Algorithms is "Cool"

National University of Singapore
School of Computing

- Where do we use Algorithms
- Analysis of Algorithms & Why it is important
- Sample Computational Problems

Why Algorithm is Cool:

Algorithms is Anywhere & Everywhere.

Hon Wai Leong, NUS

Motivation (Why algorithms?)

- **Where do we use Algorithms?**
  - **Computer Science, Engineering**
  - **Business, Operations Research**
  - **Finance, Social Sciences, … , Everywhere**
- **Why is Performance Important?**
  - **Exponential-size solution space**
  - **NP-completeness**
- **Problem Size Explosion**
  - **Computer Chip Complexity,**
  - **Database Sizes,**
  - **Human Genome Project, WWW (Google),**

Diverse Applications (where?)

- **Design the next generation CPU/GPU chip**
  - ***how to design optimally (w.r.t. speed, power, area)***
- **Internet (WWW)**
  - ***how to manage, manipulate large volume of data***
- **e-Commerce**
  - ***how to manage transaction (secure, private)***
- **Logistics**
  - ***how to manage transport/transfer of goods, people***
- **Human Genome Project**
  - ***how to analyze huge volume of DNA/protein data***

Analysis of algorithms

The theoretical study of program performance and resource usage.

What's are the important aspects of software?

- modularity
- correctness
- maintainability
- functionality
- robustness
- user-friendliness
- programmer time
- simplicity
- extensibility
- reliability

* *Speed / performance*

Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is ***feasible*** and what is ***impossible***.
- Performance is the ***currency*** of computing.
- The lessons of program performance ***generalize*** to other computing resources.
- Speed is ***fun***!

Some Combinatorial Problems

Combinatorial Problem	*Computational Complexity*
❖ Maintaining student records	*Easy*
❖ Data Compression	*Easy*
❖ Traveling Salesman Problem	*Hard*
❖ Shortest Route Planning	*Easy*
❖ Program Halting Problem	*Impossible*
❖ VLSI Chip Layout Problem	*Hard*
❖ Examination Time Table Scheduling	*Hard*
❖ Checking if graph is acyclic	*Easy*
❖ Computer Deadlock Problem	*Easy*
❖ Sorting records in a Database	*Easy*
❖ Finding patterns in a text	*Easy*

Combinatorial Problems…

- **General Combinatorial Problem**
  - Given a finite, discrete set S of objects
  - ***To compute some function*** $f(S)$

- **Algorithmic Issues…**
  - Representation of the set S
  - Efficient manipulation of the set S
  - Efficient algorithm to compute $f(S)$

Design and Analysis of Algorithms

- **Given a problem *P*,**
  - **Can it be solved?**
- **If "Yes", given an algorithm *A* for solving *P*,**
  - **Is algorithm *A* correct ?**
  - **How good is algorithm *A* ?**
  - **Can find a better algorithm *A'* ?**
- **How do we define good?**
  - **How much time it takes.**
  - **How much space it uses.**

Computability

Verification

Efficiency

Time Complexity

Space Complexity

Hon Wai Leong, NUS

Complexity

- Given an algorithm *A* for problem *P*,
- How to do better?
  - Is the time complexity of *A* polynomial ?
  - Can we design faster algorithm A' ?
  - Can we design algorithm *A''* that uses less space

Complexity of Algorithm

- Can we do better?
  - Is the problem NP-complete?
  - Can we establish lower bounds
  - Is the algorithm "*best possible*"

Complexity of Problem

Why study program performance?

But, is it useful at all?

Q: Can we trust the CPUs in our laptop / iPad / Apple Watch?

Is the MULT(*) operation correct?

Your laptop / iPad / Apple Watch
all have a CPU inside.

The CPU has a MULT operation (*)

1994, Pentium FDIV bug

Pentium FDIV bug

From Wikipedia, the free encyclopedia

The **Pentium FDIV bug** was a computer bug that affected the floating point unit (FPU) of the early Intel Pentium processors. Because of the bug, the processor could return incorrect binary floating point results when dividing a number. Discovered in 1994 by Professor Thomas R. Nicely at Lynchburg College,[1] Intel attributed the error to missing entries in the lookup table used by the floating-point division circuitry.[2]

66 MHz Intel Pentium (sSpec=SX837) with the FDIV bug

The severity of the FDIV bug is debated. Intel, producer of the affected chip, claims that the common user would experience it once every 27,000 years while IBM, manufacturer of a chip competing with Intel's Pentium, claims that the common user would experience it once every 24 days. Though rarely encountered by most users (*Byte* magazine estimated that 1 in 9 billion floating point divides with random parameters would produce inaccurate results),[3] both the flaw and Intel's initial handling of the matter were heavily criticized by the tech community. The man who

https://en.wikipedia.org/wiki/Pentium_FDIV_bug

What if we don’t true the CPU?

How can we check it…

Q: Can we trust the CPUs in our laptop / iPad / Apple Watch?

Testing the * operation in a CPU

Q: How to test that the "*" operation of your CPU is correct?

A: Check exhaustively. For all a, b
Check $a * b = c$

Q: How long will it take?

A: Any guesses?

Testing the * operation in a CPU

…very fast.
Laptop ~3G-Flop

Q: How long will it take?

Assume we use a 100G-Flop CPU
can do 100B operations per sec.

a is a 32-bit number (2^{32} cases)
b is a 32-bit number (2^{32} cases)
So, $(a * b)$ there are (2^{64} cases)

Time taken = $(2^{64} / 100 \times 10^{9})$ sec

WolframAlpha computational knowledge engine

2^64 / 100x10^9 seconds

Examples Random

Assuming seconds of time for "seconds" | Use seconds of arc instead

http://www.wolframalpha.com/input/?i=2^64+%2F+100x10^9+seconds ←- URL

Input interpretation:

$$\frac{2^{64}}{100 \times 10^9} \text{ seconds}$$

Unit conversions: More

3.074×10^6 minutes

51 241 hours

2135 days

305 weeks

70.19 months

Testing * operation in a CPU

Q: How long will it take?

Assume we use a 100G-Flop machine can take 100B operations per sec.

a is a 32-bit number (2^{32} cases)
b is a 32-bit number (2^{32} cases)
So, ($a * b$) there are (2^{64} cases)

Time taken = ($2^{64} / 100 \times 10^9$) sec

≈ 6 years!

Summary

Testing * operation in a CPU

Q: If we use $O(n^2)$ algorithm?

≈ 6 years!

Impossible.
Not practical

Testing * operation in a CPU

Q: What if we have (n lg n) algorithm?

Assume we use a 100G-Flop machine
can do 100B operations per sec.

a is a 32-bit number (2^{32} cases)
b is a 32-bit number (2^{32} cases)

When $n = 2^{32}$, with (n lg n) algorithm

Time taken = ($2^{32} * 32 / 100 \times 10^9$) sec

< 2 sec!

Hon Wai Leong, NUS

Summary

Testing * operation in a CPU

Q: If we use $O(n^2)$ algorithm?

≈ 6 years!

Impossible.
Not practical

Q: If we use $O(n \lg n)$ algorithm?

< 2 sec!

Fast & Practical

Moral of the story

Analysis of algorithms
help us make *predictions*.

Analysis of algorithms
help us *prepare for the worst case*.

Application in Web-Service

Suppose you code up a new web-service – *CoolApp*
- you debugged your code, and after some time.
- you got it *working*, you *tested it* a little bit
- it is quite fast

Can you release *CoolApp*?
Will it work? Or will it bomb?

If dream come true & *CoolApp*' s wildly popular?
How fast is "quite fast", will server die? When?

Moral of the story

Analysis of algorithms
help us make *predictions*.

Analysis of algorithms
help us *prepare for the worst case*.

Note: If operation is "quite fast", 0.02sec/op
that's 3min for 10,000 clicks per second,
that's 12min for 40,000 clicks per second,

Also, how big a load can a server take before dying?

40,000 clicks per seconds (July 2015)

Google

number of google search per second

Web Images News Videos More Search tools

About 257,000,000 results (0.49 seconds)

Google now processes over **40,000** search queries every second on average (visualize them here), which translates to over **3.5 billion** searches per day and **1.2 trillion** searches per year worldwide. The chart below shows the number of searches per year throughout Google's history:

Google Search Statistics - Internet Live Stats
www.internetlivestats.com/**google-search**-statistics/

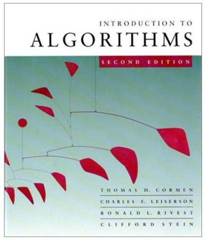
Feedback

Story of Algorithms in Action

Credit card processing centre in SG (Sci Park):
- monitor showing servers load for diff. countries,
  - blue, green, yellow, orange (send SMS alert),
- RED (URGENT Alert! ➔ deploy more servers!)

Note: ***Picture is NOT the real thing.***
But it gives the rough idea
and "demos" my point.

Note to Self:
Picture is NOT very good.
Will find a better one.



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!

Thank you.

Q & A