Design and Analysis of Algorithms

CSE 5311
Lecture 2  Algorithms and Growth Functions

Junzhou Huang, Ph.D.
Department of Computer Science and Engineering
Administration

• Course CSE5311
  – What: Design and Analysis of Algorithms
  – When: Friday 1:00 ~ 3:50pm
  – Where: ERB 130
  – Who: Junzhou Huang (Office ERB 650) jzhuang@uta.edu
  – Office Hour: Friday 3:50 ~ 5:50pm and/or appointments
  – Homepage: http://ranger.uta.edu/~huang/teaching/CSE5311.htm
    (You’re required to check this page regularly)

• Lecturer
  – PhD in CS from Rutgers, the State University of New Jersey
  – Research areas: machine learning, computer vision, medical image analysis and bioinformatics

• GTA
  – Saiyang Na (Office ERB 403), sxn3892@mavs.uta.edu
  – Office hours: Friday 10:00am ~ 12:00pm and/or appointments
Reviewing: Study Materials

• Prerequisites
  – Algorithms and Data Structure (CSE 2320)
  – Theoretical Computer Science (CSE 3315)
  – What this really means:
    ➢ You have working experience on software development.
    ➢ You know compilation process and programming
    ➢ Elementary knowledge of math and algorithms

• Text book
  – https://mitpress.mit.edu/books/introduction-algorithms
Reviewing: What?

• The theoretical study of design and analysis of computer algorithms

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

• Our class: performance
  – Performance often draws the line between what is possible and what is impossible.

• Design and Analysis of Algorithms
  – **Analysis:** predict the cost of an algorithm in terms of resources and performance
  – **Design:** design algorithms which minimize the cost
Reviewing: 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
Reviewing: Running Time

• Running Time
  – Depends on the input
  – An already sorted sequence is easier to sort.

• Major Simplifying Convention
  – Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
  – \( T_A(n) = \) time of \( A \) on length \( n \) inputs. Generally, we seek upper bounds on the running time, to have a guarantee of performance.

• 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:** (Never) Cheat with a slow algorithm that works fast on some input.
Machine-independent Time

• Question
  – Machine-independent Time

• Idea
  – Ignore machine dependent constants, otherwise impossible to verify and to compare algorithms
  – Look at growth of $T(n)$ as $n \to \infty$.

“Asymptotic Analysis”
Recall: \( \Theta \)-notation

**Definition:**

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

**Basic Manipulations:**

- 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.

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

\[ T(n) \]

\[ n \quad n_0 \]
Insertion Sort Analysis

**Worst case:** Input reverse sorted.

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

[arithmetic series]

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

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

Is insertion sort a fast sorting algorithm?

- Moderately so, for small \( n \).
- Not at all, for large \( n \).
Integer Multiplication

- Let $X = \begin{bmatrix} A & B \end{bmatrix}$ and $Y = \begin{bmatrix} C & D \end{bmatrix}$ where $A, B, C$ and $D$ are $n/2$ bit integers

- **Simple Method:**
  
  $XY = (2^{n/2}A+B)(2^{n/2}C+D) = 2^n AC + 2^{n/2}(AD+BC) + BD$

- **Running Time Recurrence**
  
  $T(n) < 4T(n/2) + \Theta(n)$

  Recursive Calls Addition and Shift

- **Solution** $T(n) = \Theta(n^2)$
Integer Multiplication

\[ T(n) = \begin{cases} 
0 & \text{if } n = 0 \\
4T(n/2) + n & \text{otherwise} 
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\lg n} n 2^k = n \left( \frac{2^{1+\lg n} - 1}{2 - 1} \right) = 2n^2 - n \]

Assume \( n \) is a power of 2.

\[
\begin{array}{c}
T(n) \\
\downarrow \\
T(n/2) \quad T(n/2) \quad T(n/2) \quad T(n/2) \\
\downarrow \\
T(n/4) \quad T(n/4) \quad T(n/4) \quad T(n/4) \\
\downarrow \\
\vdots \\
\vdots \\
\downarrow \\
T(n/2^k) \\
\vdots \\
\downarrow \\
T(2) \quad T(2) \quad T(2) \quad T(2) \\
\end{array}
\]
Better Integer Multiplication

- Let \( X = \begin{array}{c|c} A & B \end{array} \) and \( Y = \begin{array}{c|c} C & D \end{array} \) where \( A, B, C \) and \( D \) are \( n/2 \) bit integers

- [Karatsuba-Ofman 1962] : Can multiply two \( n \)-bit integers in \( O(n \log^3 3) \) bit operations.

\[
XY = (2^{n/2}A+B)(2^{n/2}C+D) = 2^n AC + 2^{n/2}(AD+BC)+BD
= (2^n - 2^{n/2})AC + 2^{n/2}(A+B)(C+D) + (1 - 2^{n/2}) BD
\]

- **Running Time Recurrence**

\[
T(n) < 3T(n/2) + \Theta(n)
\]

- **Solution:** \( T(n) = O(n^{\log 3}) \)
Better Integer Multiplication

\[
T(n) = \begin{cases} 
0 & \text{if } n = 0 \\
3T(n/2) + n & \text{otherwise} 
\end{cases}
\]

Assume \( n \) is a power of 2

\[
T(n) = \sum_{k=0}^{\lfloor \log_3 n \rfloor} n \left( \frac{3}{2} \right)^k = n \left( \frac{\left( \frac{3}{2} \right)^{\lfloor \log_3 n \rfloor} - 1}{\frac{3}{2} - 1} \right) = 3n^{\log_3 3} - 2n
\]
Merge Sort

MERGE-SORT  $A[1 \ldots n]$

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

Key subroutine: MERGE
Merging Two Sorted Arrays
Merging Two Sorted Arrays

20  12
13  11
  7  9
  2  1
   1
Merging Two Sorted Arrays

<table>
<thead>
<tr>
<th>20 12</th>
<th>20 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 11</td>
<td>13 11</td>
</tr>
<tr>
<td>7 9</td>
<td>7 9</td>
</tr>
</tbody>
</table>

2 1

1
Merging Two Sorted Arrays

20 12 | 20 12
13 11 | 13 11
 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 2 ||
1 2 ||
Merging Two Sorted Arrays

<table>
<thead>
<tr>
<th>20 12</th>
<th>20 12</th>
<th>20 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 11</td>
<td>13 11</td>
<td>13 11</td>
</tr>
<tr>
<td>7 9</td>
<td>7 9</td>
<td>7 9</td>
</tr>
<tr>
<td>2 1</td>
<td>2 2</td>
<td>7 7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
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 7 9
2 1 2 1 2 1 2

1 2 7 2

Merging Two Sorted Arrays

\begin{array}{cccc}
20 & 12 & 20 & 12 \\
13 & 11 & 13 & 11 \\
7 & 9 & 7 & 9 \\
2 & 1 & 2 & 7 \\
1 & 2 & 7 & 9 \\
\end{array}
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   |  7  9   |  7  9
2  1   |  2  1   |  7  9   |  9      |  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 7 9 7 9
2 1 2 2 1 2 2 1 2
1 2 7 9 9 9 9 9 9
7 9 9 9 9 9 9 9 9
1 2 7 9 9 9 9 9 9

Merging Two Sorted Arrays
Merging Two Sorted Arrays

20 12
13 11
7 9
2 1

20 12
13 11
7 9
2 2

20 12
13 11
7 9
2 7

20 12
13 11
7 9
2 9

20 12
13 11
7 9
2 11

20 12
13 11
7 9
2 12
Merging Two Sorted Arrays

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

\[ T(n) \]
\[ \Theta(1) \]
\[ 2T(n/2) \]
\[ \Theta(n) \]

MERGE-SORT \( A[1 \ldots n] \)

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

**Sloppiness:** 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.

- Next Lecture will 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.

$h = \log n$

\[ cn \]
\[ \cdots \]
\[ \Theta(1) \]
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 = $n$

$\Theta(n)$
Recursion Tree

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

- $\Theta(n \lg n)$ 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.