Why should we study the techniques for algorithm design and algorithm analysis?

and motivating the Master Theorem
(students call it the “Monster Theorem”)
So you want to be a computer scientist?

Produced by mutilating (without permission) great slides made by
Professor Jeff Edmonds,
York University, Canada
Is your goal to be a mundane programmer?
Or a great leader and thinker?
Original Thinking

Boss assigns task:

– Given today’s prices of pork, grain, sawdust, …
– Given constraints on what constitutes a hotdog
– Make the cheapest hotdog!

Everyday industry asks these questions!
Case 1 - Your answer:

- Um? Tell me what to code.

With more suffocated software engineering job market, the demand for mundane programmers will diminish.
Case 2 - Your answer:

• I learned this great algorithm that will work.

Soon all known algorithms will be available in libraries!
Case 3 - Your answer:

• I can develop a new algorithm for you.

Great thinkers will always be needed!
The future belongs to the computer scientist who:

- understands the principles and techniques needed to solve vast array of unfamiliar problems that arise in a rapidly changing field
Course Content

- A list of algorithms.
  - Learn their code.
  - Trace them until you are convinced that they work.
  - Implement them.

We should teach people how to catch fish themselves, not just give them a fish or two!
Course Content

• A survey of algorithmic design techniques
• Abstract thinking
• How to develop new algorithms for any problem that may arise
A survey of fundamental ideas and algorithmic design techniques

For example ...
Start With Some Math

Classifying Functions

\[ f(i) = n^{\Theta(n)} \]

Time Complexity

\[ t(n) = \Theta(n^2) \]

Adding Made Easy

\[ \sum_{i=1} f(i) \]

Recurrence Relations

\[ T(n) = a \ T(n/b) + f(n) \]
Recursive Algorithms
Network Flows

Flow

Augmentation Graph

Path

Cut

cap = 51 = rate
Greedy Algorithms
Dynamic Programming

Solve each subinstance

$s, v_1, v_2, ..., t$
Refinement:
The best solution comes from a process of repeatedly refining and inventing alternative solutions.
A Few Example Algorithms

Grade School Revisited: How To Multiply Two Numbers

2 × 2 = 4
Slides in this section produced from Professor Steven Rudich’s slides
Carnegie Mellon University
(mutilated without Steve’s permission)
😊
Complex Numbers

• Remember how to multiply 2 complex numbers?

• \((a + bi)(c + di) = [ac - bd] + [ad + bc] i\)

• Input: \(a, b, c, d\)     Output: \(ac - bd, ad + bc\)

• If a real multiplication costs $1 and an addition cost a penny. What is the cheapest way to obtain the output from the input?

• Can you do better than $4.02?
Gauss’ $3.05$ Method:
Input: $a,b,c,d$  
Output: $ac-bd, ad+bc$

- $m_1 = ac$
- $m_2 = bd$
- $A_1 = m_1 - m_2 = ac-bd$
- $m_3 = (a+b)(c+d) = ac + ad + bc + bd$
- $A_2 = m_3 - m_1 - m_2 = ad+bc$
Question:

• The Gauss “hack” saves one multiplication out of four. It requires 25% less work.

• Could there be a context where performing 3 multiplications for every 4 provides a more dramatic savings?
How to add 2 n-bit numbers.

+ * * * * * * * * * * * *
  * * * * * * * * * * * * *

--------------------------
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
Time complexity of grade school addition

$T(n) = \text{The amount of time grade school addition uses to add two } n\text{-bit numbers}$

$= \theta(n) = \text{linear time.}$

On any reasonable computer adding 3 bits can be done in constant time.
Is there a faster way to add?

• **QUESTION:** Is there an algorithm to add two n-bit numbers whose time grows sub-linearly in n?

• Any algorithm for addition must read all of the input bits!! Thus, linear time addition is the fastest!
So any algorithm for addition must use time at least linear in the size of the numbers.

Grade school addition is essentially as good as it can be.
How to multiply 2 n-bit numbers.
How to multiply 2 n-bit numbers.

I get it! The total time is bounded by $cn^2$. 
Grade School Addition: $\theta(n)$ time
Grade School Multiplication: $\theta(n^2)$ time

Is there a clever algorithm to multiply two numbers in linear time?
Despite years of research, no one knows!
Is there a faster way to multiply two numbers than the way you learned in grade school?
Divide And Conquer
(an approach to faster algorithms)

• **DIVIDE** my instance to the problem into smaller instances to the same problem.

• **Have a friend** (recursively) solve them. Do not worry about it yourself.

• **GLUE** the answers together so as to obtain the answer to your larger instance.
Multiplication of 2 n-bit numbers

1. $X = \begin{array}{c|c}
a & b \\
\hline
c & d \\
\end{array}$

2. $Y = \begin{array}{c|c}
a & b \\
\hline
c & d \\
\end{array}$

- $X = a \ 2^{n/2} + b$ 
- $Y = c \ 2^{n/2} + d$

- $XY = ac \ 2^n + (ad+bc) \ 2^{n/2} + bd$
Multiplication of 2 n-bit numbers

• $X = \begin{array}{cc} a & b \end{array}$

• $Y = \begin{array}{cc} c & d \end{array}$

• $XY = ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd$

MULT(X,Y):
If $|X| = |Y| = 1$ then RETURN XY
Break X into a;b and Y into c;d
RETURN

$MULT(a,c) \cdot 2^n + (MULT(a,d) + MULT(b,c)) \cdot 2^{n/2} + MULT(b,d)$
Time required by MULT

- $T(n) = \text{time taken by MULT on two n-bit numbers}$

- What is $T(n)$? What is its growth rate? Is it $\theta(n^2)$?
Recurrence Relation

\[ \text{MULT}(X,Y): \]
If \(|X| = |Y| = 1\) then RETURN XY
Break X into a;b and Y into c;d
RETURN

\[ \text{MULT}(a,c) \, 2^n + (\text{MULT}(a,d) + \text{MULT}(b,c)) \, 2^{n/2} + \text{MULT}(b,d) \]

• \(T(1) = m\) for some constant \(m\)
• \(T(n) = 4 \, T(n/2) + k \, n\) for some constant \(k\)

How do we unravel \(T(n)\) so that we can determine its growth rate?
Technique 1: Guess and Verify

- Recurrence Relation:
  \[ T(1) = 1 \quad \text{and} \quad T(n) = 4T(n/2) + kn \]

- Guess: \( T(n) = n^2(1+k) - nk \)

- Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) )</td>
<td>1</td>
</tr>
<tr>
<td>( = (1+k)(1)^2 - k )</td>
<td>( 4T(n/2) + kn )</td>
</tr>
<tr>
<td>( = (k+1) - k = 1 )</td>
<td>( = 4 \left[ (1+k)\left(\frac{n}{2}\right)^2 - k \left(\frac{n}{2}\right) \right] + kn )</td>
</tr>
<tr>
<td>( T(n) )</td>
<td>( (1+k)n^2 - nk )</td>
</tr>
<tr>
<td>( = (1+k)n^2 - kn )</td>
<td>( = (1+k)n^2 - kn )</td>
</tr>
</tbody>
</table>

This is a somewhat silly technique: it is hard to guess right!

We need a **more systematic way** of estimating the solution!
A Better Way - Technique 2:

“Unwind” the Recursion!
\[ T(n) = \frac{n}{2} T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) \]
\[ T(n) = \frac{n}{2} + T(n/4) + T(n/4) + T(n/4) \]
\[ T(n) = \frac{n}{2} \]
\[ T(n) = 4T(n/2) + k \, n = 4(4T(n/4) + k \, n/2) + k \, n = 4^2T(n/2^2) + 2 \, k \, n + k \, n \]

\[ = 4^2 (4T(n/2^3) + k \, n/2^2) + 2 \, k \, n + k \, n \]

\[ = 4^3T(n/2^3) + 2^2 \, k \, n + 2 \, k \, n + k \, n = 4^3T(n/2^3) + k \, n (2^2 + 2 + 1) \]

\[ = 4^3 (4T(n/2^4) + k \, n/2^3) + k \, n (2^2 + 2 + 1) \]

\[ = 4^4 T(n/2^4) + k \, n (2^3 + 2^2 + 2 + 1) = \ldots = \ldots = \ldots \]

\[ = 4^{\log_2 n} T(n/2^{\lfloor \log_2 n \rfloor}) + k \, n (2^{\lfloor \log_2 n \rfloor - 1} + \ldots + 2^2 + 2 + 1) \]

\[ \approx 4^{\log_2 n} T(n/2^{\log_2 n}) + k \, n (2^{(\log_2 n)-1} + \ldots + 2^2 + 2 + 1) \]

\[ = 2^{\log_2 n} T(1) + k \, n (2^{(\log_2 n)-1} + \ldots + 2^2 + 2 + 1) = \]

\[ = n^2 + k \, n (2^{\log_2 n} - 1) \]

\[ = n^2 + k \, n (n - 1) = n^2 (k + 1) - k \, n \]

\[ = \Theta(n^2) \]

- \[2^{\log_2 n} = (2^{\log_2 n})^2 = n^2\]
- \[T(1) = 1\]
- \[2^M + \ldots + 2^2 + 2 + 1\]
- \[(2^M+1 - 1)/(2 - 1)\]
- \[(2^M+1 - 1)\]
Divide and Conquer MULT: $\theta(n^2)$ time
Grade School Multiplication: $\theta(n^2)$ time

All that work for nothing!
MULT(X,Y): MULT revisited
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
RETURN
\[
MULT(a,c)\cdot 2^n + (MULT(a,d) + MULT(b,c))\cdot 2^{n/2} + MULT(b,d)
\]
• MULT calls itself 4 times. Can you see a way to reduce the number of calls?
Gauss’ Hack:

Input: $a, b, c, d$  
Output: $ac, ad + bc, bd$

- $A_1 = ac$
- $A_3 = bd$
- $m_3 = (a+b)(c+d) = ac + ad + bc + bd$
- $A_2 = m_3 - A_1 - A_3 = ad + bc$
Gaussified MULT
(Karatsuba 1962)

\[ T(n) = 3 \ T(n/2) + n \]

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN e2^n + (MULT(a+b, c+d) - e - f) 2^{n/2} + f

\[ \bullet \ T(n) = 3 \ T(n/2) + n \]

\[ \bullet \text{Actually: } T(n) = 2 \ T(n/2) + T(n/2 + 1) + kn \]
\[ T(n) = \frac{n}{2} + T(n/4) + T(n/4) + T(n/4) + T(n/4) \]
\[ T(n) = n = T(n/2) + T(n/2) + T(n/2) \]
\[ T(n) = T(n/2) + T(n/2) + T(n/2) + T(n/4) + T(n/4) + T(n/4) \]
$T(n) = \frac{n}{2} + T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right)$
\[
T(n) = 3T(n/2) + k \cdot n = 3(3T(n/4) + k \cdot n/2) + k \cdot n = 3^2 T(n/2^2) + 3/2 \cdot k \cdot n + k \cdot n
\]
\[
= 3^2 (3T(n/2^3) + k \cdot n/2^2) + 3/2 \cdot k \cdot n + k \cdot n
\]
\[
= 3^3 T(n/2^3) + (3/2)^2 \cdot k \cdot n + 3/2 \cdot k \cdot n + k \cdot n = 3^3 T(n/2^3) + k \cdot n((3/2)^2 + 3/2 + 1)
\]
\[
= 3^3 (3T(n/2^4) + k \cdot n/2^3) + k \cdot n ((3/2)^2 + (3/2) + 1)
\]
\[
= 3^4 T(n/2^4) + k \cdot n ((3/2)^3 + (3/2)^2 + 3/2 + 1) = \ldots = \ldots = \ldots
\]
\[
= 3 \left\lfloor \log_2 n \right\rfloor T(n/2 \left\lfloor \log_2 n \right\rfloor) + k \cdot n ((3/2) \left\lfloor \log_2 n \right\rfloor - 1) + \ldots + (3/2)^2 + 3/2 + 1
\]
\[
\approx 3^{\left\lfloor \log_2 n \right\rfloor} T(n/2^{\log_2 n}) + k \cdot n ((3/2)^{\log_2 n - 1} + \ldots + (3/2)^2 + (3/2) + 1)
\]
\[
= n^{\log_2 3} T(1) + k \cdot n ((3/2)^{\log_2 n - 1} - 1)/(3/2 - 1)
\]
\[
= n^{\log_2 3} + 2 \cdot k \cdot n (n^{\log_2 3/2} - 1) =
\]
\[
= n^{\log_2 3} + 2 \cdot k \cdot n^{\log_2 3} - 2 \cdot k \cdot n
\]
\[
= n^{\log_2 3} (2k + 1) - 2 \cdot k \cdot n
\]
\[
= \Theta(n^{\log_2 3})
\]

- \(3^{\left\lfloor \log_2 n \right\rfloor} = n^{\log_2 3};\ (3/2)^{\left\lfloor \log_2 n \right\rfloor} = n^{\log_2 (3/2)}\)
- \(n \cdot n^{\log_2 (3/2)} = n^{1 + \log_2 (3/2)} = n^{\log_2 2 + \log_2 (3/2)} = n^{\log_2 3}\)
- \(T(1) = 1\)
- \((3/2)^M + \ldots + (3/2)^2 + 3/2 + 1 = ((3/2)^{M+1} - 1)/(3/2 - 1)\)

\[
= ((3/2)^{M+1} - 1)/(1/2) = 2((3/2)^{M+1} - 1)
\]
Conclusions:

1. Dramatic improvement in efficiency of multiplication for large $n$:
   Not just a 25% savings but $\theta(n^2)$ vs $\theta(n^{1.58...})$

2. Estimation of the growth rate of the solution to a recurrence is a messy and lengthy computation; can we make it more uniform and given by a ready made formula?

   Answer: Yes, this is provided by the Master Theorem. Even more importantly, the methods used in the proof of the Master Theorem can be applied even in cases when the theorem itself cannot be applied!