

# Lecture 1: Introduction and Peak Finding

## Lecture Overview

- Administrivia
- Course Overview
- “Peak finding” problem — 1D and 2D versions

## Course Overview

This course covers:

- Efficient procedures for solving problems on large inputs (Ex: U.S. Highway Map, Human Genome)
- Scalability
- Classic data structures and elementary algorithms (CLRS text)
- Real implementations in Python
- Fun problem sets!

The course is divided into 8 modules — each of which has a motivating problem and problem set(s) (except for the last module). Tentative module topics and motivating problems are as described below:

1. Algorithmic Thinking: Peak Finding
2. Sorting & Trees: Event Simulation
3. Hashing: Genome Comparison
4. Numerics: RSA Encryption
5. Graphs: Rubik’s Cube
6. Shortest Paths: Caltech → MIT
7. Dynamic Programming: Image Compression
8. Advanced Topics

## Peak Finder

### One-dimensional Version

Position 2 is a peak if and only if  $b \geq a$  and  $b \geq c$ . Position 9 is a peak if  $i \geq h$ .


1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h	i

Figure 1: a-i are numbers

Problem: Find a peak if it exists (Does it always exist?)

### Straightforward Algorithm

6	7	4	3	2	1	4	5
---	---	---	---	---	---	---	---



Start from left

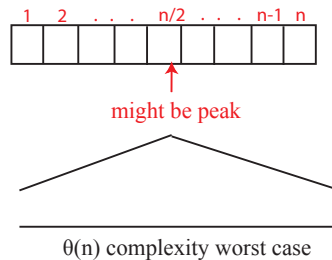
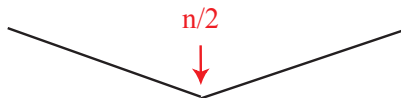


Figure 2: Look at  $n/2$  elements on average, could look at  $n$  elements in the worst case

What if we start in the middle? For the configuration below, we would look at  $n/2$  elements. Would we have to ever look at more than  $n/2$  elements if we start in the middle, and choose a direction based on which neighboring element is larger than the middle element?



Can we do better?

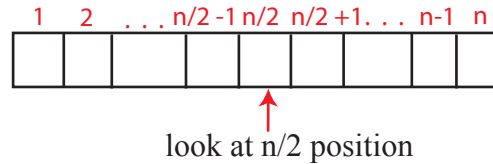


Figure 3: Divide & Conquer

- If  $a[n/2] < a[n/2 - 1]$  then only look at left half  $1 \dots n/2 - 1$  to look for peak
- Else if  $a[n/2] < a[n/2 + 1]$  then only look at right half  $n/2 + 1 \dots n$  to look for peak
- Else  $n/2$  position is a peak: WHY?

$$a[n/2] \geq a[n/2 - 1]$$

$$a[n/2] \geq a[n/2 + 1]$$

What is the complexity?

$$T(n) = T(n/2) + \underbrace{\Theta(1)}_{\text{to compare } a[n/2] \text{ to neighbors}} = \Theta(1) + \dots + \Theta(1) \text{ (} \log_2(n) \text{ times)} = \Theta(\log_2(n))$$

In order to sum up the  $\Theta(i)$ 's as we do here, we need to find a constant that works for all. If  $n = 1000000$ ,  $\Theta(n)$  algo needs 13 sec in python. If algo is  $\Theta(\log n)$  we only need 0.001 sec. Argue that the algorithm is correct.

## Two-dimensional Version

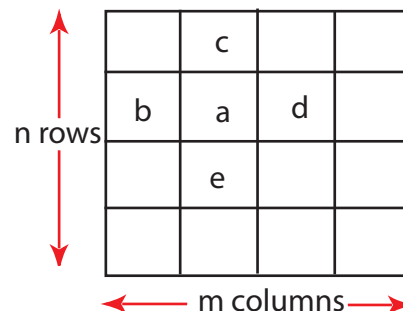


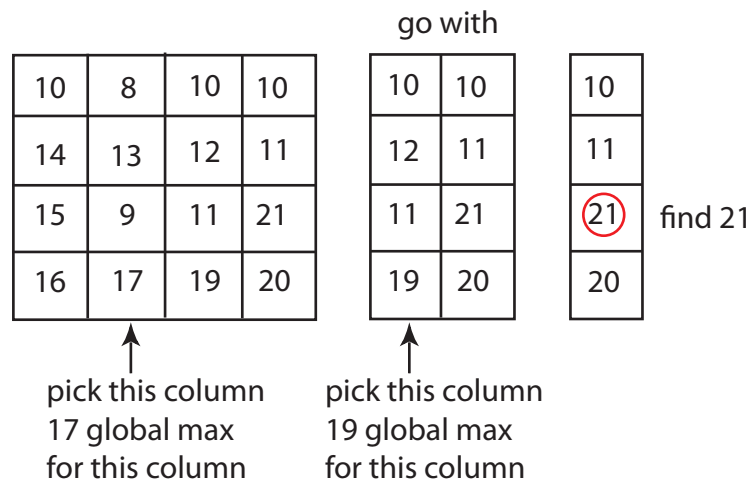
Figure 4: Greedy Ascent Algorithm:  $\Theta(nm)$  complexity,  $\Theta(n^2)$  algorithm if  $m = n$

$a$  is a 2D-peak iff  $a \geq b, a \geq d, a \geq c, a \geq e$



**Attempt # 2**

- Pick middle column  $j = m/2$
- Find global maximum on column  $j$  at  $(i, j)$
- Compare  $(i, j - 1), (i, j), (i, j + 1)$
- Pick left columns of  $(i, j - 1) > (i, j)$
- Similarly for right
- $(i, j)$  is a 2D-peak if neither condition holds  $\leftarrow$  WHY?
- Solve the new problem with half the number of columns.
- When you have a single column, find global maximum and you're done.

**Example of Attempt #2****Complexity of Attempt #2**

If  $T(n, m)$  denotes work required to solve problem with  $n$  rows and  $m$  columns

$$\begin{aligned}
 T(n, m) &= T(n, m/2) + \Theta(n) \text{ (to find global maximum on a column — (n rows))} \\
 T(n, m) &= \underbrace{\Theta(n) + \dots + \Theta(n)}_{\log m} \\
 &= \Theta(n \log m) = \Theta(n \log n) \text{ if } m = n
 \end{aligned}$$

Question: What if we replaced global maximum with 1D-peak in Attempt #2? Would that work?

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.006 Introduction to Algorithms  
Fall 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.