `c2` = `n - 1`
- `c3` = `n - 1`
- `c4` = `n - 1`
- `c5` = `n - 1`
- `c6` = `n - 1`
- `c7` = `n - 1`
- `c8` = `n - 1`

What do you EXPECT this value to be for each `j`?

number of times while condition tested for that value of `j`
Expected Values

• What is expected value of random variable $X$?
  
  – $E[x] = \sum (x \cdot Pr\{X = x\})$.
  
  – $X$ in range $\{1..5\}$.
  
  – $Pr\{X = x\} = 1/5$, for all $x$ (uniform probability distribution)
  
  – $E[x] = 1\cdot1/5 + 2\cdot1/5 + 3\cdot1/5 + 4\cdot1/5 + 5\cdot1/5 = 3$ (i.e. the mean)

• Indicator Random Variable
  
  – $I\{A\} = \{1: \text{if } A \text{ occurs. } 0: \text{if } A \text{ does not occur. }\}$
  
  – $X_A = I\{A\}$. $E[X_A] = E[I\{A\}]$.
  
  – If there are a series of events, we can calculate the expected number of those events occurring ...
    
    – $X = \sum X_i$.
    
    – $E[X] = E[\sum X_i] = \sum E[X_i]$  
      
      = $\sum (1 \cdot Pr\{X_i \text{ occurs }\} + 0 \cdot Pr\{X_i \text{ doesn’t occur}\})$
    
    – $E[X] = \sum Pr\{X_i \text{ occurs }\}$
Analyzing Your Solution

• In Context of Insertion Sort and **Merge Sort**

• Proving Correctness
  – Loop Invariants
  – Induction

• Bounding Asymptotic Growth (i.e. efficiency)
  – Big-Theta, Big-O, and Big-Omega

  – **Growth and Recursion**
    • Substitution Method
    • Recursion Tree
    • Master Theorem
Recursive Solutions
(Divide and Conquer)

• Recursive call is when a program calls itself.
  Sometimes conceptually cleaner, sometimes not.

Steps of Divide-and-Conquer
• Divide into a number of subproblems.
• Conquer each by solving recursively.
• Combine or merge solutions to subproblems.

```
MERGE-SORT(A, p, r)
    if p < r        // check for base case
        q = ⌊(p + r)/2⌋  // divide
        MERGE-SORT(A, p, q)  // conquer
        MERGE-SORT(A, q + 1, r)  // conquer
        MERGE(A, p, q, r)  // combine
```
Merge Sort (Recursive Sorting Algorithm)

**Merge-Sort**($A,p,r$)

if $p < r$

$q = \lfloor (p + r)/2 \rfloor$

**Merge-Sort**($A, p, q$)

**Merge-Sort**($A, q + 1, r$)

**Merge**($A, p, q, r$)

$n_1 = q - p + 1$

$n_2 = r - q$

let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays

for $i = 1$ to $n_1$

$L[i] = A[p + i - 1]$

for $j = 1$ to $n_2$

$R[j] = A[q + j]$

$L[n_1 + 1] = \infty$

$R[n_2 + 1] = \infty$

$i = 1$

$j = 1$

for $k = p$ to $r$

if $L[i] \leq R[j]$

$A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$

**Loop Invariant**

At each iteration $k$, $A[p..k-1]$ are sorted and contain smallest elements of $L$ and $R$. $L[i]$ and $R[j]$ are smallest elements not yet in $A$.

(Proof in book.)

**Time Complexity of Merge**

$n = r - p + 1 = n_1 + n_2$; \( \Theta(n) \)

What about **Merge-Sort**??

B1.7
Growth and Recursion

T(n) : execution time of the recursive algorithm

Want to Show $T(n) \in O(g(n))$

Merge Sort

$T(n) = T(\text{floor}(n/2)) + T(\text{ceil}(n/2)) + c_2n + c_1$

```
MERGE-SORT(A, p, r)
if p < r
    q = [(p + r)/2]  // check for base case
    T(\text{ceil}(n/2)) : conquer 1
    T(\text{floor}(n/2)) : conquer 2
    c1 : check and divide
    c2 n : merge
    MERGE-SORT(A, p, q)  // divide
    MERGE-SORT(A, q + 1, r)  // conquer
    MERGE(A, p, q, r)  // combine
```
Visualizing the Recursive Calls

### Merge-Sort

```
MERGE-SORT(A, p, r)
if p < r // check for base case
    q = ⌊(p + r)/2⌋ // divide
    MERGE-SORT(A, p, q) // conquer
    MERGE-SORT(A, q + 1, r) // conquer
    MERGE(A, p, q, r) // combine
```

### Recurrence Tree

Each level = one call to the recursive algorithm
Each node = exec. time for divide and combine with respect to “n”
Each leaf has constant execution time (e.g. T(1))

# of levels = # of calls to recursive algorithm
Nodes + Leaves = overall execution time

What is the ratio of leaves to nodes??
### Master Theorem

\[ T(n) = a \cdot T(n/b) + f(n) \]

<table>
<thead>
<tr>
<th>Bound on ( f(n) )</th>
<th>Growth of ( T(n) )</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(n) = O(n^{\log_b a - \varepsilon}) ) for some ( \varepsilon &gt; 0 )</td>
<td>( T(n) = \Theta(n^{\log_b a}) )</td>
<td>leaves dominate ( f(n) ) is slower growing thus upper bound is ( n^{\log_b a} )</td>
</tr>
<tr>
<td>( f(n) = \Theta(n^{\log_b a}) )</td>
<td>( T(n) = \Theta(n^{\log_b a \cdot \log n}) )</td>
<td>leaves and nodes same ( f(n) ) grows at equal rate</td>
</tr>
<tr>
<td>( f(n) = \Omega(n^{\log_b a + \varepsilon}) ) for some ( \varepsilon &gt; 0 ) AND IF ( af(n/b) \leq cf(n) ) for some ( c &lt; 1 )</td>
<td>( T(n) = \Theta(f(n)) )</td>
<td>nodes dominate ( f(n) ) is faster growing thus lower bound is ( n^{\log_b a} )</td>
</tr>
</tbody>
</table>

**You try these ...**
1) MERGE: \( T(n) = 2 \cdot T(n/2) + c_2 n + c_1 \) \( a = 2, b = 2, f(n) = c_2 n + c_1. \)
2) \( T(n) = 6 \cdot T(n/3) + cn \)
3) \( T(n) = 2 \cdot T(n/2) + n \log n \)
Substitution Method for Solving Recurrence Growth

**Inductive Proof**

\[ T(n) = aT(n/b) + f(n) = O(g(n)) \]

\[ T(n) \leq c \times g(n) \]

- Guess at \( g(n) \) and assume true for some \( T(n/b) \).
  - \( T(n/b) = O(g(n)) \)

- Substitute your assumption for \( T(n/b) \) to show true for \( T(n) \).
  - \( T(n) = a \times c \times g(n/b) + f(n) \) manipulate to show \( T(n) \leq c \times g(n) \)

- Show true for some initial value(s) \( T(n_i) \).
  - \( T(n_i) \) AND \( T(n/b) \) -> \( T(n) \) => \( T(n_i \times b) = \)

B1.13
Analyzing Your Solution

- In Context of Insertion Sort and Merge Sort
- Proving Correctness
  - Loop Invariants
  - Induction
- Bounding Asymptotic Growth (i.e. efficiency)
  - Big-Theta, Big-O, and Big-Omega
  - Growth and Recursion
    - Substitution Method
    - Recursion Tree
    - Master Theorem
This Course

• Sorting
  – A necessary tool for managing and navigating a variety of datasets.
  – A great context in which to learn computer science
    • Analyze Complexity
    • Prove Correctness

• Data Structures
  – Explore variety of structures beyond arrays and linked lists.
  – How you store data affects how easily you can manage and navigate it.

• Optimization Problems
  – Learn sophisticated methods that are alternates to exponential brute force approach.

• Graph Problems
  – Applicable to wide range of problems (many seemingly unrelated problems map to a graph problem).
  – Another great context in which to learn computer science.