Welcome to 6.006!

Reading: CLRS Ch. 1, 3, 4.1

Today: → Administrivia
→ Course overview
→ Two problems:
- exponentiation
- stock gain (divide & conquer)

Administrivia:

→ Website on Stellar
→ Please read the course handout

Important: Sign up at ALG.CSAIL.MIT.EDU by 5PM TODAY
(Recitation assignment tonight)

→ Pre-reqs: 6.01 (Python)
6.042 (discrete math, proofs)

→ Grade: 6 psets (20% Python + Later)
Quiz 1 (20% 3/10, 7:30-9:30PM)
Quiz 2 (20% 4/14, 7:30-9:30PM)
Final (40%)

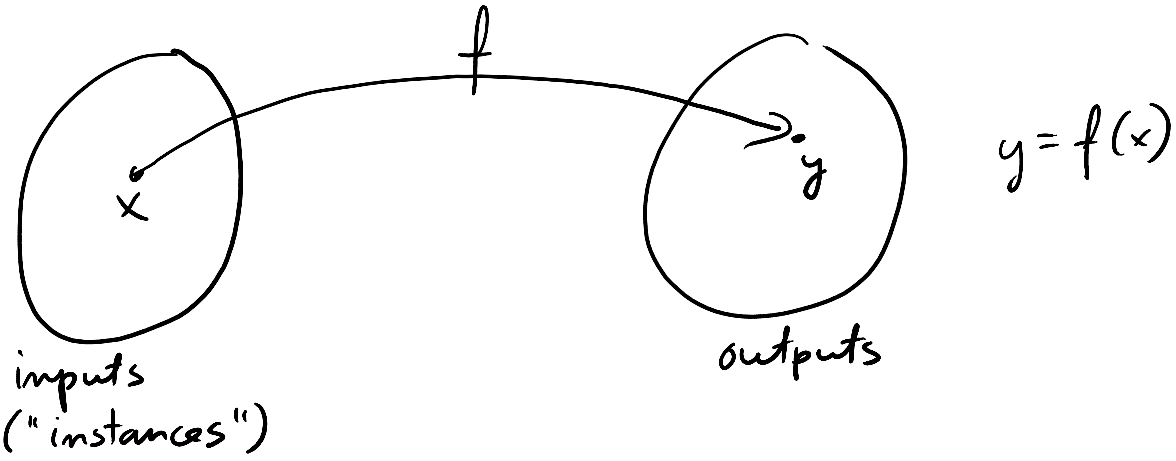
→ Policies: Late homework (grace days)
Missed quiz
Collaboration (!)

Course overview:

Key notion: Algorithm = well-specified procedure for solving a computational problem
(Mathematical abstraction of a computer program)

→ May be specified in English (preferred), or pseudo-code, as long as it is precise
(Avoid using real code!)

Problem: Specifies desired output for each input



E.g. x = an integer
y = smallest prime ≥ x

→ Difference between CS and math:
We care <u>how</u> f is computed

Want algorithms that are:

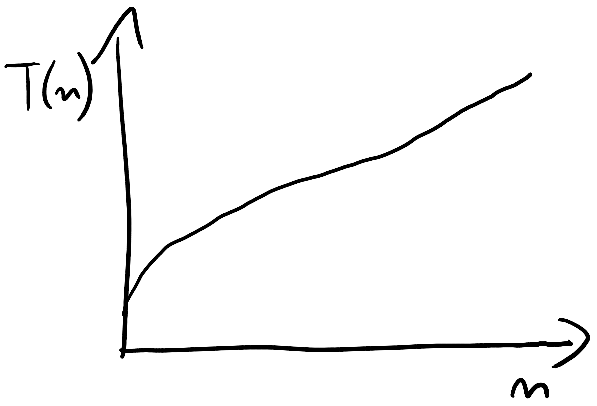
→ Correct (obviously!) } focus here
→ Fast ≈ Scalable }
→ Simple

Scalability:

→ Measure running time / space / etc. as input size grows

→ Our focus: $T(n)$ = run time as a function of input size n

(needs to be defined for each problem, e.g., n for $n \times n$ matrix input)



→ We care only about "big picture" here

→ ignore minor details (machine instruction set, compiler optimization, ...)

IMPORTANT IDEA ⇓

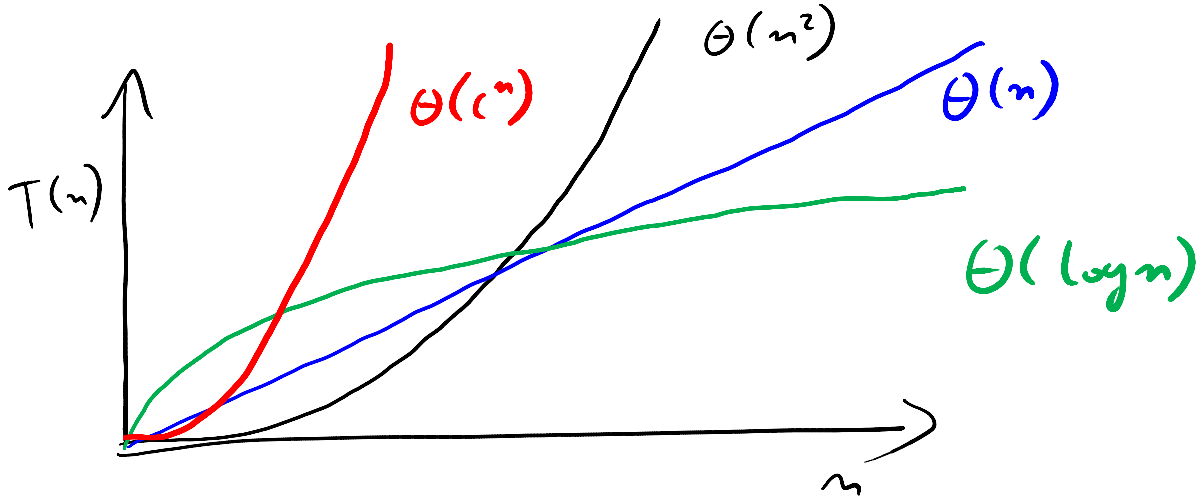
ignore constant factors and lower order terms

→ Key tool: Asymptotic analysis ($\Theta, O, \Omega, o, \omega$)

E.g., $5n^2 - 7n + 4 = \Theta(n^2)$

Examples:

$T(n) = \Theta(\log n)$	logarithmic	}	GREAT
$T(n) = \Theta(n)$	linear		
$T(n) = \Theta(n^2)$	quadratic		OK
$T(n) = \Theta(c^n)$	exponential		BAD!



→ Our chief goal here:
Learn how to reason about correctness and efficiency of algorithms in a precise and principled manner

Think: Algorithmic literacy ☺

Material overview:

- → Sorting (one of the most basic problems)
- → Data structures (organizing data to make it easy to access)
  - → Heaps
  - → Binary search trees
  - → Hashing
- → Graph search (how to explore graphs)
- → Shortest paths (how Google Maps works)
- → Iterative algorithms (optimization via repeated refinement)
- → Dynamic programming (structured exhaustive search)
- → Advanced topics (world beyond 6.006)

Some advice:

- → This is a highly conceptual class
- → You need to truly internalize the material. This takes time and regular work
- → Homework + Recitations: your best friends
- → Memorization / last-minute cramming will NOT work
- → Don't let yourself fall behind. If you need help, ask for it right away — we CAN help
- → Have FUN!

Exponentiation:

Problem: a, b, c – positive integers

Compute $a^b \pmod c$

(Ensures #s are not too big)

→ Fundamental computational problem (Esp., in cryptography)

→ Assume we can multiply mod c efficiently

(e.g., in time $M(c) = \Theta(\log^2 c)$)

Corresponds to "standard" long multiplication (It is quadratic in the size of bit representation of c)

Natural / Naive algorithm:

Just use the definition!

$$\text{Exp}(a, b, c) = \begin{cases} 1 & \text{if } b = 0 \\ a \cdot \text{Exp}(a, b-1, c) \pmod c & \text{o.w.} \end{cases}$$

Correctness? Ok, we just implemented definition

Running time?

$$T(a, b, c) = \begin{cases} \Theta(1) & \text{if } b = 0 \\ T(a, b-1, c) + \Theta(M(c)) & \text{o.w.} \end{cases}$$

Note: $\log a + \log b + \log c$ is the # of bits needed to store the whole input

$$\Downarrow$$

$$T(a, b, c) = \Theta(b \cdot M(c)) = \text{exponential in } \log b$$

($\log b$ = # of bits in b)

So, to process, say, 10 bits of input we need $\approx 2^{10}$ steps

BAD! :(

(Much) better idea: "Repeated squaring"

$$\text{Exp2}(a,b,c) = \begin{cases} 1 & \text{if } b=0 \\ (\text{Exp2}(a,\frac{b}{2},c))^2 \pmod c & \text{if } b>0 \text{ \& } b \text{ even} \\ a \cdot \text{Exp2}(a,b-1,c) & \text{o.w.} \end{cases}$$

Correctness? Associativity of multiplication

Running time?

$$T(a,b,c) = \Theta(\log b \cdot M(c))$$

Exponential speed up! :)

→ This simple but clever algorithm makes the modern crypto feasible (used in RSA & many other places)

→ Favorite algorithm of Ron Rivest

Stock gain problem:

(Rumored to be a Facebook interview question)

Problem: Given an array $A[0 \ldots n-1]$ of #s,
find $0 \le i^* \le j^* < n$ s.t.
$A[j^*] - A[i^*]$ is maximized (gain)

Story: You learned what the Facebook stock price will be for next n days, on condition you can buy once (on day i^*) & sell once (on day j^*)

Example: $A = [19\ 1\ 3\ 8\ 16\ 20\ 6\ 12]$
(i^* = 1, j^* = 20)
optimal gain = 19

How to solve this problem?

First attempt: Take $i^* = \arg\min_i A[i]$
$j^* = \arg\max_j A[j]$

Correctness? No! Could have $i^* > j^*$

$A = [19\ 20\ 3\ 8\ 16\ 1\ 6\ 12]$
(max = 20, min = 1)
max gain = 13

Naive algorithm: "Brute force"

Try all possible pairs (i^*, j^*), $0 \le i^* \le j^* < n$ and choose the one maximizing $A[j^*] - A[i^*]$

Correctness? Yes, we try all possible solutions

Running time?

Input size $= n$

$$\#\text{ pairs} = \binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$$

→ Quadratic time algorithm

→ Ok if n is indeed # of days

10 years $\approx$ 2500 weekdays

time $\approx$ 6250000

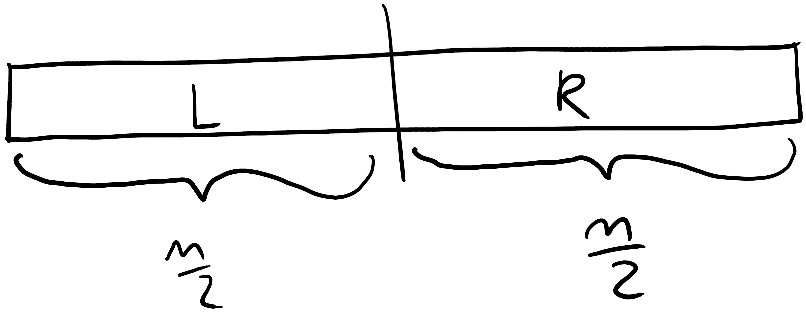
→ Not so much if n is large

10 years $\approx 2500 \cdot 8\text{h} \approx 10^6$ minutes

time $\approx 10^{12}$:(

Better algorithm?

Divide & conquer:



Three cases: → $i^* \in L$, $j^* \in L$ (recurse on L) $T(\frac{n}{2})$

(Here the "real" work is done) → $i^* \in R$, $j^* \in R$ (recurse on R) $T(\frac{n}{2})$

→ $i^* \in L$, $j^* \in R$ (special case) $\Theta(n)$

$A[i^*] = \underline{\text{min}}$ in L $\quad$ $A[j^*] = \underline{\text{max}}$ in R

At the end: Take the max of those 3 cases

Correctness? Already argued above

Running time?

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

2: # of subproblems; $\frac{n}{2}$: size of subproblems; $\Theta(n)$: handling special case

$$\Rightarrow T(n) = \Theta(n \log n)$$

$\Theta(n^2)$ vs. $\Theta(n \log n)$:

$n \approx 10^6$ $\quad \Theta(n^2) \rightarrow \approx 10^{12}$

$\Theta(n \log n) \rightarrow \approx 20 \cdot 10^6$

50 K x improvement!

→ See Python code posted

Divide & conquer: A general algorithmic technique

(will see more of it soon)

We got $\Theta(n \log n)$ algorithm.

Can we get even faster one?

Will see in the next lecture!