Data Structures and Algorithms
CMPSC 465

Lectures 1 & 2
Analysis of Algorithms
• Course information
• What are algorithms?
• Why study them?

Adam Smith
Course information

1. Staff
2. Prerequisites
3. Lectures
4. Recitations
5. Textbook
6. Course website
7. Homework
8. Grading policy
9. Collaboration policy
Etymology of “Algorithm”

Abu Abdullah Muhammad ibn Musa
al-Khwarizmi (c. 780 -- 850 AD)

- Persian astronomer and mathematician
- lived in Baghdad, father of algebra
- “On calculating with hindu numerals”
  a treatise in Arabic, 825
- “Agoritmi de numero Indorum”
  translation into Latin, 12th century
- author’s name, mistaken for a plural noun, came to mean “calculation methods”
Algorithm Design and Analysis

Theoretical study of how to solve computational problems

- sorting a list of numbers
- finding a shortest route on a map
- scheduling when to work on homework
- answering web search queries

(Generally: precisely defined set of inputs and, for each input, acceptable outputs)
Algorithms

• Definition: Finite set of unambiguous instructions for solving a problem.
  – An algorithm is correct if on all legitimate inputs, it outputs the right answer in a finite amount of time

• Can be expressed as
  – pseudocode
  – flow charts
  – text in a natural language (e.g. English)
  – computer code
Data Structures

• **Data structures** are ways to store information for which there are **algorithms** for performing particular operations (retrieving/manipulating information), e.g.
  
  – linked lists
  – hash tables
  – arrays
  – trees
  – heaps
Course Objectives

• classical algorithms and data structures
• analysis of algorithms
• standard design techniques
Why study algorithms?

• a *language* for talking about program behavior
• standard set of algorithms and design techniques
• feasibility (what can and cannot be done)
  – halting problem, NP-completeness
• analyzing correctness and resource usage
• successful companies (Google, Mapquest, Akamai)
• computation is fundamental to understanding the world
  – cells, brains, social networks, physical systems all can be viewed as computational devices
• **IT IS FUN!!!**
Performance isn’t everything

- Typical goal: Find most space- and time-efficient algorithm for given problem.
- What else is important?
  - modularity
  - correctness
  - maintainability
  - functionality
  - robustness
  - user-friendliness
  - programmer time
  - simplicity
  - extensibility
  - reliability
Performance isn’t everything

• Typical goal: Find most space- and time-efficient algorithm for given problem.
• Even performance has many facets:
  – type of memory access
  – cache usage
  – network usage
  – parallelism
• This course: simple models, general skills
The problem of sorting

**Input:** sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

**Output:** permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

**Example:**

**Input:** 8 2 4 9 3 6

**Output:** 2 3 4 6 8 9
Insertion Sort

```
INSERTION-SORT(A, n)  ▷ A[1 . . n]
    for j ← 2 to n
do  key ← A[j]
    i ← j - 1
    while i > 0 and A[i] > key
    do  A[i+1] ← A[i]
        i ← i - 1
    A[i+1] = key
```

“pseudocode”
**Insertion Sort**

```
INSERTION-SORT (A, n) ▷ A[1 . . n]
for j ← 2 to n
do  key ← A[j]
i ← j – 1
while i > 0 and A[i] > key
do  A[i+1] ← A[i]
i ← i – 1
A[i+1] = key
```

“A pseudocode”

\[ A: \]

\[ \text{sorted} \]

\[ \text{key} \]
Example of Insertion Sort

8 2 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6
Example of Insertion Sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
Example of Insertion Sort

8  2  4  9  3  6

2  8  4  9  3  6

2  4  8  9  3  6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
Example of Insertion Sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
2  4  8  9  3  6
2  3  4  8  9  6
Example of Insertion Sort

1. 8 2 4 9 3 6
2. 2 8 4 9 3 6
3. 2 4 8 9 3 6
4. 2 4 8 9 3 6
5. 2 3 4 8 9 6

S. Raskhodnikova and A. Smith; based on slides by E. Demaine and C. Leiserson
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9
2 3 4 6 8 9 done
Unit 1

Basics of reasoning about algorithms
Review: mathematical induction

- **Theorem:** All humans have the same height.
- **Inductive formulation:** For all integers \( n \geq 1 \),
  - \( P(n) \): Every set of \( n \) people contains people of a single height.
- **Base case** \( P(1) \): When a set contains only 1 person, everyone in the set has the same height.
- **Inductive step:** Assume \( P(k) \) holds for some \( k \geq 1 \).
  - Consider a set of \( k+1 \) people, \( h_1, \ldots, h_{k+1} \)
    - Since \( P(k) \) holds, sets A and B contain people of the same height
    - Since A and B overlap, people \( h_1, \ldots, h_{k+1} \) all have the same height

What’s wrong with this proof?
Reasoning about Algorithms

• Focus of this course: thinking about algorithms

• Basic tool: mathematical induction
  – Look for iterative structure in the algorithm
  – Try to identify guarantees that (should) hold at each step of the algorithm
  – Use induction to show that the guarantees hold at every step of the algorithm’s execution
Insertion Sort

**Pseudocode**

```plaintext
INSERTION-SORT (A, n)  \[ A[1 \ldots n] \]
for \( j \leftarrow 2 \) to \( n \)
    do  \( key \leftarrow A[j] \)
        \( i \leftarrow j - 1 \)
        while \( i > 0 \) and \( A[i] > key \)
            do \( A[i+1] \leftarrow A[i] \)
                \( i \leftarrow i - 1 \)
        \( A[i+1] = key \)
```

**Diagram**

- **Array** \( A \):
  - 1
  - \( i \)
  - \( j \)
  - \( n \)

- **Key** \( key \)

- **Sorted** elements

---

S. Raskhodnikova and A. Smith; based on slides by E. Demaine and C. Leiserson
Correctness of Insertion Sort

Loop Invariant:
After execution of execution $j$ of `for` loop

1. $A[1 \ldots n]$ is a permutation of the input array, and

2. $A[1 \ldots j]$ is sorted
Loop Invariants

**A tool for analyzing iterative algorithms**
*(example of inductive reasoning)*

Usually, we prove 3 statements

- **Initialization**: invariant holds on first execution
- **Maintenance**: if invariant held on all previous passes through the loop, it holds on current pass
- **Termination**: if invariant holds at the end, then some desired property holds (e.g. algorithm is correct).
Correctness of Insertion Sort

- **Initialization**: $A[1]$ is sorted.
- **Maintenance**: If $A[1..j-1]$ is sorted before pass $j$ through the `for` loop, then $A[1..j]$ is sorted after the pass. This holds because $A[j]$ is inserted in the correct place in $A[1..j-1]$.
  - Proving this formally requires looking carefully at the `while` loop.
- **Termination**: If loop invariant holds at termination ($j = n$), Insertion Sort is correct. Loop invariant states that $A[1..n]$ is sorted when the `for` loop exits. Since the array elements were never changed (only permuted), $A$ now contains the sorted version of the input.
How to measure running time?

- Parameterize the running time by the size of the input, denoted by $n$, since short sequences are easier to sort than long ones.
- Issue: the running time depends on the input: an already sorted sequence is easier to sort.
- Generally, we seek upper bounds on the running time
Kinds of analyses

**Worst-case:** (usually)
- \( T(n) = \) maximum time of algorithm on any input of size \( n \).

**Average-case:** (sometimes)
- \( T(n) = \) expected time of algorithm over all inputs of size \( n \).
- Requires assumption about distribution of inputs.

**Best-case:** (bogus!)
- Cheat with a slow algorithm that works fast on some input.
Machine-independent time

What is Insertion Sort’s worst-case time?

- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

**Big Idea:**

- Ignore machine-dependent constants.
- Look at growth of $T(n)$ as $n \to \infty$.

“Asymptotic Analysis”
Worst case: Input reverse sorted.

\[ T(n) = \sum_{j=2}^{n} c \cdot (j - 1) = cn(n - 1)/2 = \Theta\left(n^2\right) \]

[arithmetic series]

The “\(\Theta\)” notation ignores constants and “low-order” terms. Defined next lecture.

Is insertion sort a fast sorting algorithm?
- Moderately so, for small \(n\).
- Not at all, for large \(n\).