Today: Peak Finding
- administrivia
- course overview
- peak finding problem <10
- divide & conquer

6.006: Introduction to Algorithms
Profs. Erik Demaine, Ronald Rivest, & Nir Shavit
TAs: Ashwin Deshpande, Chris Graves, Jennifer Jang,
Gurtej Kanwar, Vlad Kontsevoi, Justin Kopinsky,
Quangquan Liu, Arvind Thiagarajan, Sherwin Wu
http://stellar.mit.edu/S/course/6/fa13/6.006/
6.006-staff@mit.edu
Handout 1:
* sign up at alg.csail.mit.edu by 5pm. TODAY
  - recitation assignment tonight
    (note: 2 per week / 1 per lecture)
  - sign up for Piazza (recommended)

- prereqs: 6.01 (Python)
  & 6.042 (discrete math)
- grading: 6 psets (30%, Python + LaTeX)
  Quiz 1  (20%, Oct. 16, 7:30-9:30)
  Quiz 2  (20%, Nov. 13, 7:30-9:30)
  final  (30%)
- read policies: late psets, missed quiz, collaboration
Algorithm = mathematical abstraction of a computer program = well-specified procedure for solving a computational problem
-usually: finite sequence of operations described via structured English, pseudocode, or real code (preferred)

History/naming: al-Khwārizmī (c. 780-850) "al-khā-raz-mē"
- father of algebra
- author of The Compendious Book on Calculation by Completion and Balancing [c.830]
- some of the first algs: linear & quadratic eqns.

Want an algorithm that's:
- correct
- fast
- small space
- general, simple, clever...
Scalability: measure time/space/etc. as problem size $n$ grows $\to \infty$

- **BAD**: $\Theta(2^n)$
- **GOOD**: $\Theta(n)$
- **GREAT**: $\Theta(\log n)$

- big idea: IGNORE CONSTANT FACTORS using asymptotic notation ($\Theta, O, \Omega, \omega, \theta$)
  
e.g. $5n^2 - 7n + 4 = \Theta(n^2)$

$\Rightarrow$ minor details (machine instruction set, compiler optimization, ...) irrelevant
- dominant for large $n$

- polynomial time $= \Theta(n), \Theta(n^2), \Theta(n^3), \ldots$ = "GOOD"
- exponential time e.g. $\Theta(2^n) = "BAD"$
Main topics in 6.006:
- sorting
- data structures:
  - heaps
  - binary search trees
  - hashing
- numerics
- graph search
- shortest paths
- techniques:
  - divide & conquer
  - dynamic programming
- complexity (NP-complete)

Sample applications:
- spreadsheets
  - Google
- simulation
  - Google Maps
- file sync.
  - computing π, big ints.
  - Rubik's Cube
- optimization
  - lots...
Peak finding: find a local minimum/maximum in a 1D or 2D array
(think: blind Geordi LaForge searching for water accumulation on alien mountain range, with transporter)

1D: input = array \( A[0..n-1] \)
output = \( i \) such that \( A[i-1] \leq A[i] \geq A[i+1] \)
where \( A[-1] = -\infty \) \& \( A[n] = -\infty \)

Algorithm 1: brute force
- test \( A[0], A[1], \ldots, A[n-1] \) for peakyness
- \( \Theta(1) \) time each \( \Rightarrow \Theta(n) \) time

Algorithm 2: \( \max(A) \) is a local max
- \( \Theta(n) \) time

Algorithm 3: divide \& conquer
- look at element \( i \)
- if it's a peak: done, return it
- else it has a neighbor < \( A[i] \)
  - if \( A[i-1] < A[i] \): must be peak to left
  - if \( A[i+1] > A[i] \): must be peak to right
  \( \Rightarrow \) recurse on \( A[i] \) or \( A[i+1] \)

(Why? walk left/right until descend or hit end \( \Rightarrow \) peak)
\( \langle i \rangle \langle \rangle \langle i \rangle \)

\( \Rightarrow \) left with \( \max \{i, n-i-1\} \) elements in the \text{worst case}

\( \Rightarrow \) set \( i = n-i-1 \) to balance

\( \Rightarrow \) \( i = (n-1)/2 \Rightarrow \text{MIDDLE} \)

\text{Divide & conquer algorithm:}
1. divide input into part(s)
2. conquer (solve) each part recursively
3. combine result(s) to solve original

- if size-\( n \) input divided into \( n_1, n_2, \ldots, n_k \) then \( T(n) = \text{divide time} + T(n_1) + T(n_2) + \ldots + T(n_k) \) \( \{ \text{RECURSION} \} \)

- \text{1D peak:} \( T(n) \leq T(n/2) + \Theta(1) \)
- solve: \( T(n) \leq T(n/2) + c \leq T(n/4) + c + c \leq \ldots \leq T(n/2^k) + c \cdot k \leq T(n/2^{\log_2 n}) + c \cdot \log n = T(1) + c \log n = O(\log n) \)

\( \text{can't use } \Theta(1) \text{ here because of } \ldots \) - need to keep track of constant used \( \log n \) times
2D: input = n×n matrix A
output = (i, j) such that:

\[
\begin{align*}
A[i-1, j] & \leq A[i,j] & \leq A[i+1, j] \\
\end{align*}
\]

Algorithm 1: brute force
- test all \( n^2 \) elements for peakyness
\( \Rightarrow O(n^2) \) time

Algorithm 2: reduce to 1D
1. take max element in each column
2. solve 1D problem on max array
\( \Theta(n^2) \) time

Algorithm 3: be lazy about 1
- solve 1D problem on max array
- for each of \( O(\log n) \) accesses to max, compute in \( \Theta(n) \) time
\( \Rightarrow O(n \log n) \) time
Algorithm 4: 2D divide & conquer
- window = first, middle, & last row & column
- if it has a peak: done
- else: find max(window)
  - must have a larger neighbor
    ⇒ can't be at window corners, center, ...
    ⇒ recurse in unique subwindow containing it

Correctness: subwindow contains peak
- walk up (N/S/E/W as appropriate) from neighbor
  until get stuck at a peak
- walk ≥ neighbor > max ≥ subwindow boundary
  ⇒ never left subwindow

Time: \( T(n) = T(n/2) + O(n) \)
\( n \times n \Rightarrow \frac{n}{2} \times \frac{n}{2} \)
- solve: \( T(n) \leq T(n/2) + c \cdot n \)
  \( \leq T(n/4) + c \cdot \frac{n}{2} + c \cdot n \)
  \( \leq T(n/8) + c \cdot \frac{n}{4} + c \cdot \frac{n}{2} + c \cdot n \)
  \( \leq \ldots \)
  \( \leq T(1) + c \cdot n (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots) \)
  \( < 2^n \)
  \( = O(n) \)