Lecture 1
Analysis of Algorithms
• Insertion sort
• Asymptotic analysis
• Merge sort
• Recurrences

Prof. Charles E. Leiserson
Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

What’s more important than performance?

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time
- simplicity
- extensibility
- reliability
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!
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)  \(\triangleright\) 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** $(A, n) \implies A[1 \ldots n]$

for $j \leftarrow 2$ to $n$

do $key \leftarrow A[j]$

$i \leftarrow j - 1$

while $i > 0$ and $A[i] > key$

do $A[i+1] \leftarrow A[i]$

$i \leftarrow i - 1$

$A[i+1] = key$

---

$pseudocode$
Example of insertion sort

8 2 4 9 3 6
Example of insertion sort
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

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are m choices for each of a_1, a_2, …, a_r, but once these are chosen, exactly one choice for a_0 causes x and y to collide, namely

\[
\left( x_0 - y_0 \right) 1 = -a_i (x_i - y_i) \mod m. 
\]

Thus, the number of h's that cause x and y to collide is

\[
m r \cdot 1 = |H|/m.
\]
Example of insertion sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6

Q. How many h's cause x and y to collide?

A. There are m choices for each of a_1, a_2, …, a_r, but once these are chosen, exactly one choice for a_0 causes x and y to collide, namely

\[
\sum_{i=1}^{r} \cdot (x_i - y_i) \mod m.
\]

Thus, the number of h's that cause x and y to collide is m \cdot 1 = m \cdot \frac{|H|}{m}.
Example of insertion sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are m choices for each of a₁, a₂, …, aᵣ, but once these are chosen, exactly one choice for a₀ causes x and y to collide, namely

\[
\sum_{i=1}^{r} (x_i - y_i) \mod m.
\]

Thus, the number of h's that cause x and y to collide is mᵣ = |H|/m.
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

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>4</th>
<th>9</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
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  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 *growth* of $T(n)$ as $n \to \infty$.

“*Asymptotic Analysis*”
Θ-notation

Math:
Θ(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** \( A[1 \ldots n] \)

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

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

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

20 12 13 11 7 9 2 1

20 12 13 11 7 9 2

Proof (completed)

Q. How many h's cause x and y to collide?
A. There are \( m \) choices for each of \( a_1, a_2, \ldots, a_r \), but once these are chosen, exactly one choice for \( a_0 \) causes \( x \) and \( y \) to collide, namely \( \beta \cdot \gamma \cdot \ldots \cdot a_i(x_i y_i) \mod m \).

Thus, the number of h's that cause \( x \) and \( y \) to collide is \( m r = |H|/m \).
Merging two sorted arrays

20  12
13  11
 7   9
 2  1
 1  2

20  12
13  11
 7   9
 2  1
 1  2

Proof (completed)

Q. How many h's cause x and y to collide?
A. There are m choices for each of a₁, a₂, …, aᵣ, but once these are chosen, exactly one choice for a₀ causes x and y to collide, namely:

\[
\sum_{i=1}^{r} (x_i - y_i) \equiv a_0 \pmod{m}.
\]

Thus, the number of h's that cause x and y to collide is mᵣ = |H|/m.
Merging two sorted arrays

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

Thus, the number of $h$'s that cause $x$ and $y$ to collide is $m^r \cdot 1 = m^r = |H|/m$. 
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 1 || 2 1
1 2 || 2 1 || 7 1
Merging two sorted arrays

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

Proof (completed)

Q. How many h's cause x and y to collide?
A. There are m choices for each of a₁, a₂, …, aᵣ, but once these are chosen, exactly one choice for a₀ causes x and y to collide, namely

\[
\sum_{i=1}^{r} \cdot (x_i - y_i) = -a_0 \cdot (x_0 - y_0) \mod m.
\]

Thus, the number of h's that cause x and y to collide is m · 1 = m = |H| / m.
Merging two sorted arrays

20 12 13 11 7 9 2 1
1 2

20 12 13 11 7 9 2 1
2 2

20 12 13 11 7 9 2 1
7 9

20 12 13 11 7 9 2 1
9

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are m choices for each of a1, a2, …, ar, but once these are chosen, exactly one choice for a0 causes x and y to collide, namely

\[
\sum_{i=1}^{r} a_i (x_i - y_i) \pmod{m}.
\]

Thus, the number of h's that cause x and y to collide is

\[
m r \cdot 1 = m r = |H|/m.
\]
Merging two sorted arrays

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

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are \( m \) choices for each of \( a_1, a_2, \ldots, a_r \), but once these are chosen, exactly one choice for \( a_0 \) causes \( x \) and \( y \) to collide, namely

\[
\left( x_0 - y_0 - a_i \right) \mod m.
\]

Thus, the number of h's that cause \( x \) and \( y \) to collide is \( m \cdot 1 = m \cdot m = |H|/m \).
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
1 7

20 12
13 11
9 9
2 9

20 12
13 11
11 11
2 11

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are m choices for each of a1, a2, …, ar, but once these are chosen, exactly one choice for a0 causes x and y to collide, namely

\[
\begin{bmatrix}
    \cdot \\
    \cdot \\
\end{bmatrix}
\]

Thus, the number of h's that cause x and y to collide is m·1 = m = |H|/m.
Merging two sorted arrays

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

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

Proof (completed)

Q. How many h's cause x and y to collide?

A. There are m choices for each of a_1, a_2, …, a_r, but once these are chosen, exactly one choice for a_0 causes x and y to collide, namely

\[
\begin{align*}
\left( x_0 - y_0 \right) \mod m.
\end{align*}
\]

Thus, the number of h's that cause x and y to collide is m r · 1 = m r = |H|/m.

October 5, 2005

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
### Merging two sorted arrays

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

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

**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.
- 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.
Recursion tree

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

$h = \lg n$

\[ \Theta(1) \]
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.
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

- $\Theta(n \log n)$ 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!