

# *Algorithms* is “Cool”

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

*Why Algorithm is Cool:*

*Algorithms is Anywhere & Everywhere.*

# Where do we need algorithms?

---

- ❖ 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*

# 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),

# Analysis of algorithms

---

*The theoretical study of program performance and resource usage.*

How long does the program run?

\* *Speed / performance*

How much resources does the program use?

\* *Memory/disk space, custom devices,*

# Other aspects of software

---

What are the important aspects of software?

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

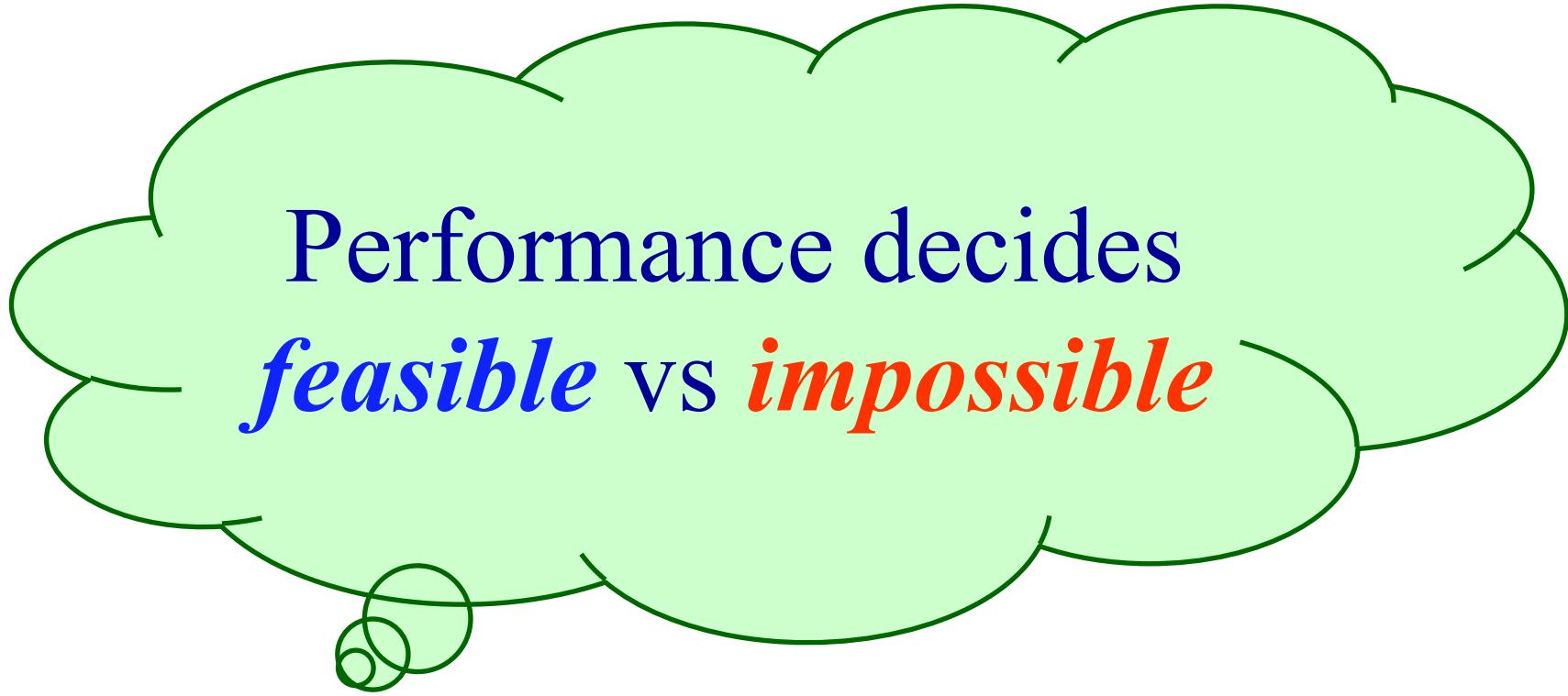
*But often, speed / performance is  
the crucial factor.*

# 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*!

# Why study program performance?



Performance decides  
***feasible*** vs ***impossible***

**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,<sup>[1]</sup> Intel attributed the error to missing entries in the lookup table used by the floating-point division circuitry.<sup>[2]</sup>

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),<sup>[3]</sup> 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](https://en.wikipedia.org/wiki/Pentium_FDIV_bug)

# What if we don't trust the CPU?

---



# 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

**Q:** How long will it take?

This is very fast.  
(most laptops ~3G-Flop)



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

2<sup>64</sup> / 100x10<sup>9</sup> seconds

≡ Examples    ⚡ Random

Assuming seconds of time for "seconds" | Use [seconds of arc](#) instead

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

Input interpretation:

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

Unit conversions:

[More](#) $3.074 \times 10^6$  minutes

51241 hours

2135 days

305 weeks

70.19 months

(Algorithms is Cool) Page 13

# 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?

$\approx 6$  years!



*Impossible.  
Not practical*



# Why study program performance?

The task is seemingly  
*impossible*

**Q:** What can we do?

**A:** Of course,  
better algorithm.

# 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<sup>32</sup> cases)
- $b$  is a 32-bit number (2<sup>32</sup> cases)

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

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

< 2 sec!



# Good algorithm makes BIG difference

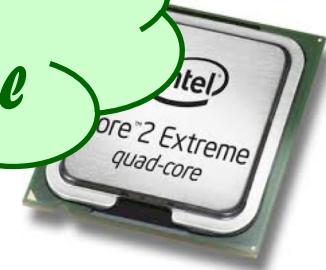
## Testing \* operation in a CPU

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

$\approx 6$  years!



*Impossible.  
Not practical*



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

< 2 sec!

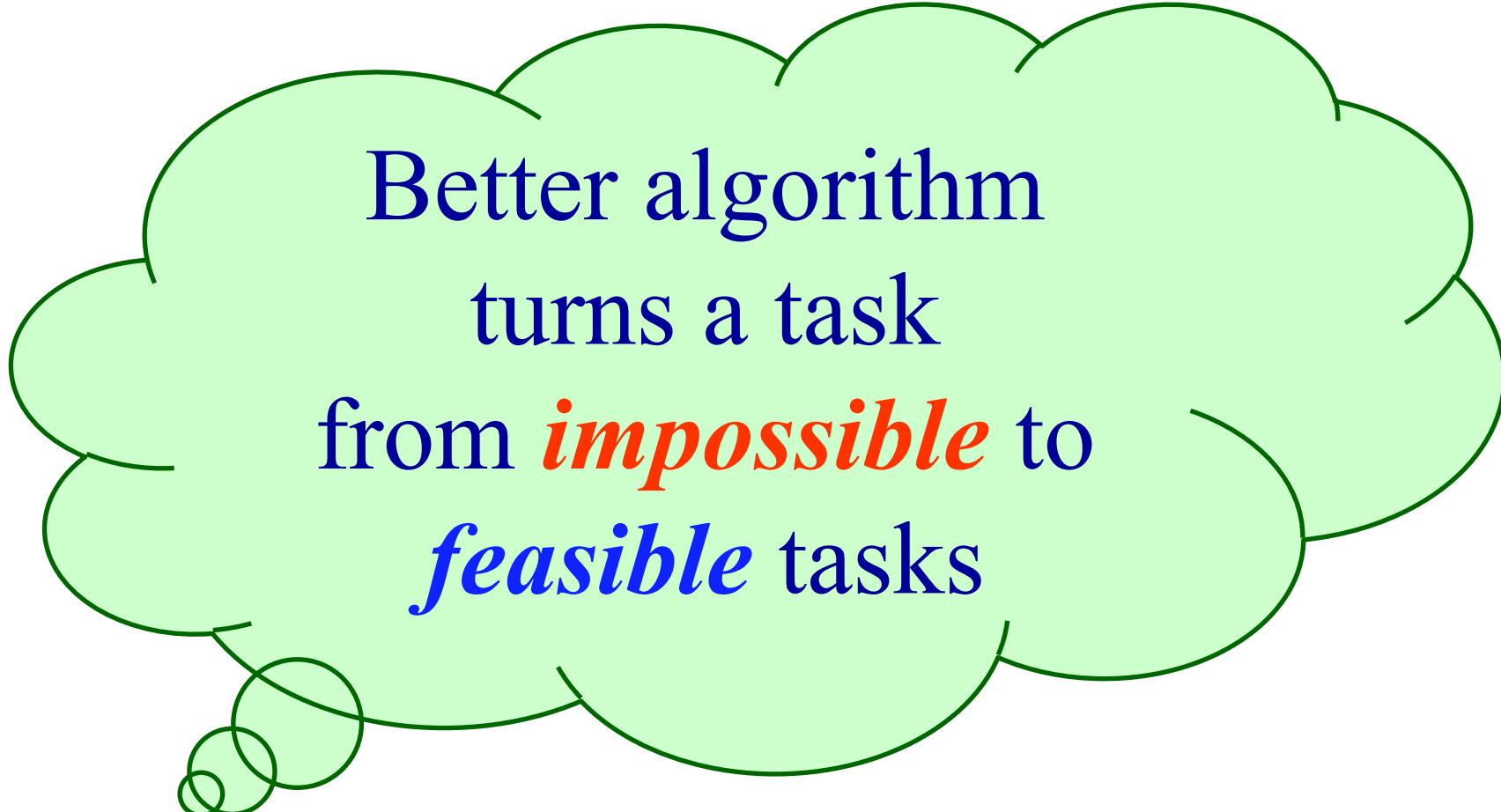


*Fast & Practical*



# Conclusion 1:

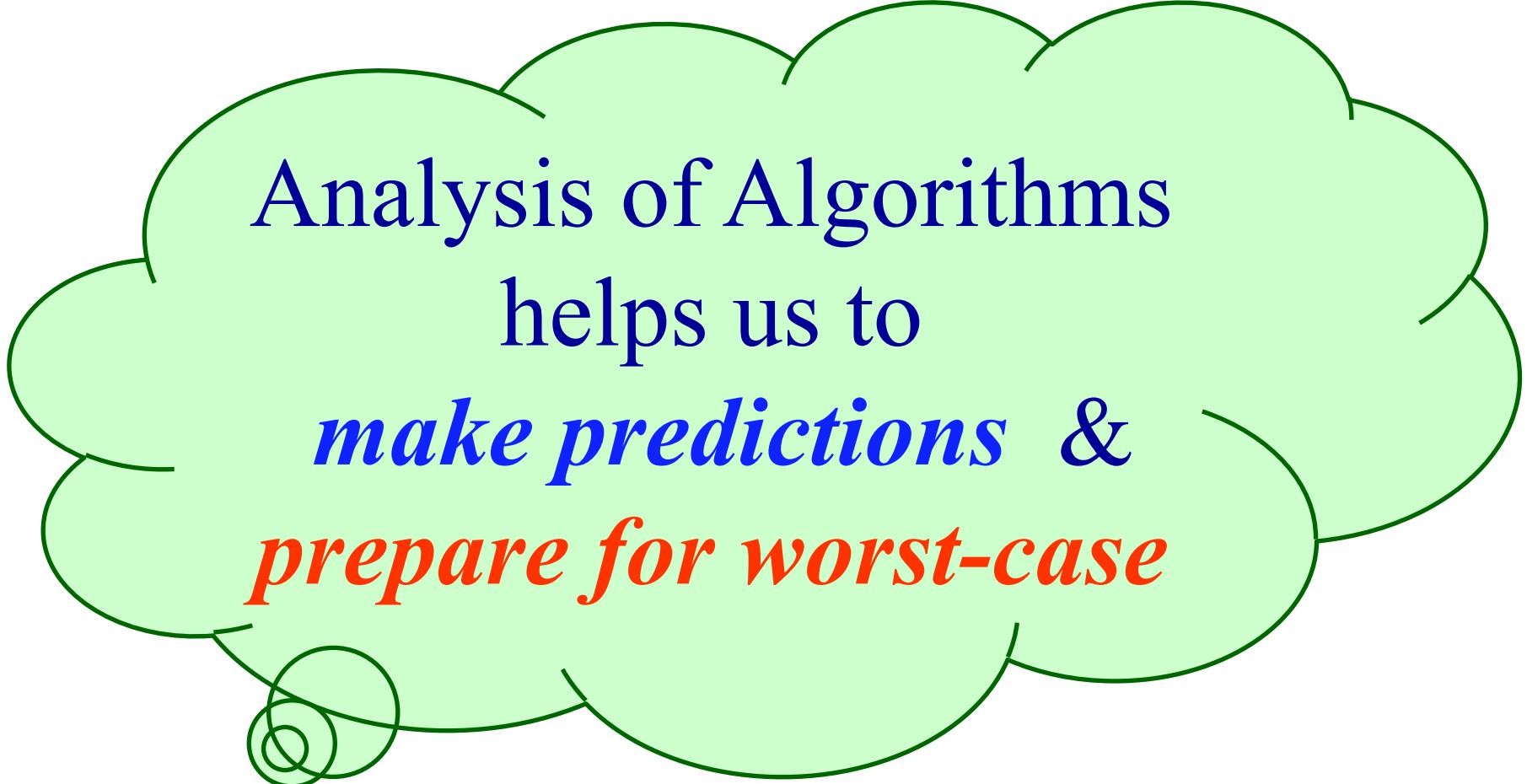
---



Better algorithm  
turns a task  
from *impossible* to  
*feasible* tasks

# Another Story...

---



Analysis of Algorithms  
helps us to  
*make predictions &*  
*prepare for 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?

# Making predictions, for worst-case

Your app is *quite fast*...

One operation takes 0.02sec (“*quite fast*”)

But is it fast *enough*?

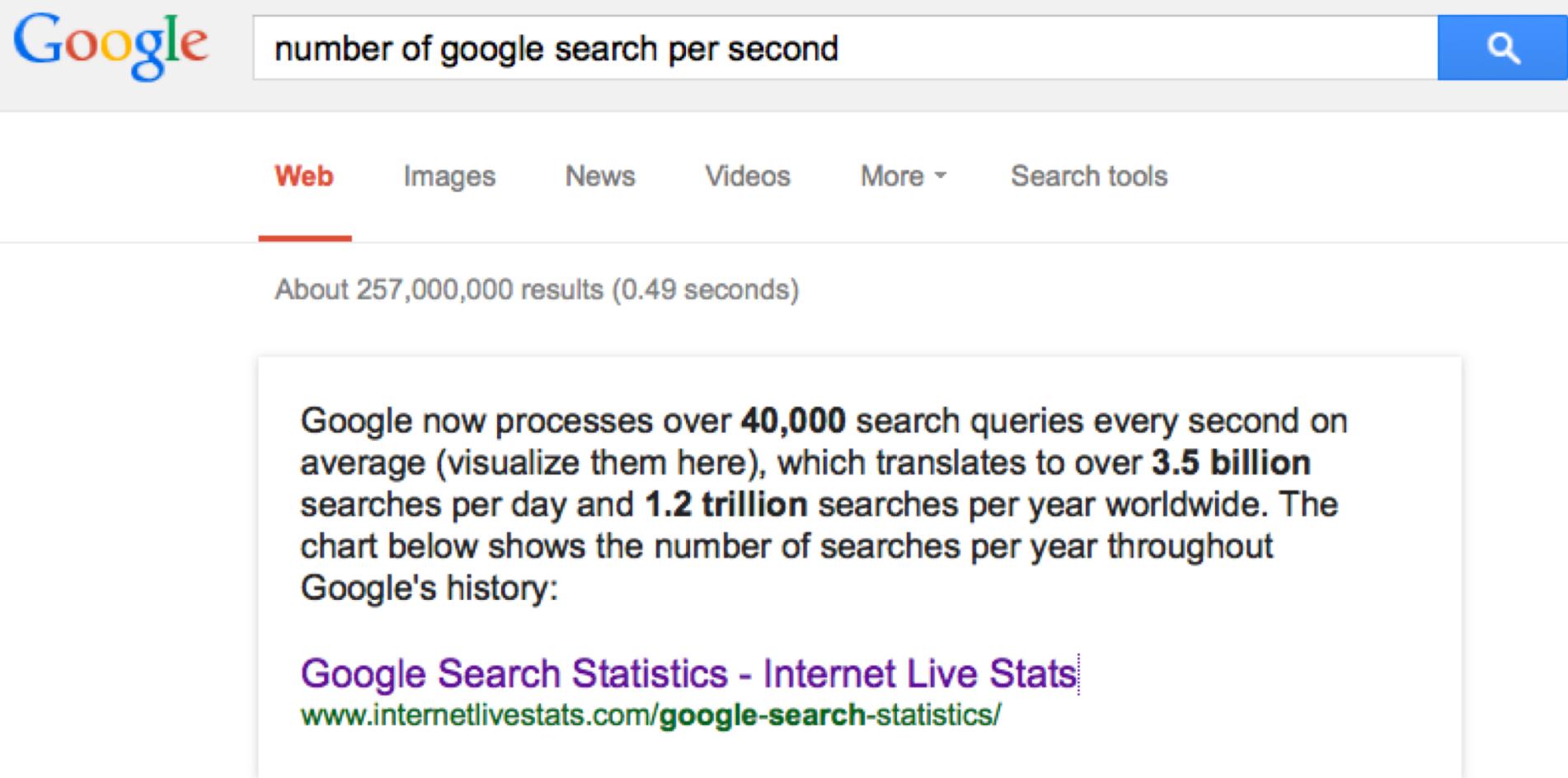
10,000 clicks/sec will take 3min

40,000 clicks/sec will take 12min

Too long to wait.  
App is *NOT Cool*.

**Q:** How big a load, before server dies?

# 40,000 clicks per seconds (July 2015)



The image shows a screenshot of a Google search results page. At the top left is the Google logo. To its right is a search bar containing the query "number of google search per second". On the far right of the search bar is a blue search button with a white magnifying glass icon. Below the search bar is a navigation menu with options: Web (which is underlined in red), Images, News, Videos, More, and Search tools. A horizontal line separates this menu from the search results. Below the line, the text "About 257,000,000 results (0.49 seconds)" is displayed. The main content area contains a text box with the following text:

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/](http://www.internetlivestats.com/google-search-statistics/)

[Feedback](#)

(Algorithms is Cool) Page 23

# Story of Algorithms in Action

Credit card processing centre in SG (Sci Park):

- monitors 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.

# This is simple case (linear)

---

- We had assume algorithm is linear
  - ❖ Running time is  $O(n)$ ,  $n$  is size of problem
  
- Why if algorithm is quadratic:  $O(n^2)$ ?
  - ❖ OK, when  $n = 10^5$ ;
  - ❖ Slow when  $n$  is in the millions
  
- Why if algorithm is exponential:  $O(2^n)$ ?
  - ❖ Impossibly slow, even small  $n$ , like  $n = 100$ .

# 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!*

# How to make Algorithms relevant, interesting, FUN?

- Learn and *really* understand algorithms
- Reflect and relate lessons learnt
  - ❖ To the bigger picture of your **major/job**
  - ❖ To **software development / app development**
  - ❖ To things in your everyday life
- How much does a Google Search cost?

---

**Thank you.**

**Q & A**



**School of Computing**