Design and Analysis of Algorithms

CSE 5311
Lecture 1  Administration & Introduction

Song Jiang, Ph.D.
Department of Computer Science and Engineering
Administration

- **Course CSE 5311**
  - What: Design and Analysis of Algorithms
  - When: M/W 1:00pm – 2:20pm
  - Where: PKH 111
  - Who: Song Jiang (song.jiang@uta.edu)
  - Office Hour: Mon. & Wed. 10:00 ~ 11:00pm at ERB 101
  - or by appointments
  - Homepage: http://ranger.uta.edu/~sjiang/CSE5311-fall-18/index.htm
    (Please visit this website regularly)

- **About your instructor**
  - Research areas: file and storage system, operating system, parallel and distributed computing, and high performance computing,
Study Materials

• **Prerequisites**
  – CSE 2320 Algorithms and Data Structures or its equivalents
  – Mathematical background on summations, sets, relations, probability, and matrix computation.

• **Text book**

Grading

• Distribution
  – 5% Class attendance
  – 30% Homework Assignments
  – 20% Quizzes
  – 20% Midterm Exam
  – 25% Final Exam
  100%

• Attention
  – Homework is as important as any other aspects of your grade!
  – Attendance is strongly encouraged.
  – The university makeup policy will be strictly adhered to. Generally, no make-up exams/quizzes except for university sanctioned reasons.
  – When missing an exam/quiz due to unavoidable circumstances, please notify the instructor and request a makeup approval ahead of time.
Grading

• Late Assignments

  Late assignments will be accepted with a 20% penalty applied for each day late up to 2 days. Assignments submitted later than 2 days after the original due time will not be accepted.

• Collaboration Policy

  Students are allowed and encouraged to collaborate on homework assignments. However, You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. If you obtain a solution through research (e.g., on the Web), acknowledge your source, but write up the solution in your own words. It is a violation of this policy to submit a problem solution that you cannot orally explain to the instructor or GTA.
Final Grade

• **Final Letter Grade**
  – [90 100] --- A
  – [80 90) --- B
  – [70 80) --- C
  – [60 70) --- D
  – [00 60) --- F

• **Note**
  – [ ] denotes inclusion and ( ) denotes exclusion.
  – Your final weighted scores may be curved before assignment of your letter grade.
What’s the Course About?

• The theoretical study of analysis and design of computer algorithms
  – Analysis: predict the cost of an algorithm in terms of resources and performance
  – Design: design algorithms which minimize the cost

• Basic goals for an algorithm
  – Always correct
  – Always terminates

• Our class: performance
Why study algorithms and performance?

- Algorithms help us to understand **scalability**.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a **language** for talking about program behavior.
- Performance is the **currency** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!
Machine Model

• **Generic Random Access Machine (RAM)**
  – Executes operations sequentially
  – Set of primitive operations: Arithmetic, Logical, Comparisons, Function calls

• **Simplifying assumption**
  – All operations cost 1 unit
  – Eliminates dependence on the speed of our computer
  – Otherwise impossible to verify and to compare
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 \( a'_1 \leq a'_2 \leq \ldots \leq a'_n \).

**Example:**

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

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

```
INSERTION-SORT (A, n)  \(\triangleright A[1 \ldots n]\)
  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)  \(\triangleright A[1 \ldots n]\)

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"
Example of insertion sort

8 2 4 9 3 6

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson
Introduction to Algorithms
Example of insertion sort

8 2 4 9 3 6
Example of insertion sort

8 → 2
2 8 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

\[
\begin{array}{ccccccc}
8 & 2 & 4 & 9 & 3 & 6 \\
2 & 8 & 4 & 9 & 3 & 6 \\
2 & 4 & 8 & 9 & 3 & 6 \\
2 & 4 & 8 & 9 & 3 & 6
\end{array}
\]
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

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

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 done
Running time

• The running time depends on the input: an already sorted sequence is easier to sort.
• Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
• Generally, we seek upper bounds on the running time, because everybody likes a guarantee.
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 \( n \).
- Need assumption of statistical 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 \( \text{growth} \) of \( T(n) \) as \( n \to \infty \).

“Asymptotic Analysis”
**Θ-notation**

**Math:**
\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \} \]

**Engineering:**
- Drop low-order terms; ignore leading constants.
- Example: \[ 3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3) \]
Asymptotic performance

When \( n \) gets large enough, a \( \Theta(n^2) \) algorithm always beats a \( \Theta(n^3) \) algorithm.

- We shouldn’t ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.
Insertion sort analysis

**Worst case:** Input reverse sorted.

\[
T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \quad \text{[arithmetic series]}
\]

**Average case:** All permutations equally likely.

\[
T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)
\]

*Is insertion sort a fast sorting algorithm?*

- Moderately so, for small \(n\).
- Not at all, for large \(n\).
Merge sort

**MERGE-SORT** \(A[1 . . n]\)

1. If \(n = 1\), done.
2. Recursively sort \(A[1 . . \lfloor n/2 \rfloor]\) and \(A[\lceil n/2 \rceil + 1 . . n]\).
3. “Merge” the 2 sorted lists.

*Key subroutine:* **MERGE**
Merging two sorted arrays

20 12
13 11
7  9
2  1
Merging two sorted arrays

20 12
13 11
7 9
2 1
1
Merging two sorted arrays

20 12 20 12
13 11 13 11
7 9 7 9
2 1 2
1
Merging two sorted arrays

\[
\begin{array}{c|c}
20 & 12 \\
13 & 11 \\
7 & 9 \\
1 & 2 \\
\end{array} \quad \begin{array}{c|c}
20 & 12 \\
13 & 11 \\
7 & 9 \\
2 & 2 \\
\end{array}
\]
Merging two sorted arrays

20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11
7  9 ||  7  9 ||  7  9
2  1 ||  2  2 ||
Merging two sorted arrays

20 12 | 20 12 | 20 12
13 11 | 13 11 | 13 11
7 9   | 7 9   | 7 9
2 1   | 2     | 7
1 2   |       | 7
Merging two sorted arrays

20 12 || 20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11 || 13 11
7 9 || 7 9 || 7 9 || 9
2 1 || 2 || 7 || 9
1 2 7

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson
Introduction to Algorithms
Merging two sorted arrays

20 12 || 20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11 || 13 11
7 9 || 7 9 || 7 9 || 9
2 1 || 2 || 7 || 9
1 || 2 || 7 || 9
Merging two sorted arrays

20 12 || 20 12 || 20 12 || 20 12 || 20 12
13 11 || 13 11 || 13 11 || 13 11 || 13 11
7 9 || 7 9 || 7 9 || 9 9 || 9 9
2 1 || 2 2 || 7 7 || 9 9 || 
1 1 || 2 2 || 7 7 || 9 9 ||
Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2

7

9

11
Merging two sorted arrays
Merging two sorted arrays

20 12 | 20 12 | 20 12 | 20 12 | 20 12 | 20 12 | 20 12
13 11 | 13 11 | 13 11 | 13 11 | 13 11 | 13 11 | 13 11
7 9 | 7 9 | 7 9 | 7 9 | 7 9 | 7 9 | 7 9
2 1 | 2 1 | 2 1 | 2 1 | 2 1 | 2 1 | 2 1
1 2 | 1 2 | 1 2 | 1 2 | 1 2 | 1 2 | 1 2

1 2 | 7 9 | 9 11 | 11 12

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson
Introduction to Algorithms

September 7, 2005
Merging two sorted arrays

Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).
Analyzing merge sort

\( T(n) \) | **MERGE-SORT** \( A[1 \ldots n] \)
--- | ---
\( \Theta(1) \) | 1. If \( n = 1 \), done.
\( 2T(n/2) \) | 2. Recursively sort \( A[1 \ldots \lceil n/2 \rceil] \) and \( A[\lfloor n/2 \rfloor + 1 \ldots n] \).
\( \Theta(n) \) | 3. "Merge" the 2 sorted lists

**Abuse**: Should be \( T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) \), but it turns out not to matter asymptotically.
Recurrence for merge sort

\[ T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1; \\
2T(n/2) + \Theta(n) & \text{if } n > 1.
\end{cases} \]

• We shall usually omit stating the base case when \( T(n) = \Theta(1) \) for sufficiently small \( n \), but only when it has no effect on the asymptotic solution to the recurrence.

• CLRS and Lecture 2 provide several ways to find a good upper bound on \( T(n) \).
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{align*}
T(n) &= cn \\
     &= cn/2 + cn/4 + cn/4 + \ldots + cn/4 \\
     &= \Theta(1)
\end{align*}
\]
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{align*}
T(n) &= cn \\
&= cn/2 + cn/2 \\
&= cn/4 + cn/4 + cn/4 + cn/4 \\
&= \Theta(1)
\end{align*}
\]
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{align*}
h &= \lg n \\
cn/2 & \quad cn/2 & \quad cn/2 & \quad cn/2 \\
\vdots & & \vdots & \\
\Theta(1) & \\
\end{align*}
\]
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \lg n$

$\Theta(1)$

#leaves = $n$

$\Theta(n)$

Total = $\Theta(n \lg n)$
Conclusions

- $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.
- Go test it out for yourself!