MA407 – Algorithms and Computation

London School of Economics and Political Science

Introduction to the course

What is computation?

What is an algorithm?

What is programming?

What are examples of algorithms?

Do you know any famous algorithms?

What is computation? / What is an algorithm?

“A function on natural numbers is computable if it can be computed by a human computer”

(A. Turing, 1936)

“Algorithms compute in steps of bounded complexity” (A. Kolmogorov, 1953)

What is computation? / What is an algorithm?

"A function on natural numbers is computable if it can be computed by a human computer"

(A. Turing, 1936)

"Algorithms compute in steps of bounded complexity" (A. Kolmogorov, 1953)

Examples of (famous) algorithms:

- The Euclidean Algorithm (computing the greatest common divisor)
- Simplex method (algorithm for linear optimisation)
- RSA (public-key cryptographic algorithm)
- Dijkstra's shortest-path algorithm (navigation)
- MP3 algorithm (audio data compression)
- PageRank (the way Google finds and ranks search results)

What is computation? / What is an algorithm?

“A function on natural numbers is computable if it can be computed by a human computer”

(A. Turing, 1936)

“Algorithms compute in steps of bounded complexity” (A. Kolmogorov, 1953)

Examples of (famous) algorithms:

- The Euclidean Algorithm (computing the greatest common divisor)
- Simplex method (algorithm for linear optimisation)
- RSA (public-key cryptographic algorithm)
- Dijkstra’s shortest-path algorithm (navigation)
- MP3 algorithm (audio data compression)
- PageRank (the way Google finds and ranks search results)

THE JARGON FILE defines “program” as . . .

1. A magic spell cast over a computer allowing it to turn one’s input into error messages.
2. . . .

Computational Problems and Algorithms

Computational Problems and Algorithms

A computational problem can be defined by the correct input–output relationship:

Computational Problems and Algorithms

A computational problem can be defined by the correct input–output relationship:

$$\Pi = \{(I, O) \mid I \text{ is an instance of } \Pi \text{ and } O \text{ is a correct output for } I\}.$$

Computational Problems and Algorithms

A computational problem can be defined by the correct input–output relationship:

$$\Pi = \{(I, O) \mid I \text{ is an instance of } \Pi \text{ and } O \text{ is a correct output for } I\}.$$

So, it is a binary relation: possibly, many-to-one.

Computational Problems and Algorithms

A computational problem can be defined by the correct input–output relationship:

$$\Pi = \{(I, O) \mid I \text{ is an instance of } \Pi \text{ and } O \text{ is a correct output for } I\}.$$

So, it is a binary relation: possibly, many-to-one.

An algorithm is a clearly and precisely described set of instructions (like a recipe) to solve a computational problem.

How do we describe algorithms?

Algorithms can be described in words, or in **pseudocode**.

Example: the Euclidean algorithm to calculate the gcd of two positive integers p and q

How do we describe algorithms?

Algorithms can be described in words, or in **pseudocode**.

Example: the Euclidean algorithm to calculate the gcd of two positive integers p and q

Pseudocode variant 1: high level

iteratively calculate (x_i, y_i) with $x_1 = p$ and $y_1 = q$,
and with $x_{i+1} = y_i$
and $y_{i+1} =$ remainder of x_i divided by y_i;
stop when y_{i+1} is 0, and return x_{i+1}

How do we describe algorithms?

Algorithms can be described in words, or in **pseudocode**.

Example: the Euclidean algorithm to calculate the gcd of two positive integers p and q

Pseudocode variant 2: detailed

Input: two positive integers `p` and `q`

Output: the greatest common divisor of `p` and `q`

```
x = p, y = q
while y is not 0 do
    previous_x = x
    x = y
    y = previous_x mod y
return x
```

How do we describe algorithms?

Algorithms can be described in words, or in **pseudocode**.

Example: the Euclidean algorithm to calculate the gcd of two positive integers p and q

Pseudocode variant 3: detailed

Algorithm 1 The Euclidean Algorithm

```
1: Algorithm GCD(p, q)
2:     a ← p
3:     b ← q
4:     while b ≠ 0 do
5:         t ← a
6:         a ← b
7:         b ← t mod b
8:     return a
```

There are still a few things missing that are needed in a Python program, but close

What can you do in Python?

What is Python?

- A large snake
- A British comedy troupe, Monty Python
- A high-level general-purpose programming language

What is Python?

- A large snake
- A British comedy troupe, Monty Python
- A high-level general-purpose programming language

What can you do in Python?

- read and output text
- calculate
- “draw”
- create a graphical user interface
- access the internet
- store data / manage databases
- sort data
- encrypt data
- determine a shortest route . . .

What is Python?

- A large snake
- A British comedy troupe, Monty Python
- A high-level general-purpose programming language

What can you do in Python?

- read and output text
- calculate
- “draw”
- create a graphical user interface
- access the internet
- store data / manage databases
- sort data
- encrypt data
- determine a shortest route . . .

Many things . . . (Python is “Turing complete”)

Alan Turing, 1912 – 1954

Example: The Euclidean Algorithm

Python code:

```
1  def gcd(p,q):
2      a = p
3      b = q
4      while b != 0:
5          t = a
6          a = b
7          b = t % b
8      return a
```

This is an illustration of a Python program.

You should understand it after our tutorials.

What is Python? (continued)

What can you do in Python? **What will we do in this course?**

- read and output text
- calculate
- “draw”
- create a graphical user interface
- access the internet
- store data / manage databases
- sort data
- encode data
- encrypt data
- determine a shortest route (MA428)
- streaming (processing large data volumes) (MA421)

These are just some examples.
We will do more . . .

Python basics: The presessional

Often important

How do we store and organise the data?

- A data structure is a way of storing and organising data so as to facilitate different types of access and manipulations.
- No single data structure works well for all purposes
- So it is important to know the strengths and limitations of several of them.

Tentative topics

- Week 1: Introduction
- Week 2: The sorting algorithms Insertion Sort and Merge Sort, and introduction to proving the correctness of algorithms
- Week 3: Runtime analysis, big-O notation, recurrences, Heapsort
- Week 4: Quicksort and Randomised Quicksort
- Week 5: A lower bound for sorting algorithms, sorting in linear time
- Week 6: Introduction to basic data structures, linked lists
- Week 7: Stacks, Queues, Hashing
- Week 8: Greedy algorithms
- Week 9: Dynamic programming
- Week 10: Online algorithms

The sequel to MA407: MA421 Topics in Algorithms

MA421 Topics in Algorithms will cover

- NP-completeness
- Approximation Algorithms
- Randomised Algorithms
- Streaming Algorithms
- Random Structures and Average-Case Analysis of Algorithms

The sequel to MA407: MA421 Topics in Algorithms

MA421 Topics in Algorithms will cover

- NP-completeness
- Approximation Algorithms
- Randomised Algorithms
- Streaming Algorithms
- Random Structures and Average-Case Analysis of Algorithms

There will be some Python programming in MA421, but not as much as in MA407.

The sequel to MA407: MA421 Topics in Algorithms

MA421 Topics in Algorithms will cover

- NP-completeness
- Approximation Algorithms
- Randomised Algorithms
- Streaming Algorithms
- Random Structures and Average-Case Analysis of Algorithms

There will be some Python programming in MA421, but not as much as in MA407.

The focus is on more advanced techniques, tools and subjects in the study of algorithms.

The sequel to MA407: MA421 Topics in Algorithms

MA421 Topics in Algorithms will cover

- NP-completeness
- Approximation Algorithms
- Randomised Algorithms
- Streaming Algorithms
- Random Structures and Average-Case Analysis of Algorithms

There will be some Python programming in MA421, but not as much as in MA407.

The focus is on more advanced techniques, tools and subjects in the study of algorithms.

More technical proofs compared to MA407.

Course logistics

(For additional info and materials see Moodle Page)

Lectures and classes

Lectures

- On-campus lectures on Wednesdays, 14:00-16:00 (FAW.2.04)

Lectures and classes

Lectures

- On-campus lectures on Wednesdays, 14:00-16:00 (FAW.2.04)

Seminars (in CBG.1.05)

- Seminar Group 1: Fridays, 14:00-15:00
- Seminar Group 2: Fridays, 15:00-16:00

Lectures and classes

Lectures

- On-campus lectures on Wednesdays, 14:00-16:00 (FAW.2.04)

Seminars (in CBG.1.05)

- Seminar Group 1: Fridays, 14:00-15:00
- Seminar Group 2: Fridays, 15:00-16:00

Note: Seminars start in Week 1, this week. They are **compulsory**.

Lectures and classes

Lectures

- On-campus lectures on Wednesdays, 14:00-16:00 (FAW.2.04)

Seminars (in CBG.1.05)

- Seminar Group 1: Fridays, 14:00-15:00
- Seminar Group 2: Fridays, 15:00-16:00

Note: Seminars start in Week 1, this week. They are **compulsory**.

In Week 11, there will be no new material, only the last class (and, possibly, a revision lecture if desired).

Self-study Python tutorials

I hope you have been engaging with Python training on Dataquest and enjoying it!

Self-study Python tutorials

I hope you have been engaging with Python training on Dataquest and enjoying it!

There are also be a few self-study tutorials to go over the basics of Python programming.

They also briefly cover

(a) how to use Python on the LSE computers or how to install Python on your personal machine

(b) the very basics of Python programming to set you up for the programming that you will do in the first few weeks.

Python usage

- The LSE computers should have Python installed.
- Programs can be edited with Notepad++ (or other text editors)

Python usage

- The LSE computers should have Python installed.
- Programs can be edited with Notepad++ (or other text editors)

- If you can, working on your own computer is recommended
- You can use any IDE (Integrated Development Environment), Anaconda, Jupyter
- For a simple IDE, you can try Wing 101, which is free and linked on Moodle

Problem sets

- There will be weekly problem sets.
- Mix of theoretical and practical questions.

- Will be released after the seminars.
- Will be due the following week.
- First problem set has been released and is due this Friday.

- They will be marked for feedback, and contribute to the final mark as continuous assessment.
- You will submit your solutions on a platform called Gradescope.
- Please write your name, class group number, and class teacher's name on solutions that you turn in.

Lectures

Purpose of the lectures

- Hit the most important points of the material
- Focus on design and analysis of algorithms

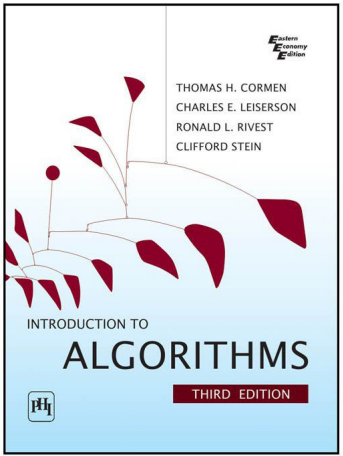
Material provided

- Slides
- Pointers to relevant chapters/sections in the course's main textbook
  - Read those sections!

Textbook

Cormen, Leiserson, Rivest, Stein.
Introduction to Algorithms, 3rd edition. MIT Press.
Available online via LSE Library.

In addition:
Kleinberg, Tardos. Algorithm Design, 1st edition. Pearson.



Seminars

Purpose of the seminars

- Discuss the solutions to the most recent homework
- Introduce additional material/examples as time permits

Material provided

- Written solutions
- Python code

Assessment

Written examination (65%)

- Standard, closed-book in-person exam
- In the January examination period

Assessed coursework (25%)

- Released towards the end of AT
- “Big” problem set
- You will have a couple of weeks to solve it (due soon after AT ends)

Continuous assessment (10%)

- Based on your submissions on weekly homeworks
- “Good enough” submission get 1.25 marks
- Maximum total mark for continuous assessment is 10

How to be successful in this course

Start working early! Expect a decent amount of self study!

- Catch up with Python
- Do the problem sets
- Recap right after lectures
- Read relevant book chapters

Ask questions!

- Ask questions in lectures, seminars, office hours
- Use the discussion forum on Moodle

Do you know how to multiply integers?

Outline

Problem specification

Primary-school multiplication

Karatsuba's algorithm

Problem Specification

The multiplication problem:

Input: Two n-digit integers x, y

Output: Their product $x \cdot y$

E.g.: for $n = 3$, $x = 359$, $y = 724$, $x \cdot y = 259916$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$\begin{array}{rrr} & 1 & 2 \\ \times & 3 & 2 \\ \hline \end{array}$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$\begin{array}{rcc} & 1 & 2 \\ \times & 3 & 2 \\ \hline 3 & 6 & 0 \end{array}$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{rrr}
 & 1 & 2 \\
\times & 3 & 2 \\
\hline
3 & 6 & 0 \\
 & 2 & 4 \\
\hline
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{rrr}
 & 1 & 2 \\
\times & 3 & 2 \\
\hline
3 & 6 & 0 \\
 & 2 & 4 \\
\hline
3 & 8 & 4
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$\begin{array}{rrr} & 2 & 3 \\ \times & 1 & 4 \\ \hline \end{array}$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$\begin{array}{rrr} & 2 & 3 \\ \times & 1 & 4 \\ \hline 2 & 3 & 0 \end{array}$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{rrr}
 & 2 & 3 \\
\times & 1 & 4 \\
\hline
2 & 3 & 0 \\
 & 9^{1} & 2 \\
\hline
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{rrr}
 & 2 & 3 \\
\times & 1 & 4 \\
\hline
2 & 3 & 0 \\
 & 9^{1} & 2 \\
\hline
3 & 2 & 2
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$\begin{array}{rrrr} & 3 & 5 & 9 \\ \times & 7 & 2 & 4 \\ \hline \end{array}$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
 & & & & & \\
2 & 5^{4} & 1^{6} & 3 & 0 & 0
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
2 & 5^{4} & 1^{6} & 3 & 0 & 0 \\
 & & 7^{1} & 1^{1} & 8 & 0
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
 2 & 5^{4} & 1^{6} & 3 & 0 & 0 \\
 & & 7^{1} & 1^{1} & 8 & 0 \\
 & & 1 & 4^{2} & 3^{3} & 6 \\
\hline
\end{array}
$$

The primary-school algorithm

Actually, you already know an algorithm for this problem from primary school.

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
2 & 5^{4} & 1^{6} & 3 & 0 & 0 \\
 & & 7^{1} & 1^{1} & 8 & 0 \\
 & & 1 & 4^{2} & 3^{3} & 6 \\
\hline
2 & 5 & 9 & 9 & 1 & 6
\end{array}
$$

First remark

This may seem very basic to you, but was a big deal when it was introduced!

First remark

This may seem very basic to you, but was a big deal when it was introduced!

Just compare:

$$359 \times 724 \quad \text{vs} \quad \mathrm{CCCLIX} \times \mathrm{DCCXXIV}$$

Second remark

It is actually where the word “algorithm” comes from!

Second remark

It is actually where the word “algorithm” comes from!

The Persian mathematician Al-Khwarizmi ($\sim$800AD) wrote a book that describes how to do multiply two integers written in arabic numerals.

Second remark

It is actually where the word “algorithm” comes from!

The Persian mathematician Al-Khwarizmi (∼800AD) wrote a book that describes how to do multiply two integers written in arabic numerals.

Originally, “algorisme” (old French) referred to just the Arabic number system, but eventually it came to mean “algorithm” as we know today.

So how good is this algorithm?

So how good is this algorithm?

Let's look at our example, where we had $n = 3$, once more:

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
2 & 5^{4} & 1^{6} & 3 & 0 & 0 \\
 & & 7^{1} & 1^{1} & 8 & 0 \\
 & & 1 & 4^{2} & 3^{3} & 6 \\
\hline
2 & 5 & 9 & 9 & 1 & 6
\end{array}
$$

So how good is this algorithm?

Let's look at our example, where we had $n = 3$, once more:

$$
\begin{array}{cccccc}
 & & & 3 & 5 & 9 \\
 & & \times & 7 & 2 & 4 \\
\hline
2 & 5^{4} & 1^{6} & 3 & 0 & 0 \\
 & & 7^{1} & 1^{1} & 8 & 0 \\
 & & 1 & 4^{2} & 3^{3} & 6 \\
\hline
2 & 5 & 9 & 9 & 1 & 6
\end{array}
$$

We multiply each digit from y with each digit from x, so we need at least n^2 one-digit operations.

So how good is this algorithm?

Let's look at our example, where we had $n = 3$, once more:

			3	5	9
		$\times$	7	2	4
2	5^4	1^6	3	0	0
		7^1	1^1	8	0
		1	4^2	3^3	6
2	5	9	9	1	6

We multiply each digit from y with each digit from x, so we need at least n^2 one-digit operations.

...this is a lower bound!

So how good is this algorithm?

A more careful analysis shows:

Theorem. There exists a constant $c > 0$ such that no matter which two n-digit numbers numbers x, y we multiply, at most $c \cdot n^2$ one-digit operations are needed.

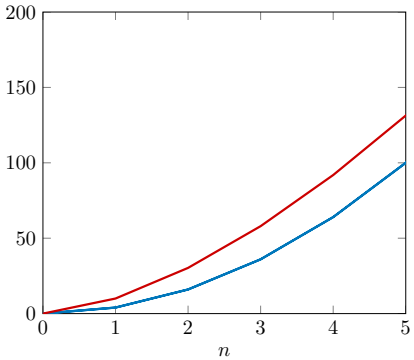
Long multiplication: how long?

Methodology

- When bounding the running time, we focus on the worst case for a given input size n
- We try to understand how the running time scales with input size n

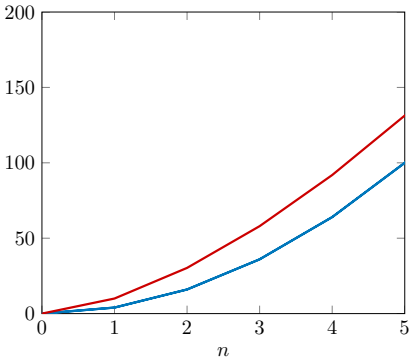
Why is this sensible?

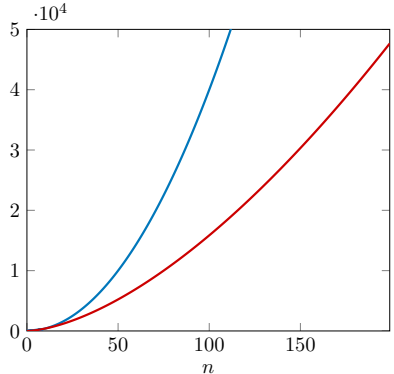
This gives robust guarantees across environments, and puts emphasis on large inputs:



Why is this sensible?

This gives robust guarantees across environments, and puts emphasis on large inputs:





$f_1(n) = 4 \cdot n^2$ $f_2(n) = 10 \cdot n^{1.6}$

Can we do better?

In 1960, the Russian mathematician Kolmogorov conjectured that it is impossible to devise an algorithm that scales better with n than n^2.

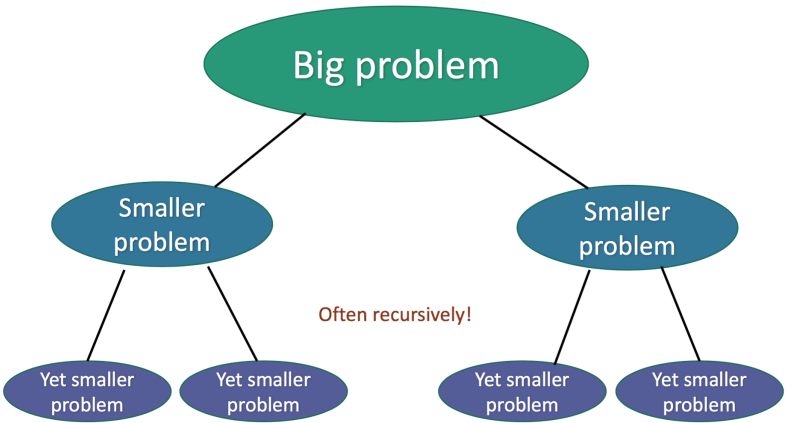
Can we do better?

In 1960, the Russian mathematician Kolmogorov conjectured that it is impossible to devise an algorithm that scales better with n than n^2.

Within a week the then 23-year old student Karatsuba proved him wrong!!

Divide-and-conquer

Idea: Break problem up into smaller (easier) subproblems



Divide and conquer for multiplication

Assume for now (for simplicity) that n is an even number.

For example $n = 4$ and

$$x = 1234 \quad \text{and} \quad y = 5678$$

Divide and conquer for multiplication

Assume for now (for simplicity) that n is an even number.

For example $n = 4$ and

$$x = 1234 \quad \text{and} \quad y = 5678$$

Write

$$x = 12 \cdot 100 + 34 \quad \text{and} \quad y = 56 \cdot 100 + 78$$

Then:

$$x \cdot y = (12 \cdot 56) \cdot 100^2 + (12 \cdot 78 + 34 \cdot 56) \cdot 100 + 34 \cdot 78$$

Divide and conquer for multiplication

Assume for now (for simplicity) that n is an even number.

For example $n = 4$ and

$$x = 1234 \quad \text{and} \quad y = 5678$$

Write

$$x = 12 \cdot 100 + 34 \quad \text{and} \quad y = 56 \cdot 100 + 78$$

Then:

$$x \cdot y = (12 \cdot 56) \cdot 100^2 + (12 \cdot 78 + 34 \cdot 56) \cdot 100 + 34 \cdot 78$$

Divide and conquer for multiplication

More generally:

Break up the n-digit integers x, y into two $n/2$-digit integers each:

$$[x_1, \ldots, x_n] = \underbrace{[x_1, \ldots, x_{n/2}]}_{=a} \cdot 10^{n/2} + \underbrace{[x_{n/2+1}, \ldots, x_n]}_{=b}$$

$$[y_1, \ldots, y_n] = \underbrace{[y_1, \ldots, y_{n/2}]}_{=c} \cdot 10^{n/2} + \underbrace{[y_{n/2+1}, \ldots, y_n]}_{=d}$$

Divide and conquer for multiplication

More generally:

Break up the n-digit integers x, y into two $n/2$-digit integers each:

$$[x_1, \ldots, x_n] = \underbrace{[x_1, \ldots, x_{n/2}]}_{=a} \cdot 10^{n/2} + \underbrace{[x_{n/2+1}, \ldots, x_n]}_{=b}$$

$$[y_1, \ldots, y_n] = \underbrace{[y_1, \ldots, y_{n/2}]}_{=c} \cdot 10^{n/2} + \underbrace{[y_{n/2+1}, \ldots, y_n]}_{=d}$$

Then:

$$\begin{aligned} x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= (a \cdot c) \cdot 10^n + (a \cdot d + b \cdot c) \cdot 10^{n/2} + (b \cdot d) \end{aligned}$$

A recursive algorithm

Multiply(x,y):

If $n = 1$

- Return $x \cdot y$

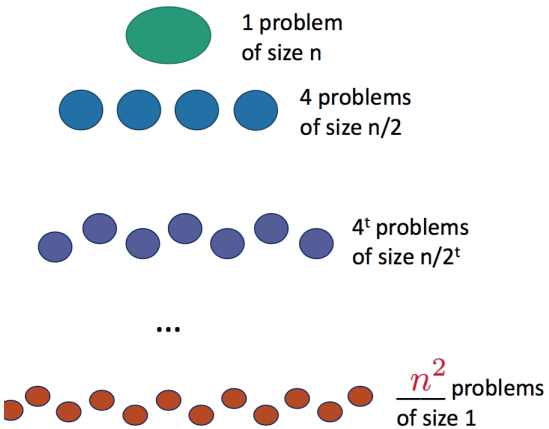
Else

- Write $x = a \cdot 10^{n/2} + b \quad (a = \lfloor x/10^{n/2} \rfloor, b = x \bmod 10^{n/2})$
- Write $y = c \cdot 10^{n/2} + d \quad (c = \lfloor x/10^{n/2} \rfloor, d = x \bmod 10^{n/2})$
- $ac =$ Multiply(a, c)
- $\ldots$
- Return $x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$

No improvement!

Theorem. For every n there exist two n-digit integers x, y such that this algorithm requires at least n^2 one-digit operations.

Proof sketch



- If you cut n in half $\log_2(n)$ times, you get down to 1
- So we do this $\log_2(n)$ times and get $4^{\log_2(n)} = n^2$ problems of size 1

Another way to see this

Let $T(n)$ be the time to multiply two n-digit numbers in this method.

Another way to see this

Let $T(n)$ be the time to multiply two n-digit numbers in this method.

Recurrence relation:

$$T(n) = 4 \cdot T(n/2) + \text{(in the order of } n \text{ to do additions)}$$

Another way to see this

Let $T(n)$ be the time to multiply two n-digit numbers in this method.

Recurrence relation:

$$T(n) = 4 \cdot T(n/2) + \text{(in the order of } n \text{ to do additions)}$$

Ignore the "(in the order of n)" term for now.

Another way to see this

Let $T(n)$ be the time to multiply two n-digit numbers in this method.

Recurrence relation:

$$T(n) = 4 \cdot T(n/2) + \text{(in the order of } n \text{ to do additions)}$$

Ignore the "(in the order of n)" term for now.

Then,

$$\begin{aligned} T(n) &= 4 \cdot T(n/2) \\ &= 4 \cdot (4 \cdot T(n/4)) \\ &= \dots \\ &= 4^{\log_2(n)} \cdot T(1) = n^2 \cdot T(1) \end{aligned}$$

More carefully...

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 4^2 \cdot T\left(\frac{n}{2^2}\right) + (2+1) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 4^2 \cdot T\left(\frac{n}{2^2}\right) + (2+1) \cdot cn \\
&= 4^2 \cdot \left(4 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{4}\right)\right) + (2+1) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 4^2 \cdot T\left(\frac{n}{2^2}\right) + (2+1) \cdot cn \\
&= 4^2 \cdot \left(4 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{4}\right)\right) + (2+1) \cdot cn \\
&= 4^3 \cdot T\left(\frac{n}{2^3}\right) + (4+2+1) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 4^2 \cdot T\left(\frac{n}{2^2}\right) + (2+1) \cdot cn \\
&= 4^2 \cdot \left(4 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{4}\right)\right) + (2+1) \cdot cn \\
&= 4^3 \cdot T\left(\frac{n}{2^3}\right) + (4+2+1) \cdot cn \\
&\vdots \\
&= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + cn \\
&\vdots \\
&= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn
\end{aligned}
$$

Assume $n = 2^k$

More carefully...

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

$$\vdots$$

$$= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn$$

Assume $n = 2^k$

$$= n^2 \cdot T(1) + (2^k - 1) \cdot cn$$

More carefully...

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

$$\vdots$$

$$= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn$$

Assume $n = 2^k$

$$= n^2 \cdot T(1) + (2^k - 1) \cdot cn$$
$$= n^2 \cdot T(1) + (n - 1) \cdot cn$$

More carefully...

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

$$\vdots$$

$$= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn$$

Assume $n = 2^k$

$$= n^2 \cdot T(1) + (2^k - 1) \cdot cn$$
$$= n^2 \cdot T(1) + (n - 1) \cdot cn$$
$$= n^2 \cdot (T(1) + c) - cn$$

More carefully...

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn$$

$$\vdots$$

$$= 4^k \cdot T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 4 + 2 + 1) \cdot cn$$

Assume $n = 2^k$

$$= n^2 \cdot T(1) + (2^k - 1) \cdot cn$$
$$= n^2 \cdot T(1) + (n - 1) \cdot cn$$
$$= n^2 \cdot (T(1) + c) - cn$$
$$\approx c' n^2$$

Karatsuba's insight

Recall:

$$x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$$

Karatsuba's insight

Recall:

$$x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$$

Instead of recursively computing four terms ac, ad, bc, and bd, we can do the following:

Karatsuba's insight

Recall:

$$x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$$

Instead of recursively computing four terms ac, ad, bc, and bd, we can do the following:

First, recursively compute:

$$ac, bd, \text{ and } (a + b)(c + d)$$

Karatsuba's insight

Recall:

$$x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$$

Instead of recursively computing four terms ac, ad, bc, and bd, we can do the following:

First, recursively compute:

$$ac, bd, \text{ and } (a+b)(c+d)$$

Then, compute (using only addition)

$$bc + ad = (a+b)(c+d) - ac - bd$$

Karatsuba's insight

Recall:

$$x \cdot y = (ac) \cdot 10^n + (ad + bc) \cdot 10^{n/2} + (bd)$$

Instead of recursively computing four terms ac, ad, bc, and bd, we can do the following:

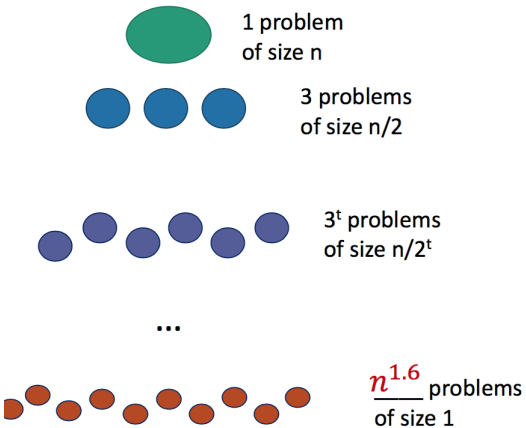
First, recursively compute:

$$ac, bd, \text{ and } (a+b)(c+d)$$

Then, compute (using only addition)

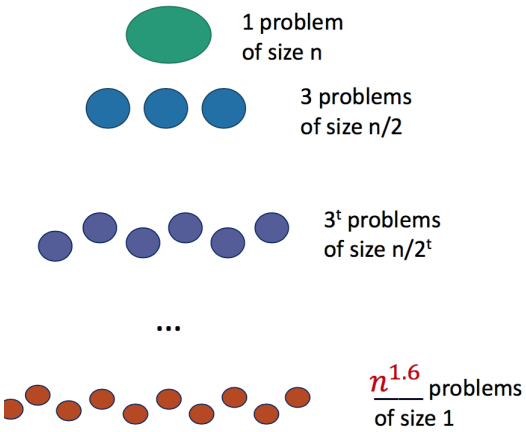
$$bc + ad = (a+b)(c+d) - ac - bd$$

What's the running time?



- If you cut n in half $\log_2(n)$ times, you get down to 1
- So we do this $\log_2(n)$ times and get $3^{\log_2(n)} \approx n^{1.6}$ problems of size 1

What's the running time?



- If you cut n in half $\log_2(n)$ times, you get down to 1
- So we do this $\log_2(n)$ times and get $3^{\log_2(n)} \approx n^{1.6}$ problems of size 1

We still aren't accounting for the work at the higher levels! We'll see later that this turns out to be okay.

In summary

Theorem. There exist a constant c such that for any two n-digit numbers x, y, we can compute $x \cdot y$ in at most $c \cdot n^{\log_2(3)} \approx c \cdot n^{1.6}$ one-digit operations.

More carefully...

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3 \cdot \left(3 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3 \cdot \left(3 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 3^2 \cdot T\left(\frac{n}{2^2}\right) + \left(\frac{3}{2} + 1\right) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3 \cdot \left(3 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 3^2 \cdot T\left(\frac{n}{2^2}\right) + \left(\frac{3}{2} + 1\right) \cdot cn \\
&= 3^2 \cdot \left(3 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{2^2}\right)\right) + \left(\frac{3}{2} + 1\right) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3 \cdot \left(3 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 3^2 \cdot T\left(\frac{n}{2^2}\right) + \left(\frac{3}{2} + 1\right) \cdot cn \\
&= 3^2 \cdot \left(3 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{2^2}\right)\right) + \left(\frac{3}{2} + 1\right) \cdot cn \\
&= 3^3 \cdot T\left(\frac{n}{2^3}\right) + \left(\left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn
\end{aligned}
$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3 \cdot \left(3 \cdot T\left(\frac{n}{2^2}\right) + c\left(\frac{n}{2}\right)\right) + cn \\
&= 3^2 \cdot T\left(\frac{n}{2^2}\right) + \left(\frac{3}{2} + 1\right) \cdot cn \\
&= 3^2 \cdot \left(3 \cdot T\left(\frac{n}{2^3}\right) + c\left(\frac{n}{2^2}\right)\right) + \left(\frac{3}{2} + 1\right) \cdot cn \\
&= 3^3 \cdot T\left(\frac{n}{2^3}\right) + \left(\left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn \\
&\vdots \\
&= 3^k \cdot T\left(\frac{n}{2^k}\right) + \left(\left(\frac{3}{2}\right)^{k-1} + \cdots + \left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn
\end{aligned}
$$

More carefully...

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$$
$$= 3^k \cdot T\left(\frac{n}{2^k}\right) + \left(\left(\frac{3}{2}\right)^{k-1} + \cdots + \left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn$$

Assume $n = 2^k$

$$= 3^k \cdot T(1) + 2 \cdot \left(\left(\frac{3}{2}\right)^k - 1\right) \cdot cn$$

More carefully...

$$
\begin{aligned}
T(n) &= 3 \cdot T\left(\frac{n}{2}\right) + cn \\
&= 3^k \cdot T\left(\frac{n}{2^k}\right) + \left(\left(\frac{3}{2}\right)^{k-1} + \cdots + \left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn
\end{aligned}
$$

Assume $n = 2^k$

$$
\begin{aligned}
&= 3^k \cdot T(1) + 2 \cdot \left(\left(\frac{3}{2}\right)^k - 1\right) \cdot cn \\
&= 3^k \cdot T(1) + 2 \cdot c \cdot 3^k - 2cn
\end{aligned}
$$

More carefully...

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$$
$$= 3^k \cdot T\left(\frac{n}{2^k}\right) + \left(\left(\frac{3}{2}\right)^{k-1} + \cdots + \left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn$$

Assume $n = 2^k$

$$= 3^k \cdot T(1) + 2 \cdot \left(\left(\frac{3}{2}\right)^k - 1\right) \cdot cn$$
$$= 3^k \cdot T(1) + 2 \cdot c \cdot 3^k - 2cn$$

$k = \log_2 n = \frac{\log n}{\log 2} \quad \Rightarrow \quad 3^k = 3^{\frac{\log n}{\log 2}} = n^{\frac{\log 3}{\log 2}} \leq n^{1.58\ldots} \leq n^{1.6}$

More carefully...

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + cn$$
$$= 3^k \cdot T\left(\frac{n}{2^k}\right) + \left(\left(\frac{3}{2}\right)^{k-1} + \cdots + \left(\frac{3}{2}\right)^2 + \frac{3}{2} + 1\right) \cdot cn$$

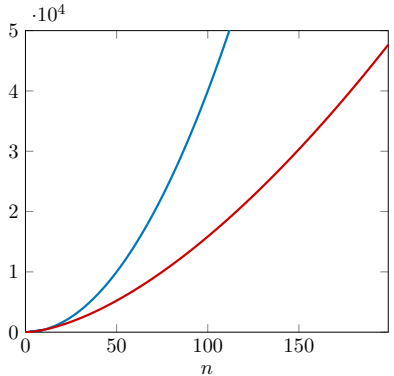
Assume $n = 2^k$

$$= 3^k \cdot T(1) + 2 \cdot \left(\left(\frac{3}{2}\right)^k - 1\right) \cdot cn$$
$$= 3^k \cdot T(1) + 2 \cdot c \cdot 3^k - 2cn$$

$k = \log_2 n = \frac{\log n}{\log 2} \quad \Rightarrow \quad 3^k = 3^{\frac{\log n}{\log 2}} = n^{\frac{\log 3}{\log 2}} \leq n^{1.58\ldots} \leq n^{1.6}$

$$\leq n^{1.6} \cdot (T(1) + 2c) \leq c' n^{1.6}$$

This is much better!



$f_1(n) = 4 \cdot n^2$ $\qquad$ $f_2(n) = 10 \cdot n^{1.6}$

What happened further?

- Toom and Cook (1963)
  - Scales like $n^{1.465}$

What happened further?

- Toom and Cook (1963)
  - Scales like $n^{1.465}$
- Schönhage and Strassen (1971)
  - Scales like $n \log(n) \log\log(n)$

What happened further?

- Toom and Cook (1963)
  - Scales like $n^{1.465}$
- Schönhage and Strassen (1971)
  - Scales like $n \log(n) \log\log(n)$
- Fürer (2007)
  - Scales like $n \log(n) C^{\log^*(n)}$
  - Harvey and Van der Hoeven (2017): $C = 4$

What happened further?

- Toom and Cook (1963)
  - Scales like $n^{1.465}$
- Schönhage and Strassen (1971)
  - Scales like $n \log(n) \log\log(n)$
- Fürer (2007)
  - Scales like $n \log(n) C^{\log^*(n)}$
  - Harvey and Van der Hoeven (2017): $C = 4$
- Harvey and Van der Hoeven (2019)
  - Scales like $n \log(n)$

Summary

- Saw first example of how to think algorithmically
- First design paradigm: divide and conquer

Summary

- Saw first example of how to think algorithmically
- First design paradigm: divide and conquer
- We were a bit sloppy!

Summary

- Saw first example of how to think algorithmically
- First design paradigm: divide and conquer
- We were a bit sloppy!
- But: we were right!
- And: we will develop tools to make all this rigorous!

Homework

- Check out course's Moodle page
- Go through self-study tutorials
- Do the problem set

Homework

- Check out course's Moodle page
- Go through self-study tutorials
- Do the problem set

Thanks! See you next week!