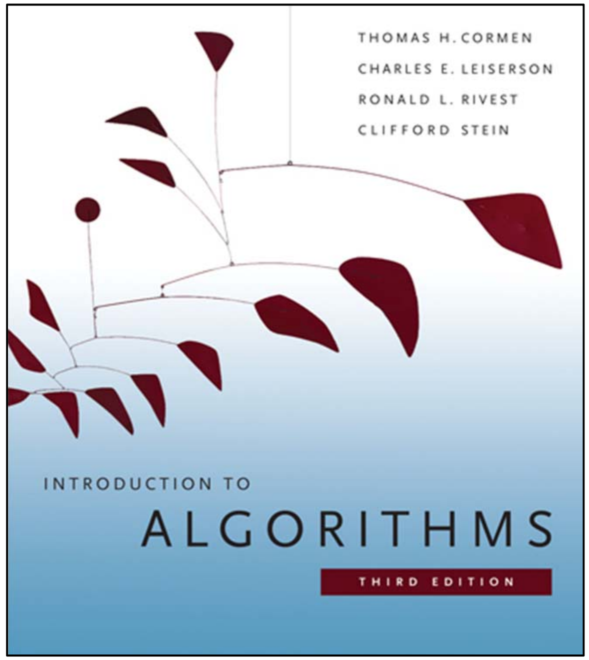
6.006

Introduction to Algorithms



Lecture 2: Peak Finding

Prof. Erik Demaine

Today

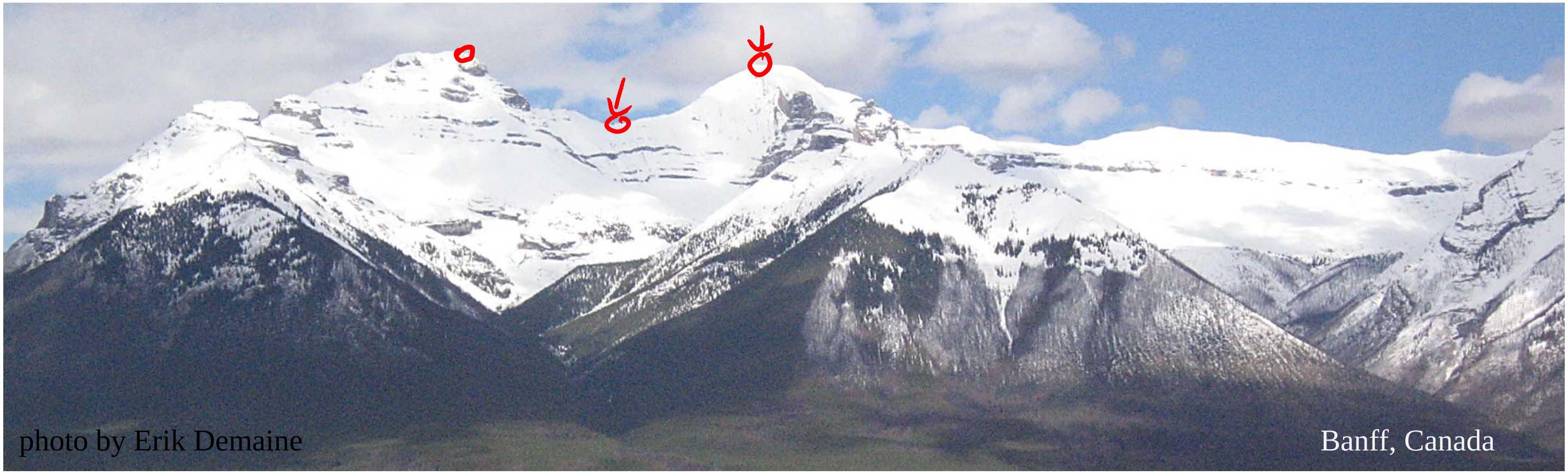
- Peak finding *(new problem)*
  - 1D algorithms
  - 2D algorithms
- Divide & conquer *(new technique)*

Finding Water…

IN SPACE

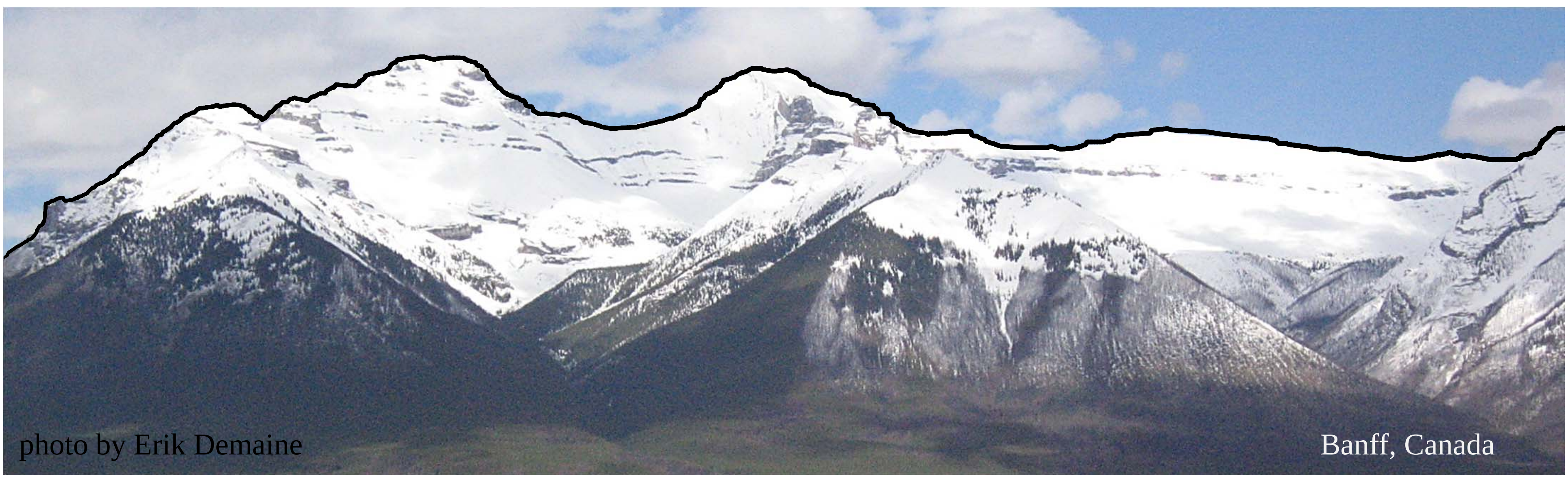
http://en.wikipedia.org/wiki/File:GeordiLaForge.jpg

- You are Geordi LaForge
- Trapped on alien mountain range
- Need to find a pool where water accumulates
- Can teleport, but can't see



Finding Water…
IN SPACE

- <u>Problem:</u> Find a local minimum or maximum in a terrain by sampling



1D Peak Finding

- Given an array $A[0..n-1]$:

A:	$-\infty$	1	2	6	5	3	7	4	$-\infty$
		0	1	2	3	4	5	6	

- $A[i]$ is a **peak** if it is not smaller than its neighbor(s):

$$A[i-1] \leq A[i] \geq A[i+1]$$

where we imagine

$$A[-1] = A[n] = -\infty$$

- Goal: Find *any* peak

“Brute Force” Algorithm

- Test all elements for peakyness

for i in range(n):

    if $A[i-1] \leq A[i] \geq A[i+1]$:   } O(1)   } O(n)

        return i

A:	1	2	6	5	3	7	4
	0	1	2	3	4	5	6

Algorithm 1½

- $\max(A)$
  - Global maximum is a local maximum

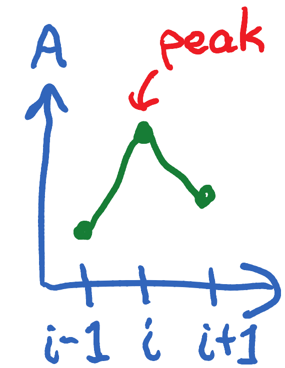
$m = 0$
for i in range$(1, n)$:
        if $A[i] > A[m]$:   } $\Theta(1)$ } $\Theta(n)$
                $m = i$
return m

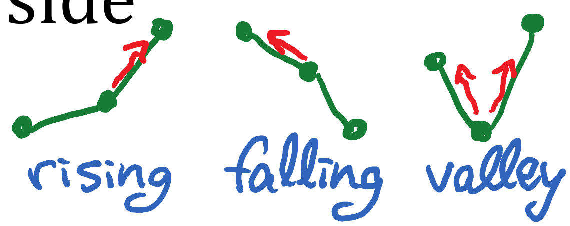
A:

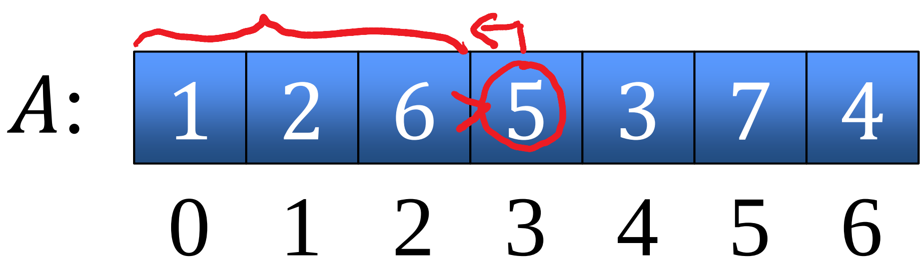
1	2	6	5	3	7	4
0	1	2	3	4	5	6

Cleverer Idea

- Look at any element $A[i]$ and its neighbors $A[i-1]$ & $A[i+1]$
  - If peak: return i
  - Otherwise: locally rising on some side
    - Must be a peak in that direction
    - So can throw away rest of array, leaving $A[:i]$ or $A[i+1:]$







Where to Sample?

- Want to minimize the worst-case remaining elements in array
  - Balance $A[:i]$ of length i with $A[i+1:]$ of length $n-i-1$
  - $i = n-i-1$
  - $i = (n-1)/2$: **middle element**
  - Reduce n to $(n-1)/2$

A:

A:	1	2	6	5	3	7	4
	0	1	2	3	4	5	6

Algorithm

peak1d(A, i, j):

    $m = \lfloor (i + j)/2 \rfloor$
    if $A[m-1] \le A[m] \ge A[m+1]$:
        return m
    elif $A[m-1] > A[m]$:
        return peak1d$(A, i, m-1)$
    elif $A[m] < A[m+1]$:
        return peak1d$(A, m+1, j)$

A:	1	2	6	5	3	7	4
	0	1	2	3	4	5	6

Divide & Conquer

- General design technique:

1. **Divide** input into part(s)
2. **Conquer** each part recursively
3. **Combine** result(s) to solve original problem

- 1D peak:

1. One half
2. Recurse
3. Return

Divide & Conquer Analysis

- **Recurrence** for time $T(n)$ taken by problem size n

1. **Divide** input into part(s):
   $n_1, n_2, \ldots, n_k$
2. **Conquer** each part recursively
3. **Combine** result(s) to solve original problem

$T(n) =$

divide cost +

$T(n_1) + T(n_2) + \cdots + T(n_k)$

+ combine cost

1D Peak Finding Analysis

- Divide problem into 1 problem of size $\sim \frac{n}{2}$
- Divide cost: $O(1)$
- Combine cost: $O(1)$
- Recurrence:

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Solving Recurrence

don't use O(1) notation to keep track of constant

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

$$T(n) = T\left(\frac{n}{4}\right) + c + c$$

$$T(n) = T\left(\frac{n}{8}\right) + c + c + c$$

$$T(n) = T\left(\frac{n}{2^k}\right) + c\,k$$

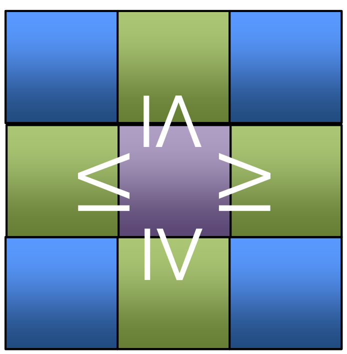
$$T(n) = T\left(\frac{n}{2^{\lg n}}\right) + c\lg n$$

$$T(n) = T(1) + c\lg n$$

$$T(n) = \Theta(\lg n)$$

2D Peak Finding

- Given $n \times n$ matrix of numbers
- Want an entry not smaller than its (up to) 4 neighbors:



9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

Divide & Conquer #0

- Looking at center element doesn't split the problem into pieces…

9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

Divide & Conquer #½

- Consider max element in each column
- 1D algorithm would solve max array in $O(\lg n)$ time
- But $\Theta(n^2)$ time to compute max array

9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

9	9	9	3	5	9	8

Divide & Conquer #1

- Look at center column
- Find global max within
- If peak: return it
- Else:
  - Larger left/right neighbor
  - Larger max in that column
  - Recurse in left/right half
- <u>Base case:</u> 1 column
  - Return global max within

9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

9	9	9	3	5	9	8

Analysis #1

- $O(n)$ time to find max in column
- $O(\lg n)$ iterations (like binary search)
- $O(n \lg n)$ time total

9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

- Can we do better?

Divide & Conquer #2

- Look at boundary, center row, and center column (**window**)
- Find global max within
- If it's a peak: return it
- Else:
  - Find larger neighbor
  - Can't be in window
  - Recurse in quadrant, including green boundary

0	0	0	0	0	0	0	0	0
0	9	3	5	2	4	9	8	0
0	7	2	5	1	4	0	3	0
0	9	8	9	3	2	4	8	0
0	7	6	3	1	3	2	3	0
0	9	0	6	0	4	6	4	0
0	8	9	8	0	5	3	0	0
0	2	1	2	1	1	1	1	0
0	0	0	0	0	0	0	0	0

Correctness

- Lemma: If you enter a quadrant, it contains a peak of the overall array [climb up]
- Invariant: Maximum element of window never decreases as we descend in recursion
- Theorem: Peak in visited quadrant is also peak in overall array

0	0	0	0	0	0	0	0	0
0	9	3	5	2	4	9	8	0
0	7	2	5	1	4	0	3	0
0	9	8	9	3	2	4	8	0
0	7	6	3	1	3	2	3	0
0	9	0	6	0	4	6	4	0
0	8	9	8	0	5	3	0	0
0	2	1	2	1	1	1	1	0
0	0	0	0	0	0	0	0	0

→ proofs in recitation

Analysis #2

- Reduce $n \times n$ matrix to $\sim \frac{n}{2} \times \frac{n}{2}$ submatrix in $O(n)$ time ($|\text{window}|$)

0	0	0	0	0	0	0	0	0
0	9	3	5	2	4	9	8	0
0	7	2	5	1	4	0	3	0
0	9	8	9	3	2	4	8	0
0	7	6	3	1	3	2	3	0
0	9	0	6	0	4	6	4	0
0	8	9	8	0	5	3	0	0
0	2	1	2	1	1	1	1	0
0	0	0	0	0	0	0	0	0

$$T(n) = T\left(\frac{n}{2}\right) + c\,n$$

$$T(n) = T\left(\frac{n}{4}\right) + c\,\frac{n}{2} + c\,n$$

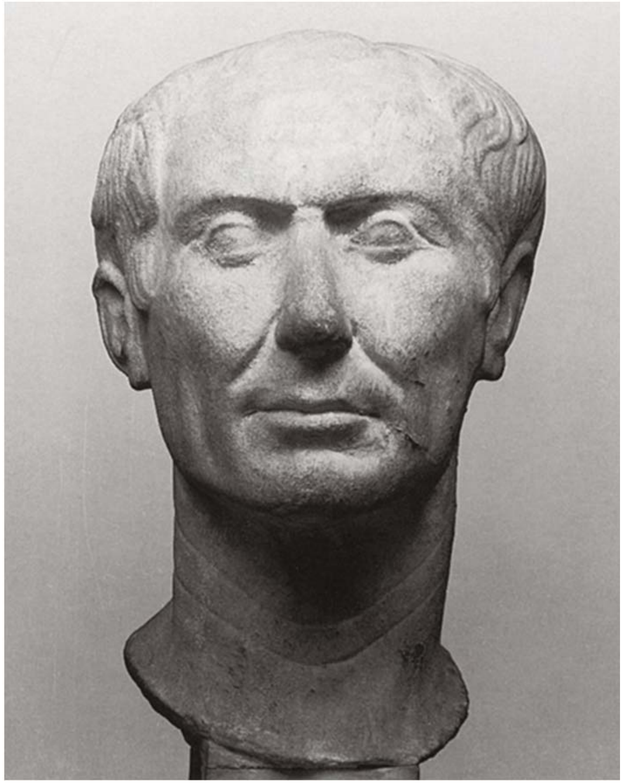
$$T(n) = T\left(\frac{n}{8}\right) + c\,\frac{n}{4} + c\,\frac{n}{2} + c\,n$$

$\Theta(n)$

$$T(n) = T(1) + c\left(1 + 2 + 4 + \cdots + \frac{n}{4} + \frac{n}{2} + n\right)$$

Divide & Conquer Wrapup

- Leads to surprisingly efficient algorithms
- Not terribly general, but still quite useful
- We'll use it again in
  – Module 4 (sorting)
  – Module 8 (geometry)



http://en.wikipedia.org/wiki/File:CaesarTusculum.jpg