Welcome to 6.006!

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+LaTeX)
  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

\[ y = f(x) \]

- E.g. \( x = \text{an integer} \)
  \( y = \text{smallest prime} \geq x \)

- **Difference between CS and math:** We care how \( f \) is computed.
Want algorithms that are:

→ Correct (obviously!)
→ 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 \( n \)

\[
\begin{array}{c}
T(n) \\
\downarrow \\
\uparrow \\
\rightarrow \quad n
\end{array}
\]

(needs to be defined for each problem, e.g., \( n \) for \( nxn \) 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 \( (\Theta, \Omega, \omega, \mathcal{O}) \)

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

\[ T(n) = \Theta(\log n) \] Logarithmic \{ GREAT \\
\[ T(n) = \Theta(n) \] Linear \{ OK \%\\
\[ T(n) = \Theta(n^2) \] 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

\[ \text{compute } a^b \pmod{c} \]

(Ensures #s are not too big)

\[ \Rightarrow \text{Fundamental computational problem} \]

(Esp., in cryptography)

\[ \Rightarrow \text{Assume we can multiply mod } c \text{ efficiently} \]

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

\[ \Rightarrow \text{Natural/Naive algorithm:} \]

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?

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

\[ \text{Note: } \log_a + \log_b + \log_c \text{ is the # of bits needed to store the whole input} \]

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

\[ \Rightarrow \text{BAD!} \]

\[ \frac{\text{# of bits in } b}{\log} \]
(Much) better idea: "Repeated squaring"

\[
\exp_2(a, b, c) = \begin{cases} 
1 & \text{if } b = 0 \\
\left(\exp_2(a, b/2, c)\right)^2 \pmod{c} & \text{if } b > 0 \text{ \& even} \\
 a \cdot \exp_2(a, b-1, c) & \text{0, U.}
\end{cases}
\]

Correctness?  Associativity of multiplication

Running time?

\[
T(a, b, c) = \Theta\left(\log b \cdot M(c)\right)
\]

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 interviews question)

Problem: Given an array $A[0...n-1]$ of #s,
find $0 \leq i^* \leq j^* < n$ s.t.

$$A[j^*] - A[i^*]$$

is maximized (gain)

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

Example: $A = [19, 13, 8, 16, 20, 12]$

$\overset{\text{optimal gain}}{\text{max}} = 19$

How to solve this problem?

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

$\overset{\text{correctness? No! could have } i^* > j^*}{\text{max}}$

$$A = [19, 20, 3, 8, 16, 1, 6, 12]$$

$\overset{\text{max gain}}{\text{min}} = 13$
Naïve algorithm: "Brute force"

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

Coredness? Yes, we try all possible solutions.

Running time?

\[
\text{Input size} = n \\
\text{# pairs} = \binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)
\]

\(\rightarrow\) Quadratic time algorithm

\(\rightarrow\) Ok if \(n\) is indeed # of days

10 years \(\approx 2500\) weekdays

\(\text{time} \approx 625,000,000\)

\(\rightarrow\) Not so much if \(n\) is (ar - \\
10 years \(\approx 2500 \cdot 8 \text{h} \approx 10^6\) minutes

\(\text{time} \approx 10^{12} : (\)

Better algorithm?
Divide & Conquer:

\[
\begin{array}{c|c}
L & R \\
\hline
\frac{n}{2} & \frac{n}{2}
\end{array}
\]

Three cases:
- \( i^* \in L, j^* \in L \) (recursively on L) \( T(\frac{n}{2}) \)
- \( i^* \in R, j^* \in R \) (recursively on R) \( T(\frac{n}{2}) \)
- \( i^* \in L, j^* \in R \) (special case) \( \Theta(n) \)

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

At the end: Take the max of those 3 cases.

Correctness? Already argued above.

Running time:

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

\( \Rightarrow \) \( T(n) = \Theta(n \log n) \)
\( \Theta(n^2) \) vs. \( \Theta(m \log n) \):

\[
\begin{align*}
  n = 10^6 & \quad \Theta(n^2) \rightarrow \approx 10^{12} \\
  \Theta(m \log n) \rightarrow & \approx 20 \cdot 10^6
\end{align*}
\]

50K x improvement!

\( \rightarrow \) See Python code posted

Divide & conquer: A general algorithmic technique

(Will see more of it soon)

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

Can we get even faster one?

Will see in the next lecture!