Analysis of Algorithms
Algorithms are Everywhere

Examples

• Maps
• Fedex
• Biology
• Physics
• Computer Operating Systems
• Self-Driving Cars
• Determining if you should get a job/loan/school admission
• Regulating your heart
• Space Shuttle
• Machine Learning
• ...
Why is this the right time to study algorithms?

- Mathematical understanding
- fast computers
- ability to get algorithm implementations to users
- good interfaces
What do we study in this class

- Given a problem, we find the right algorithm
- We use math
- We prove that our work is right
- We keep an eye on practice/implementation, but our goal is to solve the clean well-defined problem.
What are the skills most people need

- Given a new problem, how do we design an algorithm
- Knowing what is efficient and what is not, to help you
  - model problems
  - use existing algorithms
  - decide which algorithms to extend
  - realize when a problem is too hard to solve quickly
First problem to consider: Matrix Multiplication

\[ C = A \cdot B \]

\[
\begin{bmatrix}
3 & 1 & 1 \\
2 & 0 & 3 \\
\end{bmatrix}
\begin{bmatrix}
1 & 6 \\
2 & 0 \\
1 & 2 \\
\end{bmatrix} =
\begin{bmatrix}
6 & 20 \\
5 & 18 \\
\end{bmatrix}
\]
Algorithm for Matrix Multiplication

\[ C = A \cdot B \]

\[
\begin{bmatrix}
3 & 1 & 1 \\
2 & 0 & 3
\end{bmatrix}
\begin{bmatrix}
1 & 6 \\
2 & 0 \\
1 & 2
\end{bmatrix} =
\begin{bmatrix}
6 & 20 \\
5 & 18
\end{bmatrix}
\]

Write pseudocode

1 // input: A, an \( n \times m \) matrix and B, an \( m \times p \) matrix
2 // output: C, an \( n \times p \) matrix
3 for \( i = 1 \) to \( n \)
4 for \( j = 1 \) to \( p \)
5 \[ C[i, j] = 0 \]
6 for \( k = 1 \) to \( m \)
7 \[ C[i, j] + = A[i, k] \cdot B[k, j] \]
Analysis

1 // input: A, an $n \times m$ matrix and B, an $m \times p$ matrix
2 // output: C, an $n \times$ matrix
3 for $i = 1$ to $n$
4 for $j = 1$ to $p$
5 $C[i, j] = 0$
6 for $k = 1$ to $m$
7 $C[i, j] += A[i, k] \cdot B[k, j]$

Running time

- 3 nested loops
- $O(nmp)$ time
- if $n = m = p$, then $O(n^3)$ time
- Lower bound of $\Omega(n^2)$
Can we do better?

• We are implementing the standard definition efficiently, what else could we do?

• You have to do $n^3$ operation, each of $n^2$ entries of $C$, involves adding up the result of $n$ multiplications.
Can we do better?

- We are implementing the standard definition efficiently, what else could we do?
- You have to do $n^3$ operation, each of $n^2$ entries of $C$, involves adding up the result of $n$ multipications.

Maybe divide and conquer can help

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$$r = ae + bf \quad (1)$$
$$s = ag + bh \quad (2)$$
$$t = ce + df \quad (3)$$
$$u = cg + dh \quad (4)$$
Maybe divide and conquer can help

\[
\begin{bmatrix}
  a & b \\
  c & d \\
\end{bmatrix}
\begin{bmatrix}
  e & g \\
  f & h \\
\end{bmatrix} =
\begin{bmatrix}
  r & s \\
  t & u \\
\end{bmatrix}
\]

\[
\begin{align*}
  r &= ae + bf \\
  s &= ag + bh \\
  t &= ce + df \\
  u &= cg + dh
\end{align*}
\]

Multiply 2 \( n \times n \) matrices takes

- 8 multiplications of \( n/2 \times n/2 \) matrices
- 4 additions of \( n/2 \times n/2 \) matrices

- Adding two \( n \times n \) matrices takes \( O(n^2) \) time
- Adding matrices seems easier than multiplying them
Let’s Analyze

Let $T(n)$ be the time to multiply $2^n$ by $n$ matrices

$$T(n) = \begin{cases} 
8T(n/2) + 4(n/2)^2 & \text{if } n > 1 \\
1 & \text{if } n = 1
\end{cases}$$
Let's Analyze

Let \( T(n) \) be the time to multiply 2 \( n \) by \( n \) matrices

\[
T(n) = \begin{cases} 
8T(n/2) + 4(n/2)^2 & \text{if } n > 1 \\
1 & \text{if } n = 1 
\end{cases}
\]

As we will learn, this solves to \( O(n^3) \).

But consider the following recurrence

\[
T(n) = \begin{cases} 
7T(n/2) + 18(n/2)^2 & \text{if } n > 1 \\
1 & \text{if } n = 1 
\end{cases}
\]

As we will learn, this solves to \( O(n \log_2 7) = O(n^{2.81\ldots}) \).

But can we multiply 2 \( n \times n \) matrices by doing 7 multiplications of \( n/2 \times n/2 \) matrices and 18 additions of \( n/2 \times n/2 \) matrices.
Strassen’s Algorithm

To Compute

\[ r = ae + bf \] \hspace{1cm} (9)
\[ s = ag + bh \] \hspace{1cm} (10)
\[ t = ce + df \] \hspace{1cm} (11)
\[ u = cg + dh \] \hspace{1cm} (12)

Calculations

\[ P_1 = a(g - h) = ag - ah \]
\[ P_2 = (a + b)h = ah + bh \]
\[ P_3 = (c + d)e = ce + de \]
\[ P_4 = d(f - e) = df - de \]
\[ s = P_1 + P_2 \]
\[ t = P_3 + P_4 \]
\[ P_5 = (a + d)(e + h) = ae + ah + de + dh \]
\[ P_6 = (b - d)(h + f) = -dh - df + bh + bf \]
\[ r = P_5 + P_4 - P_2 + P_6 \]
\[ u = P_5 + P_1 - P_3 - P_7 \]
Course Logistics
Another Problem

Investing for someone who knows the Future: You are given the prices of a stock for each of the next $n$ days. You can buy once and sell once and you want to maximize your profit.

<table>
<thead>
<tr>
<th></th>
<th>Day</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>70</td>
<td>90</td>
<td>40</td>
<td>27</td>
<td>69</td>
<td>80</td>
<td>13</td>
<td>35</td>
<td>50</td>
<td>75</td>
<td>51</td>
<td>53</td>
<td>56</td>
<td>10</td>
<td>15</td>
<td>41</td>
<td></td>
</tr>
</tbody>
</table>

Questions:

- How long does the naive algorithm take?
- Can we improve this with divide and conquer?