Lecture 1

6.046 / 18.410 J

Introduction to Algorithms

Profs. Ronitt Rubinfeld + Madhu Sudan

http://courses.csail.mit.edu/6.046

Today:

- Introduction
- Course Information
- What & why of Algorithms
- Insertion sort
- Merge sort
- Asymptotic analyses
- Worst case vs. average case
- Recurrences
Al-Khwarizmi "al-khu-raz-mi"

(～780-850 A.D.)
Persian astronomer and mathematician (lived in Baghdad)
Credited with introducing Indian numerals
Father of algebra
Sole linear and quadratic equations and arithmetic
Work treatise entitled (translation)
"Algoritmi de numero Indorum"

Our translation "On Calculation with Hindu Numerals" (read as) or name of author?

Result: "algorythm" originally refers to arithmetic using Arabic numerals
(now: procedures for solving problems or performing computational tasks)

History (timeline):

- "Adam & Eve"
- ～1600 B.C.
- ～300 B.C.
- ～200 B.C.
- 263 A.D.
- 813-833

Algorithm recipes (cooking, rituals, agriculture, etc.)
Factoring, square roots (Babylonia)
Euclid's Algorithm for GCD (see 6.042)
The Sieve of Eratosthenes
Gaussian Elimination (Liu Hui)
Al-Khwarizmi solve linear + quadratic equations
Algorithm Design & Analysis:

- Theoretical study of how to solve computational problems
  
  i.e., sorting a list of numbers, multiplying two integers, factoring an integer, finding a shortest path in a map, finding a longest path in a map...

- Solution = provably correct algorithm

- Algorithm = mathematical abstraction of a computer program

- We'll study tools for:
  
  - designing algorithms
  - analyzing & comparing algorithms' performance & resource usage.
Typical Goal:

What's the fastest (correct) algorithm for a given problem?

- Does one exist?

  * Some problems cannot be solved via computer programs (see more about this in 6.045/18.400)

  e.g., given computer program $P$ and input $x$, does $P$ halt on $x$? "Halting Problem"

- Can we find out the answer in:

  * Our lifetime?
  * 1 year
  * 1 month
  * 1 day

  i.e., is the problem feasible? scalable?

Focus on speed: "Speed is the currency of computing!"
- Speed does not capture:
  - simplicity - programmer time
  - maintainability, extensibility, modularity
  - reliability, robustness
  - user friendliness etc.

- but speed is fun!
Classic Example:

Problem: Sorting

Input: Sequence of numbers \( \langle a_1, a_2, \ldots, a_n \rangle \)

Output: Permutation \( \langle a'_1, a'_2, \ldots, a'_n \rangle \) such that \( a'_1 \leq a'_2 \leq \ldots \leq a'_n \)

E.g. \( 8 \ 2 \ 4 \ 9 \ 3 \ 6 \rightarrow 2 \ 3 \ 4 \ 6 \ 8 \ 9 \)

Insertion-Sort \( (A, n) \)  
\[ \text{Sort} \ A[1..n] \text{ in place} \]

\[ \text{for } j \leftarrow 2 \text{ to } n \]
\[ \quad \text{do} \]
\[ \quad \quad \text{Key} \leftarrow A[j] \]
\[ \quad \quad i \leftarrow j-1 \]
\[ \quad \text{While } i > 0 \text{ and } A[i] > \text{key} \]
\[ \quad \quad \text{do} \]
\[ \quad \quad \quad A[i+1] \leftarrow A[i] \]
\[ \quad \quad \quad i \leftarrow i - 1 \]
\[ \quad \quad A[i+1] \leftarrow \text{key} \]

\[ \text{Invariant: sorted } A[1..j-1] \]
Example:

\[8 \circ 4 \ 9 \ 3 \ 6\]
\[2 \ 8 \ 4 \ 9 \ 3 \ 6\]
\[2 \ 4 \ 8 \ 9 \ 3 \ 6\]
\[2 \ 4 \ 8 \ 9 \ 3 \ 6\]
\[2 \ 3 \ 4 \ 6 \ 8 \ 9\] done

**How to measure running time?**

- depends on input (size of input, already sorted vs. reverse sorted)
- parametrize by size of input

1. **Worst case** *(usually)*
   \[T(n) = \text{max time of algorithm on any input of size } n\]

2. **Average case** *(sometimes)*
   \[T(n) = \text{expected time of algorithm over all inputs of size } n\]
   requires assumption on distribution of inputs!

3. **Best case** *(bad idea)*
Machine Independence

- $T(n)$ defined in terms of "time of algorithm on input"
- but this depends on machine
  (speed, instruction set)
  + exact implementation (as instructions)

Big Idea Asymptotic Analysis

- look at growth of $T(n)$ as $n \to \infty$
- ignore machine-dependent constants

$\Theta$ notation:

$\Theta(g(n)) = \{ f(n) : \exists$ cons. $c_1, c_2, n_0$
  such that $\forall n \geq n_0,$
  $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

"$f(n) = \Theta(g(n))$" means $f(n) \in \Theta(g(n))$

Roughly: drop lower-order terms
Ignore leading constants

E.g., $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$
Asymptotic Analysis:

- When \( n \) gets large enough, \( \Theta(n^2) \) algorithm always beats \( \Theta(n^3) \) alg. (more scalable)

\[
\begin{array}{c}
T(n) \uparrow \\
\Theta(n^2) \quad \Theta(n^3)
\end{array}
\]

- Constants + lower order terms play a role in practice
- nonetheless a powerful tool for analysis + comparison
Insertion Sort analysis:

- **Worst case**: (reverse sorted input)
  \[ T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \] arithmetic series

- **Average case**: (all permutations equally likely)
  \[ T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2) \]

- OK for small \( n \)

- Large \( n \) ?? slow slowow ...
Merge Sort $A[1..n]$:

1. if $n=1$ then done
2. recursively sort $A[1..\lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil+1..n]$
3. merge the two sorted lists

Merge subroutine

- $20\ 12$
- $13\ 11$
- $7\ 9$
- $2\ 1$
- $1$
- $20\ 12$
- $13\ 11$
- $7\ 9$
- $2\ 2$
- $1$
- $20\ 12$
- $13\ 11$
- $7\ 9$
- $11$
- $9$
- $12$
- $+13,20$

$\Theta(n)$ time to merge $n$ elements "linear time"

Merge Sort Analysis

- $- \Theta(1)$ — abuse of notation (need "wrt $n$")
- $- 2T(n/2)$ — sloppy — really $T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$
- $- \Theta(n)$

$T(n) = 2T(n/2) + \Theta(n)$ if $n > 1$

$T(1) = \Theta(1)$

generally assume $T(n) = \Theta(1)$ for $n \leq n_0$
Lecture 4 covers solving recurrences.

Here's one method:

Recursion Tree: \( T(n) = 2T(n/2) + cn \)

\[
T(n) = \begin{cases} 
  cn & \text{if } n = 1 \\
  2T(n/2) + cn & \text{otherwise}
\end{cases}
\]

\[= \underbrace{\ldots}_{\theta(n)} \underbrace{\ldots}_{\theta(n)} \]

\[h = \log n \]

Total: \( T(n) = cn \log n + \theta(n) = \theta(n \log n) \)

- grows slower than \( \theta(n^2) \)
- \( \Rightarrow \) merge sort faster than insertion sort for large enough \( n \)