CSC3160: Design and Analysis of Algorithms

Week 1: Introduction

Instructor: Shengyu Zhang

First week

- Part I: About the course
- Part II: About algorithms
  - What are algorithms?
  - Why are they important to study?
- Part III: About complexity
  - What is the complexity of an algorithm / a problem
  - Growth of functions

Part I: About the course

Tutorials

- Tutorials: Let's set the time now.
  - Tuesday: 5:30pm-6:15pm

- Webpage: http://www.cse.cuhk.edu.hk/~syzhang/course/Alg15
  - Information (time and venue, TA, textbook, etc.)
  - Lecture slides
  - Tutorials
  - Homework
  - Announcements

About the flavor of the course

- It's more of a math flavor than a programming one.
- You will need to write pseudo-code, but never C/Java/…
- You will design and analyze, think and prove (rather than code)

Prerequisites

- Officially:
  - CSC2110 DISCRETE MATHEMATICS
  - CSC2100/ESTR2102 DATA STRUCTURES
- Effectively: Basic mathematical maturity
  - functions, polynomial, exponential;
  - proof by induction;
  - basic data structure operations (stack, queue, …);
  - basic math manipulations…
- Note: As long as you'd like to learn it.

Homework, Exam

- Homework assignments (20%).
  - About 4 assignments.
- Mid-term exam (30%).
- Final (50%).

Homework Policy

- Discussions and googling on web are allowed in general
- But you have to write down the solution yourself
- And you fully understand what you write.

Zero tolerance for cheating/plagiarism

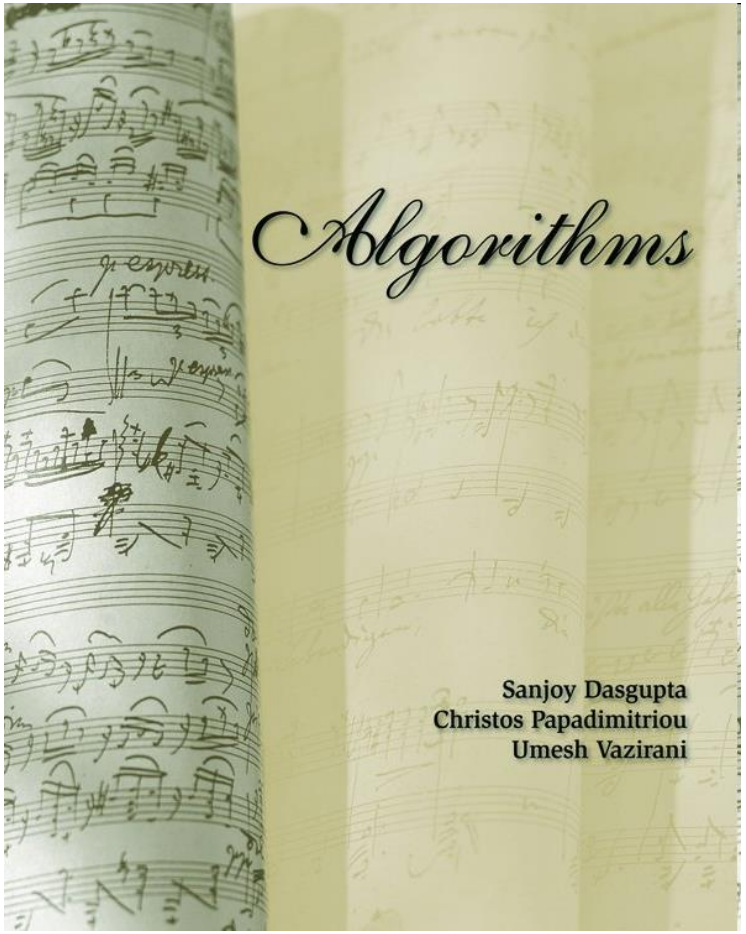
- Worse than a 0 score for this course; you may be out of dept/school/university.
- *"The Chinese University of Hong Kong places very high importance on honesty in academic work submitted by students, and adopts a policy of zero tolerance on cheating and plagiarism. Any related offence will lead to disciplinary action including termination of studies at the University."*

---- Honesty in Academic Work

(http://www.cuhk.edu.hk/policy/academichonesty)

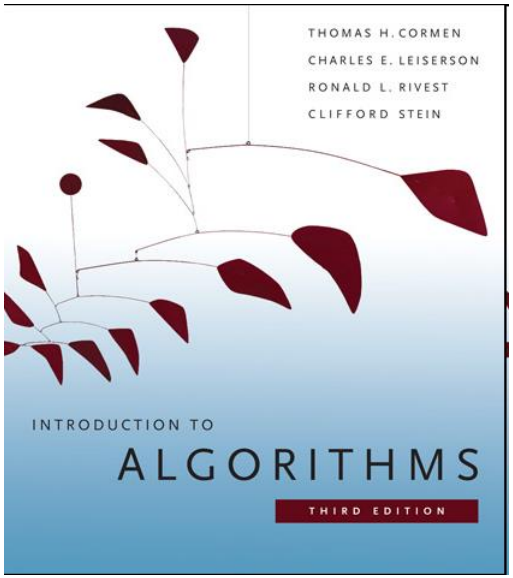
textbook

- *Algorithms*
  S. Dasgupta,
  C.H. Papadimitriou,
  U.V. Vazirani,
  McGraw-Hill Higher Education, 2007.
- Draft available at http://www.cs.berkeley.edu/~vazirani/algorithms.html

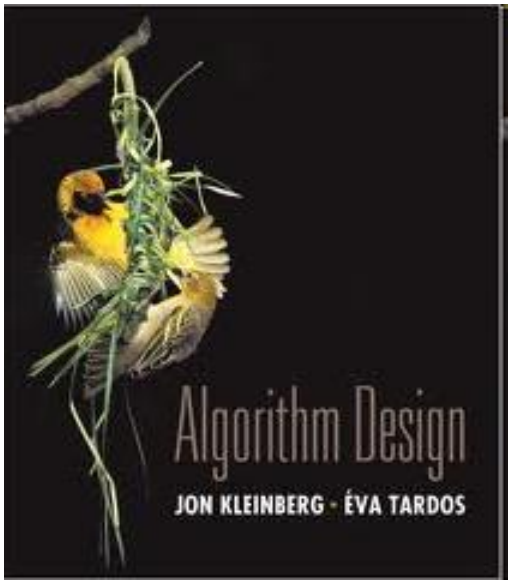


Two good references

- The following two books are very good references, containing many materials that we don't have time to cover.



Introduction to Algorithms, 3rd ed, T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *MIT Press*, 2009.



Algorithm Design, J. Kleinberg and E. Tardos, *Addison Wesley*, 2005.

Expectations

- Student/Faculty Expectations on Teaching and Learning: http://www.erg.cuhk.edu.hk/erg-intra/content.php?s=&sub_id=5&content_id=34

- Also on the course webpage

Suggestions/expectations/requirements

- Before class: no need to prepare!
- In class:
  - Try to come on time.
  - Try your best to get more involved in the class.
  - Please don't chitchat.
    - It affects you, me, and other students.
- After class: treat homework seriously
  - The exams will be related to homework.

Awards for interactions

- Interactions in class are highly welcome.
  - Don’t be shy or modest
  - Don’t be afraid of asking “stupid” questions
    - There’re no stupid questions; there’re only stupid answers.
  - Don’t be afraid of making mistakes
    - If you have to make some mistake, the earlier the better.
- I’ll give points for answering questions in class.
  - Points directly go to your final grade.

About comments

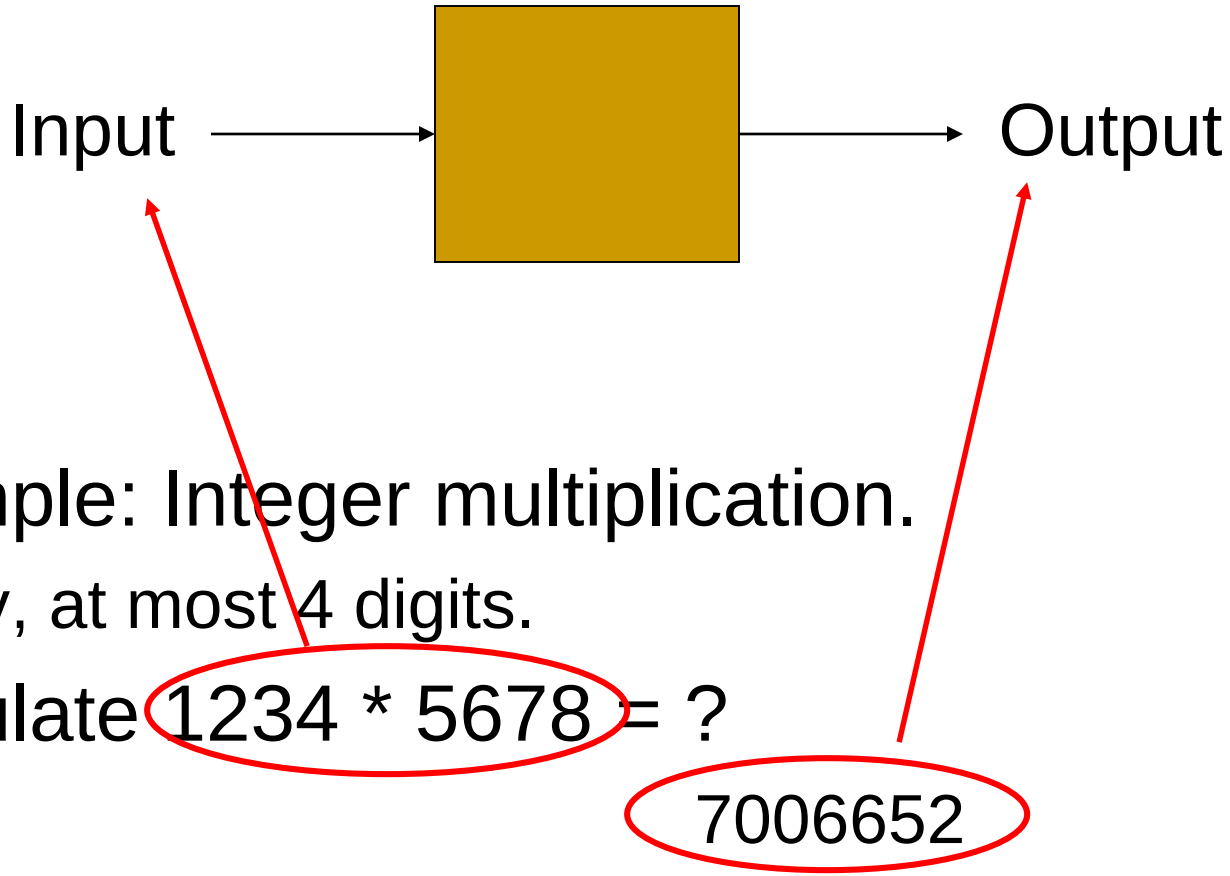
- Comments are welcome.
  - Email: syzhang@cse.cuhk.edu.hk
- But please be responsible and considerate.
  - Pace: Please understand that I cannot proceed with the majority still confused.
- Any questions about the course?
- My questions:
  - What are your goals?
  - What do you like to learn from this course?
  - What excite you the most in general?

Part II: About algorithms

Roadmap of Part II.

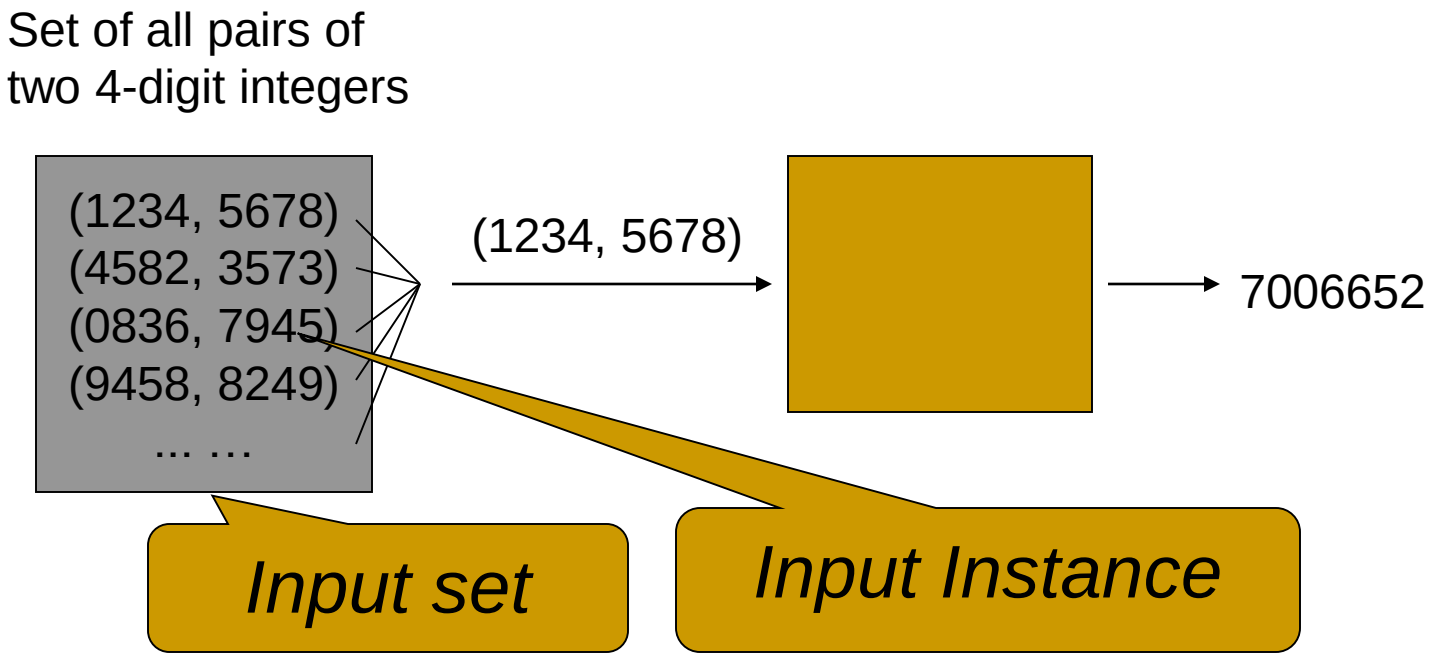
- What is computation.
- What is an algorithm.
- Algorithmic issues are everywhere.

Computation

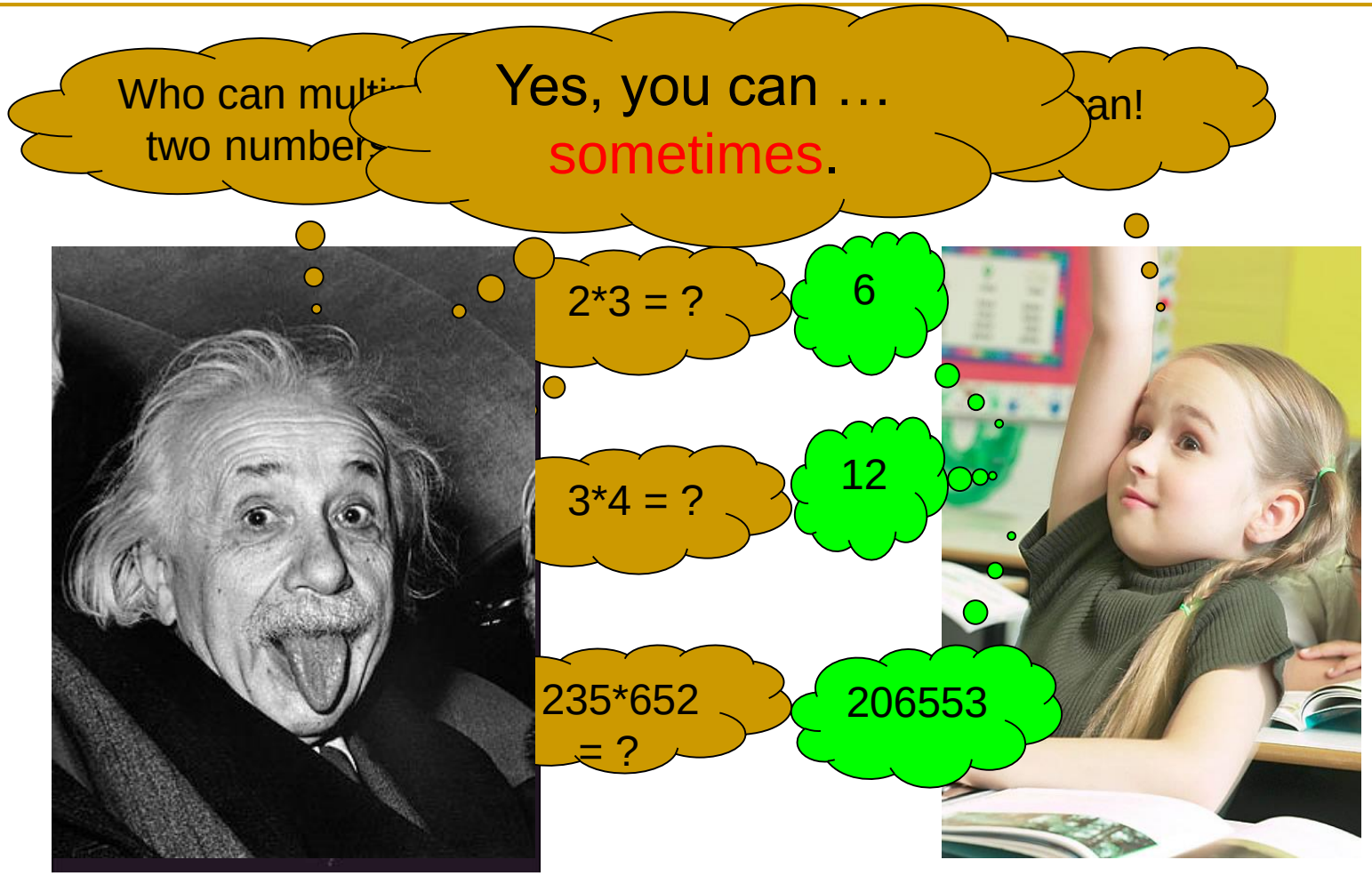


- Example: Integer multiplication.
  - Say, at most 4 digits.
- Calculate 1234 * 5678 = ?

7006652

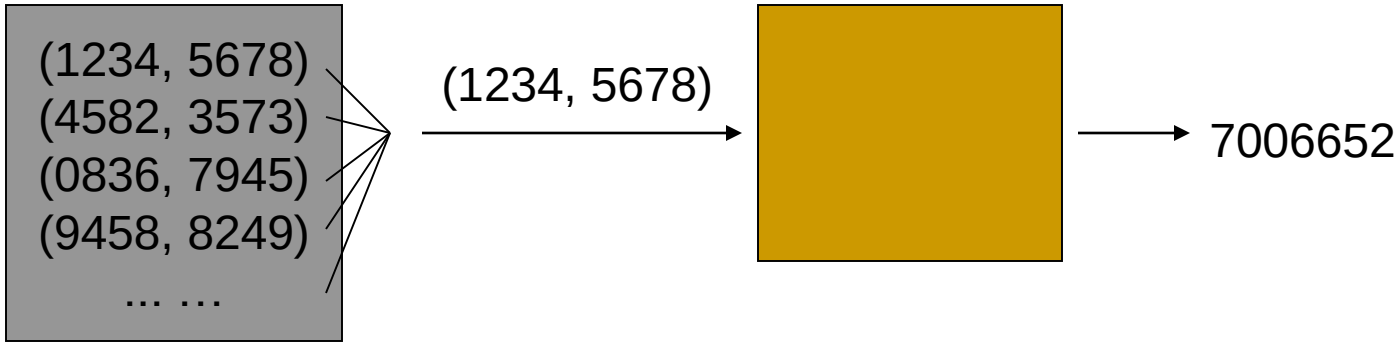


- We have a set of integer pairs --- input set
  - Each pair: (input) instance
- What do we require for computation?



- We want the computation to be correct for *all* (4-digit) integer pairs.

Computation



- So the computation in this case is:

  a box s.t. for each pair of integers fed in as input, the box can give the output correctly.
- Ok, we've known what we want, …what's the next question?

Always this question in our lives…

- On many occasions we know the goal clearly
  - I want to get an A.
  - I want to be a billionaire.
  - I want to be cool.

The question is how to achieve it.

What do we want for the implementation of the computation box?

- It's good to have a step-by-step instructions…

  … with each step being some elementary one that can is easy to implement.
  - Algorithm!
- The faster we can finish all the steps, the better.
  - Complexity!

A good example: driving directions

- Suppose we want to drive from CUHK to Central. How to route?
- Let's ask Google.
- What's good here:
  - Step by step.
  - Each step is either turn left/right, or go straight for … meters.
  - An estimated time is also given.

Algorithmic issues are everywhere…

Example 1: driving directions.

- Input: a pair of addresses (A, B)
- Output: a route from A to B

- Algorithmic question: how do they find the best route?

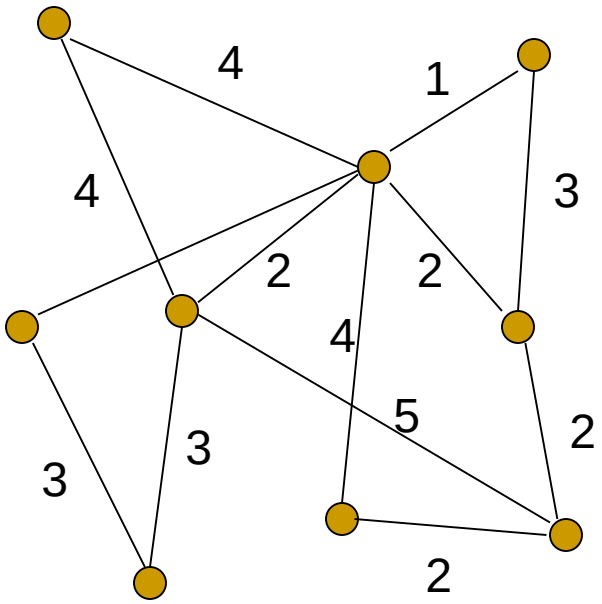
Example 2: Google ranking

- Input: a query string
- Output: an ordered list of related webpages

- Algorithmic question: how do they rank the resulting pages?

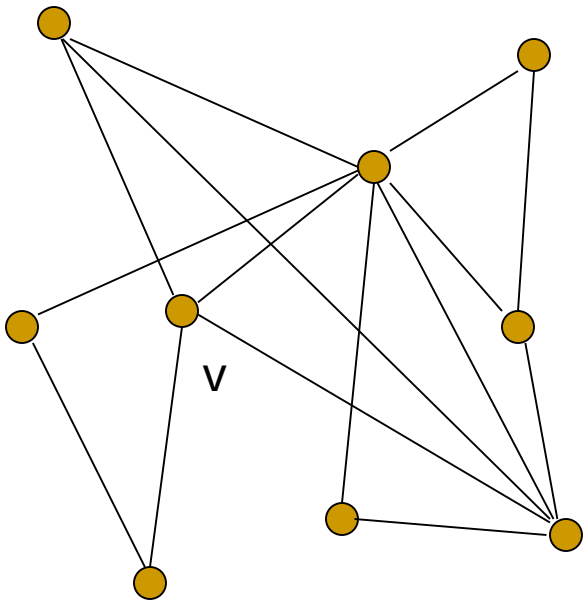
Example 3: Minimum connection

- Suppose we have n computers, connected by wires as given in the graph.
- Each wire has a renting cost.
- We want to select some wires, such that all computers are connected (i.e. every two can communicate).
- Algorithmic question: How to select a subset of wires with the minimum renting cost?



Example 4: Vertex traversal

- You plan to go to a couple of cities in US, with available airline flights as illustrated.
- Starting at some city v, you want to visit all other cities once and then come back to v.
- Algorithmic question: Is there such a route?



Btw, what if we change the question to “visit each edge exactly once”?

Example 5: Existence of irrational numbers

- Who remember a proof?
- Proof 1: Compare the "sizes" of **Q** and **R**
  - Stronger fact:
    - Irrational numbers not only exi… …are almost everywhere.
- This proof is not good since you still don't know any particular irrational number (though you know a random number is irrational with probability 1 !).
  - We say such a proof non-constructive.

BTW: This is a recurring theme in TCS: On one hand, a *random* object has some property with high prob.; on the other hand, finding even one *explicit* object with that property is terribly hard.

Existence of irrational numbers

- Proof 2: Diagonalization
  - List all rational numbers as $r_1, r_2, \ldots$
    - Rational numbers are countable
  - Construct the following number: $0.s_1 s_2$ ... such that
    - s_i is different than the i-th digit of r_i
  - Then this number is not equal to any rational number, so it must be irrational.
- This proof gives an explicit number.
  - Though the algorithm runs forever.

Summary of Part II.

- An algorithm is a computational procedure that has *step-by-step* instructions.
  - as opposed to supernatural insight (顿悟) in eastern philosophy
  - as opposed to many existence proofs in math

A final remark on algorithm vs. insight

- [Note] I'm not saying that supernatural insight is not important.

- Actually though good algorithms are fast procedures that we can follow, …
- *finding/designing those good algorithms* is nothing easy!
  - It's more like an art than science.

Good news

- There are still some strategies helpful in general!
- This course: give you some approaches and ideas
  - that have successfully be used in past, and will be useful for your future research or non-research work

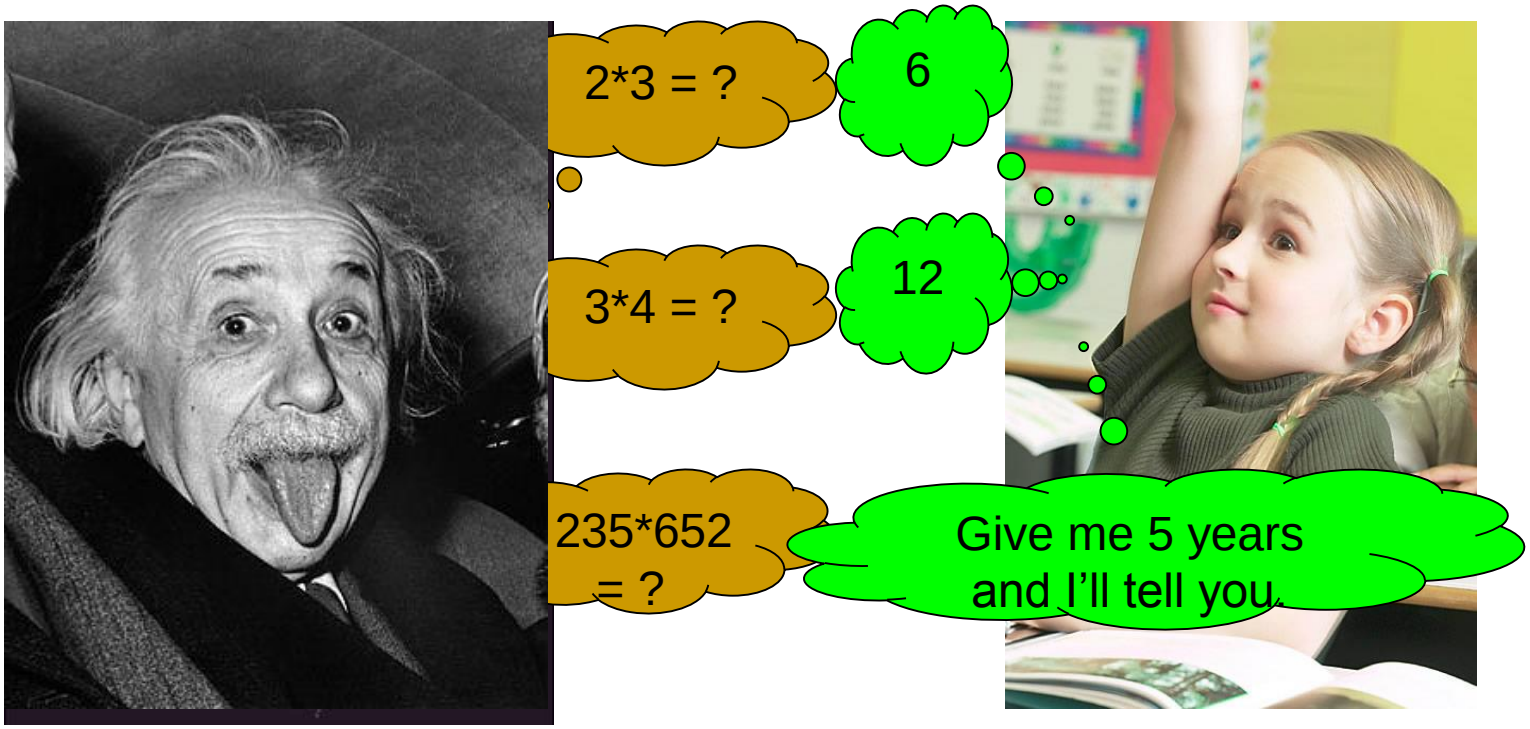
Questions about algorithms?

Part III: About complexity

More on complexity

- Why time matters?
  - Time is money!
  - Being late means 0 value
    - Weather forecast.
    - Homework.
- Running time: the number of elementary steps
  - Assuming that each step only costs a small (or fixed) amount of time.

What do we require for computation?



- In many cases, we'd like an algorithm fast on all input instances.

complexity

- The *worst-case time complexity* of an algorithm *A* is the running time of *A* on the *worst-case* input instance.
  - $C(A) = \max_{\text{instance } x}$ (running time of *A* on *x*)
- The *worst-case time complexity* of a computational problem *P* is the worst-case complexity of the *best* algorithm *A* that can solve the problem.
  - i.e. the best algorithm that can give answers on all instances.
  - $C(P) = \min_{A} \max_{x}$ (running time of *A* on *x*)

Hardness of problems can vary a lot

- Multiplication:
  - 1234 * 5678 = ?
    - 7006652
  - 2749274929483758 * 4827593028759302 = ?
    - Can you finish it in 10 minutes?
- Do you think you can handle multiplication easily?

Complexity of integer multiplication

- In general, for n-digit integers:
  - $x_1x_2 \dots x_n * y_1y_2 \dots y_n = ?$
- [Q] *How fast is our algorithm?*
- For each $y_i \quad (i = n, n-1, \dots, 1)$
  - we calculate $y_i * x_1x_2 \dots x_n$,
    - n single-digit multiplications
    - n single-digit additions

$$
\begin{array}{rl}
 & x_1x_2 \dots x_n \\
* & y_1y_2 \dots y_n \\
\hline
 & * \ * \ * \ \dots * \\
 & * \ * \ * \ \dots * \\
 & \dots \quad \dots \\
+ & * \ * \ * \ \dots * \\
\hline
 & * \ * \ * \ * \ * \ * \dots \dots * \\
\end{array}
$$

- We finally add the n results (with proper shifts)
  - $\leq 2n^2$ single-digit additions.
- Altogether: $\leq 4n^2$ elementary operations
  - single-digit additions/multiplications

- Multiplication is not very hard even by hand, isn’t it?

Now let's consider the inverse problem

- Factoring: 35 = ? * ?
- How about 437? I'll give you 1 minute.
- 8633? I'll give you 5 minutes.
- It's getting harder and harder,
  - Much harder even with one more digit added!

- If you're still confident of your factoring ability:

  135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879416377328533906109750544334999811150056977236890927563

- This I can give you 5 days.
  - If you tell me the answer, I'll pay tuition for your rest undergrad career.

Why I'm so confident?

- Let's first see the complexity of our algorithms.
- Alg 1: Try all $i < 10^n$
- Alg 2: Try all primes $p < 10^n$
- Alg 3: Try all $p \leq (10^n)^{1/2} = 10^{n/2}$
- Alg 4? …

- How large is $10^{n/2}$ for even a small n, like 200?
  - Larger than the estimated number of particles in the universe, which is somewhere between 10^{72} and 10^{87}

Are we dumb or what?

- Bad news: Probably yes, we are.
- Good news: All others are also dumb.
  - The best known: about $2^{O(n^{1/3})}$
- Good news 2:
  - If we look at the other side of the coin of hardness… it has a bright side!
- Since we are all dumb (so far), we can use this for cryptography!

- RSA [Rivest, Shamir, Adleman]:
  - widely-used today,
  - broken if one can factor quickly!

- One-line message: Quantum computers can factor quickly!

Another possibility

- Maybe it's not really because our stupidity
- Maybe no one can ever design a fast algorithm on the currently used computers for factoring!
- In other words, maybe the factoring problem is actually just very hard in nature!
  - i.e. the complexity of Factoring is huge
    - Recall: $C(P) = \min_A \max_x$ (running time of A on x)
- Many people buy this! Or even

  $\min_A$ average $_x$ (running time of A on x) is huge

What do we learn?

- Hardness of computational problems can vary a lot!
- i.e. the hardness can increase *a little* or *a lot* with the input size.
  - Multiplication: primary school pupils can do it.
  - Factoring: the smartest people in all history cannot

As a result,

- Implication 1: We care about the speed of the increase, especially when the size is very large.
- Many interesting instances in both theory and practice are of huge (and growing!) sizes.

- Implication 2: We care about the big picture first.
- Is the problem as easy as multiplication, or as hard as factoring?

- In this regard, we consider the so called *asymptotic* behavior,…
  - Eventually, i.e. for large n, is the function like n, or n^2, or 2^n?
- with constant factors ignored at first
  - i.e. we care about the difference between n^2 and 2^n much more than that between n^2 and $1000n^2$
  - Engineering reason: speedup of a constant factor (say of 10) is easily achieved in a couple of years

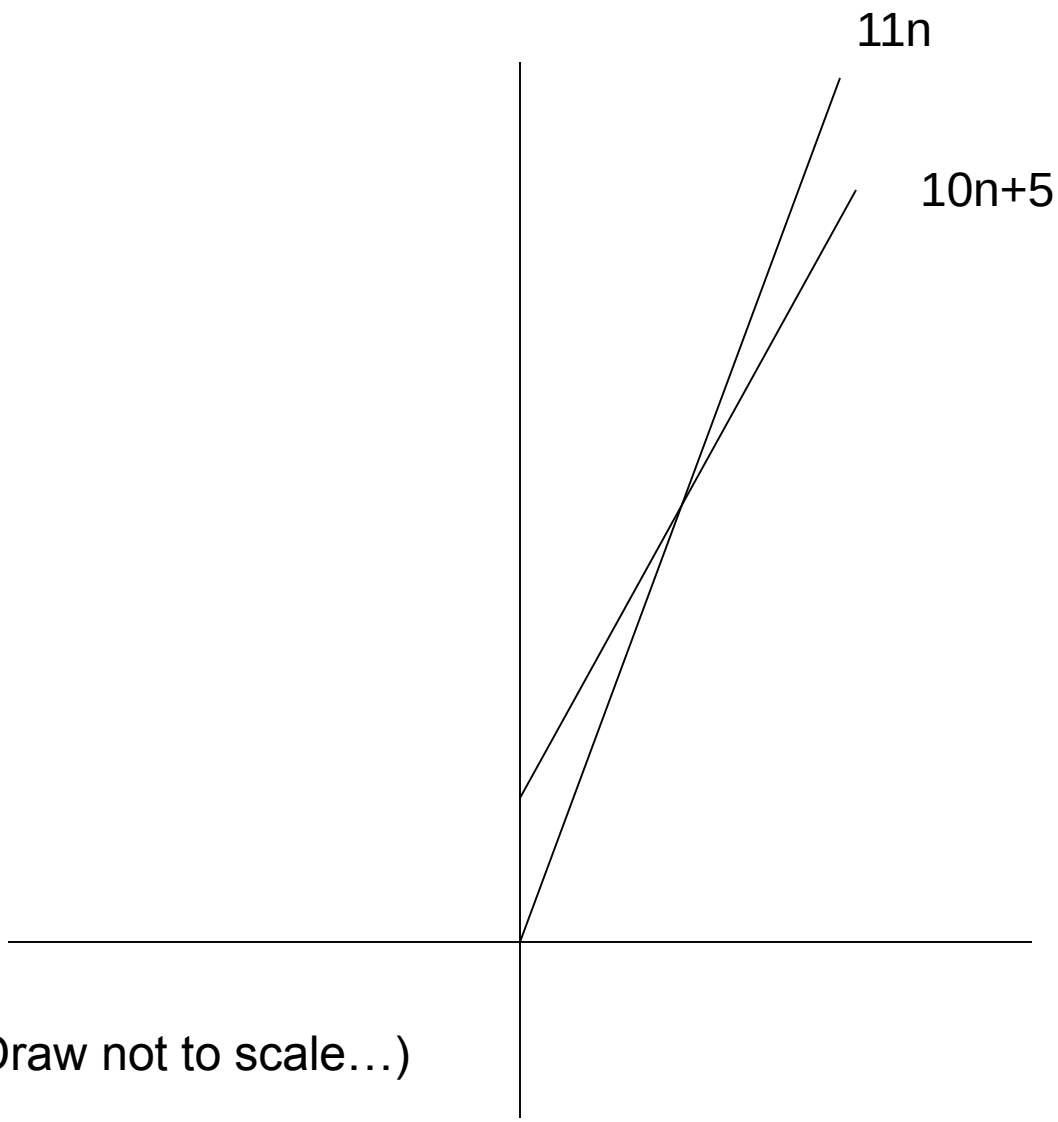
That been said…

- We never want to understate the importance of the role of the constant in practice.
  - Actually this constant surely matters a lot.
- But that's often a different category of jobs. As always, there is a division of labor:
  - First, theoreticians look at problems at a coarse scale, pinning down a rough position in the spectrum
  - Then, engineers elaborate on more (*ad hoc*) details, to finally get an algorithm as fast as possible.

The big O, Ω, and Θ notations.

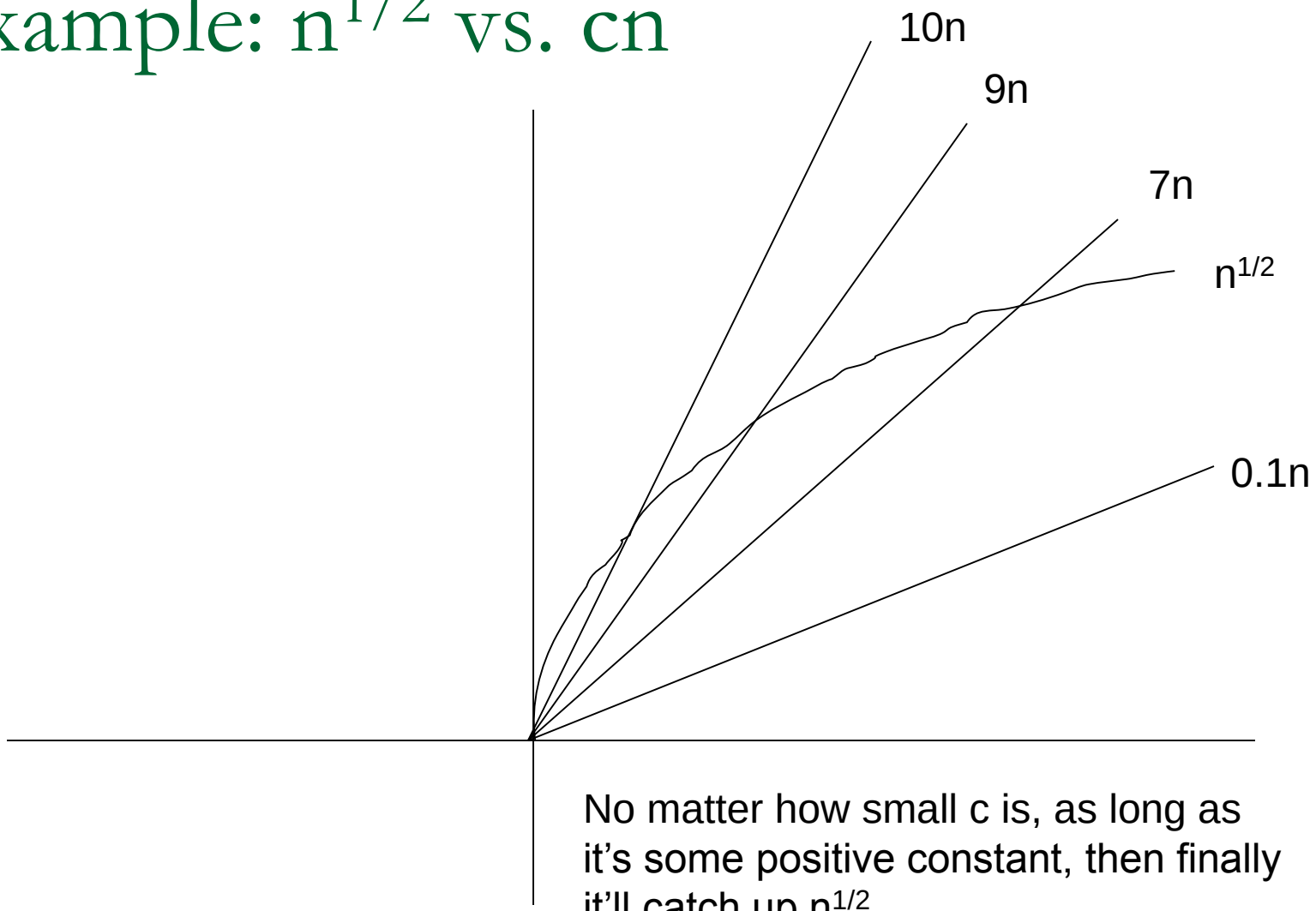
- $f(n) = O(n)$: $f(n) \leq c \cdot n$ for some constant c and large n.
  - i.e. $\exists c$, $\exists N > 0$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot n$.
- For example, $f(n) = 10n$.
  - Let $c = 10$, $N = 1$, then $\forall n > N$, we have $f(n) = 10n$.
  - So $10n = O(n)$.
  - Same for $100n$, $1000n$.
- How about $10n + 5$?
  - Let $c = 11$, $N = 5$, then $\forall n > N$, we have

    $f(n) = 10n + 5 \leq 11n$ (since $n > 5$).


11n
10n+5
(Draw not to scale…)

- In general:
- $f(n) = O(g(n))$: for some constant c, $f(n) \leq c \cdot g(n)$, when n is sufficiently large.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot g(n)$.
- $f(n) = o(g(n))$: for any constant c, $f(n) \leq c \cdot g(n)$, when n is sufficiently large.
  - i.e. $\forall c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot g(n)$.

Example: $n^{1/2}$ vs. cn



No matter how small c is, as long as it's some positive constant, then finally it'll catch up $n^{1/2}$.

Some examples

- Which increases faster?
  - $(100n^2, 0.01 * 2^n)$
  - $(0.1 * \log n\,, 10n)$
  - $(10^{10}n, 10^{-10}n^2)$

The other direction

- $f(n) = O(g(n))$: $f(n) \leq c \cdot g(n)$ for some constant c and large n.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \leq c \cdot g(n)$.
- $f(n) = \Omega(g(n))$: $f(n) \geq c \cdot g(n)$ for some constant c and large n.
  - i.e. $\exists c$, $\exists N$ s.t. $\forall n > N$, we have $f(n) \geq c \cdot g(n)$.
- $f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
  - i.e. $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for two constants c_1 and c_2 and large n.

Intuition

- $f = O(g)$
- $f = o(g)$
- $f = \Omega(g)$
- $f = \omega(g)$
- $f = \Theta(g)$

$\Leftrightarrow$

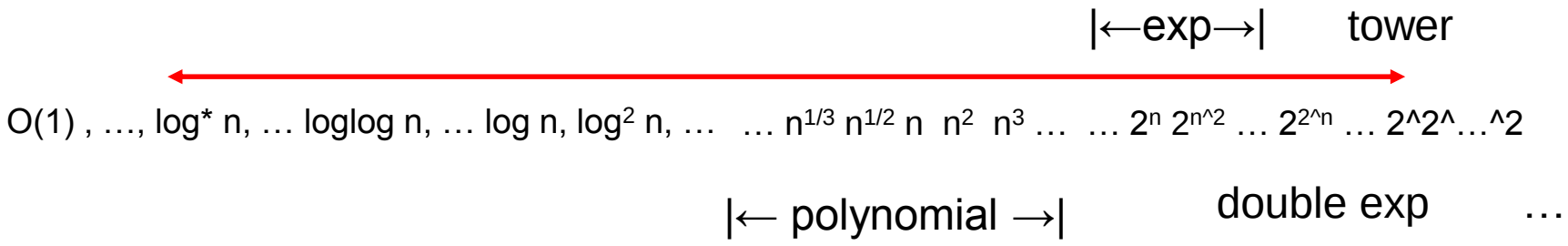
- $f \leq g$
- $f < g$
- $f \geq g$
- $f > g$
- $f = g$

- $f = O(g)$ $\Leftrightarrow$ $g = \Omega(f)$
- $f = o(g)$ $\Leftrightarrow$ $g = \omega(f)$
- $f = \Theta(g)$ $\Leftrightarrow$ $f = O(g)$ & $f = \Omega(g)$

$\Leftrightarrow$

- $f \leq g$ $\Leftrightarrow$ $g \geq f$
- $f < g$ $\Leftrightarrow$ $g > f$
- $f = g$ $\Leftrightarrow$ $f \leq g$ & $f \geq g$

Spectrum of functions…



- Even faster? $2^{2^{\cdots^2}}$ a tower of height n.
- Faster? 2^{2^n}, $2^{2^{2^n}}$, …
- Exponential: 2^n, 1.001^n, 2^{n^2}
- Polynomial: n, n^2, n^3, n^{100}, $n^{1/2}$, $n^{1/3}$, $n^{0.1}$, $n^{0.01}$,
- Logarithmic: $\log n$, $\log^2 n$, $\log^{1/2} n$,
- Slower? $\log\log n$, $\log\log\log n$, …
- Even slower? $\log^* n$.
  - It you take log, how many times to make n down to < 2?
  - $\log^*(10^{80}) = 5$. So $\log^* n$ is practically a constant.

Examples

- $10n = o(0.1n^2)$
- $n^2 = o(2^{n/10})$
- $n^{1/3} = \omega(10 \log n)$

- $n^3 = (n^2)^{3/2} = \omega(n^2)$
- $\log_2 n^2 = 2 \log_2 n = \Theta(\log_2 n)$
- $\log_2(2n) = 1 + \log_2 n = \Theta(\log_2 n)$

Other complexity measures

- We may have other complexity measures, such as space complexity.
  - We need a larger piece of paper to multiply two larger numbers.
- In this course, we'll mainly focus on time complexity.

In summary

- The complexity of a computational problem is an important issue
  - Actually the most important issue in theoretical computer science.
- The measure is more interesting for large input size n.
- Constants are usually ignored, hidden in the big O, Ω, and Θ notations.

In the rest of this course:

- We'll encounter various problems rising in practice, and we want to analyze how hard they are.
- Some are really hard.
  - How do we characterize those problems and know that they are hard?
- Others seem hard, but not really!
  - Let's see how to design good algorithms for them.

Questions?