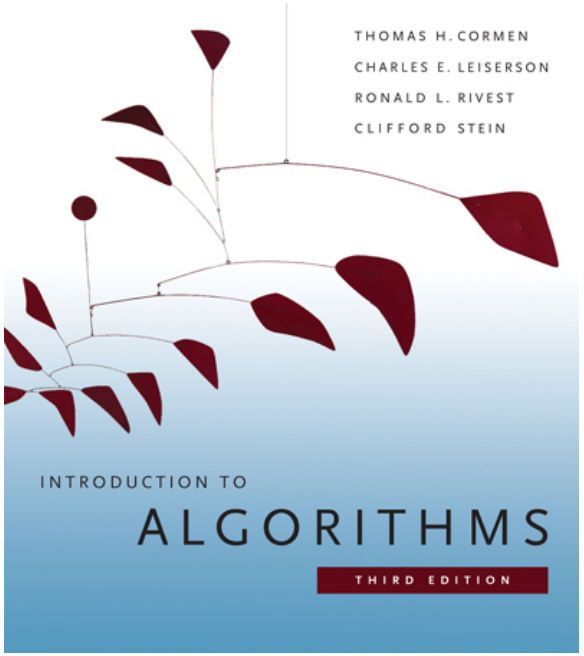
6.006- *Introduction to Algorithms*



Lecture 2

Prof. Silvio Micali

Menu

Problem: peak finding

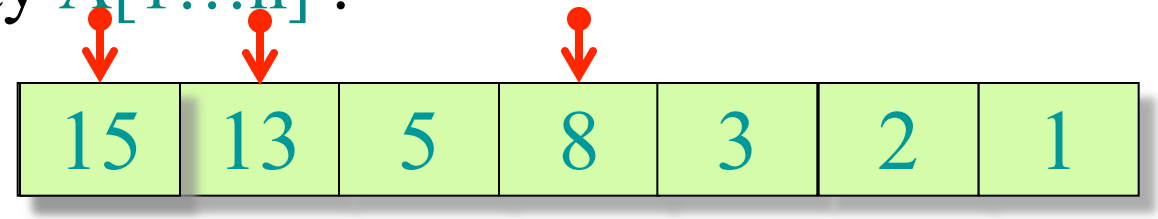
1 dimension

2 dimensions

Technique: *Divide and conquer*

Peak Finding: 1D

Consider an array A[1…n] :



Element A[i] is a *peak* if **not smaller** than its neighbor(s).

if $i \neq 1, n$: $A[i] \geq A[i-1]$ and $A[i] \geq A[i+1]$
If $i=1$: $A[1] \geq A[2]$
If $i=n$: $A[n] \geq A[n-1]$

Problem: find ***any*** peak.

Peak-Finding Ideas ?

Algorithm I:

Scan the array from left to right

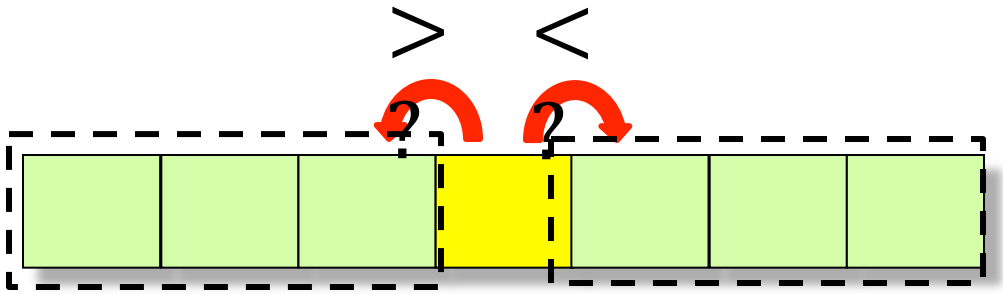
Compare each A[i] with its neighbors

Exit when found a peak

Complexity:

Might need to scan all elements, so $T(n)=\Theta(n)$

Next Idea



Algorithm II:

Compare middle element with neighbors

If A[n/2-1]>A[n/2]

then search for a peak among A[1]… A[n/2-1]

Else, if A[n/2]<A[n/2+1]

then search for a peak among A[n/2]… A[n]

Else A[n/2] is a peak!

Running time ?

Algorithm II: Complexity

Algorithm II: Complexity

- We have

Time needed to find peak in array of length n

Recursion

Time for comparing A [n/2] with neighbors

$$T(n) = T(n/2) + \Theta(1)$$

- Unraveling the recursion,

$$T(n) = \underbrace{\Theta(1) + \Theta(1) + \ldots + \Theta(1)}_{\log_2 n} = \mathbf{\Theta(\log n)}$$

- log n is much much better than n !

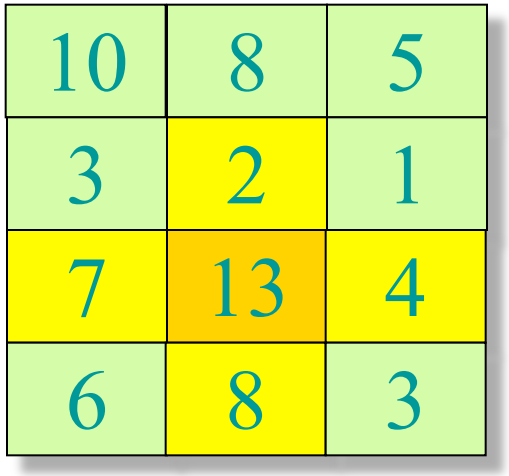
Divide and Conquer

- Very powerful design tool:
  - *Divide* input into multiple **disjoint** parts
  - *Conquer* each of the parts **separately** (using recursive call)

- *Occasionally*, we need to **combine** results from different calls (not used here)

Peak Finding: 2D

Consider a 2D array A[1…n, 1…m] :



10	8	5
3	2	1
7	13	4
6	8	3

A[i] is a *2D peak* if not smaller than its (at most 4) neighbors.

Problem: find any 2D peak.

2D-Peak-Finding Ideas?

Algorithm 0:

For each row, until you find a peak:

1. find a row-peak
2. compare it with North- and South-neighbors
3. If $\geq$, then done

?

Algorithm I: recycle better 1D algorithm

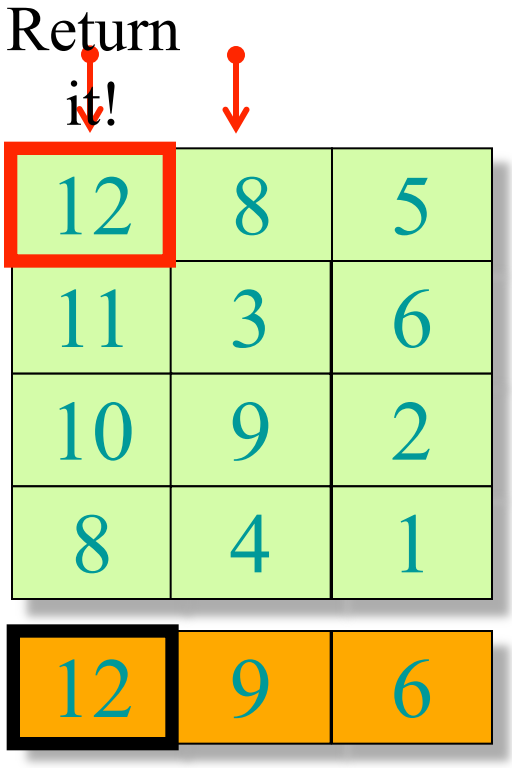
For each column j, find its *global* maximum B[j]

Apply 1D peak finder to find a peak (say B[j]) of B[1...m]

Correctness: …

Complexity: $\Theta(n \cdot m)$

Recycling is an art…



Algorithm I': use the 1D algorithm

- **Recall:** 1D peak finder uses only O(log m) entries of B
- Modify Algorithm I so that it only computes B[j] ***when needed*** **!**
- Total time ?

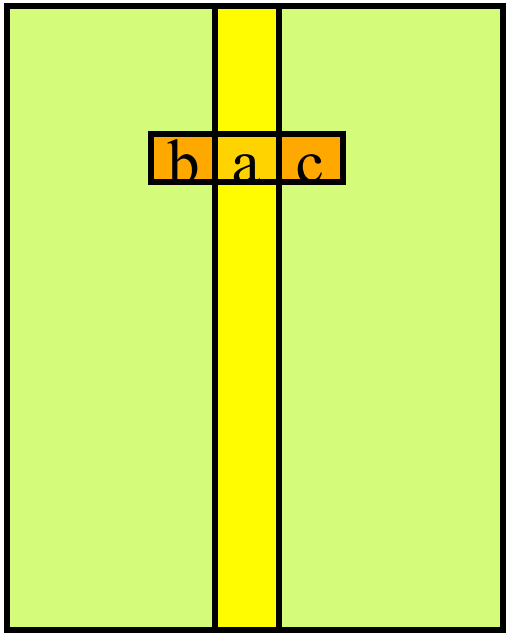
  …only O(n log m) !
  - Need O(log m) entries B[j]
  - Each computed in O(n) time

12	8	5
11	3	6
10	9	2
8	4	1

12	9	6

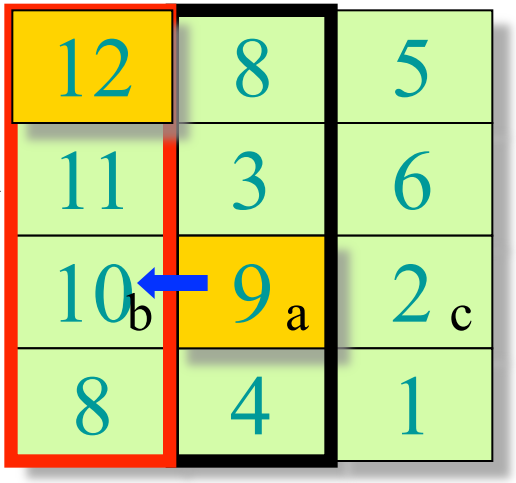
Algorithm II

- Pick middle column (j=m/2)
- Find *global* maximum a=A[i,m/2] in that column (and quit if m=1)
- Compare a to b=A[i,m/2-1] and c=A[i,m/2+1]
- If b>a
  then recurse on left columns
- Else, if c>a
  then recurse on right columns
- Else a is a 2D peak!



Algorithm II: Example

- Pick middle column (j=m/2)
- Find *global* maximum a=A[i,m/2] in that column (and quit if m=1)
- Compare a to b=A[i,m/2-1] and c=A[i,m/2+1]
- If b>a
  then recurse on left columns
- Else, if c>a
  then recurse on right columns
- Else a is a 2D peak!



Algorithm II: Correctness

Claim: If b>a, then there is a peak among the left columns

Proof (by contradiction):

Assume no peak on the left

Then b must have a neighbor b1 with higher value

And b1 must have a neighbor b2 with higher value

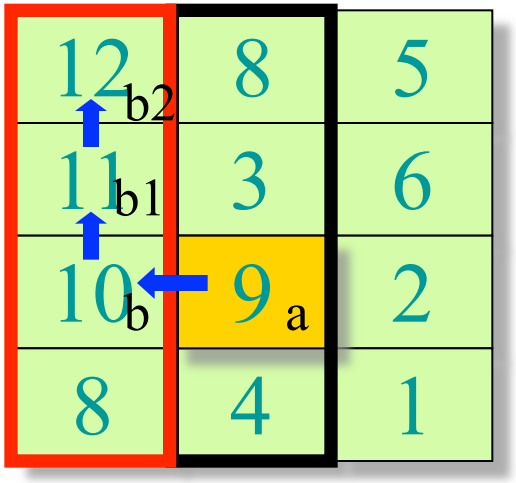
…

We have to stay on the left side – why?

(because we cannot enter the middle column)

But at some point, we would run out the elements of the left columns

Hence, we have to find a peak at some point.



Question: Does the above claim suffice for the proof of correctness of the algorithm?

Algorithm II: Complexity

- We have

Recursion

$$T(n,m) = T(n,m/2) + \Theta(n)$$

Scanning middle column

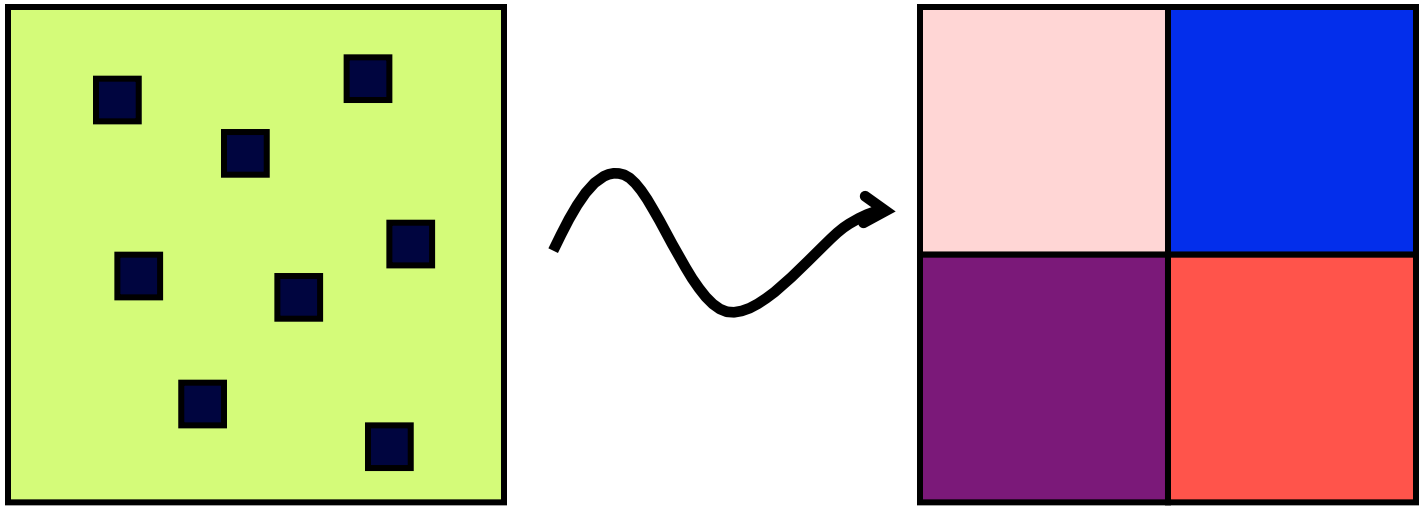
- Hence:
  - $T(n,n) = \underbrace{\Theta(n) + \Theta(n) + \ldots + \Theta(n)}_{\log_2 m} = \mathbf{\Theta(n \log m)}$

Faster than O(n log n) ?

- Idea:

  Reading only O($n + m$) elements, reduce an array of $n \times m$ candidates to an array of $n/2 \times m/2$ candidates

- Pictorially:



read only O($n + m$) elements

Faster than O(n log n) ?

- Hypothetical algorithm has recursion:

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

- Hence: $T(n,m) = \Theta(n+m) + \Theta\left(\frac{n+m}{2}\right)$

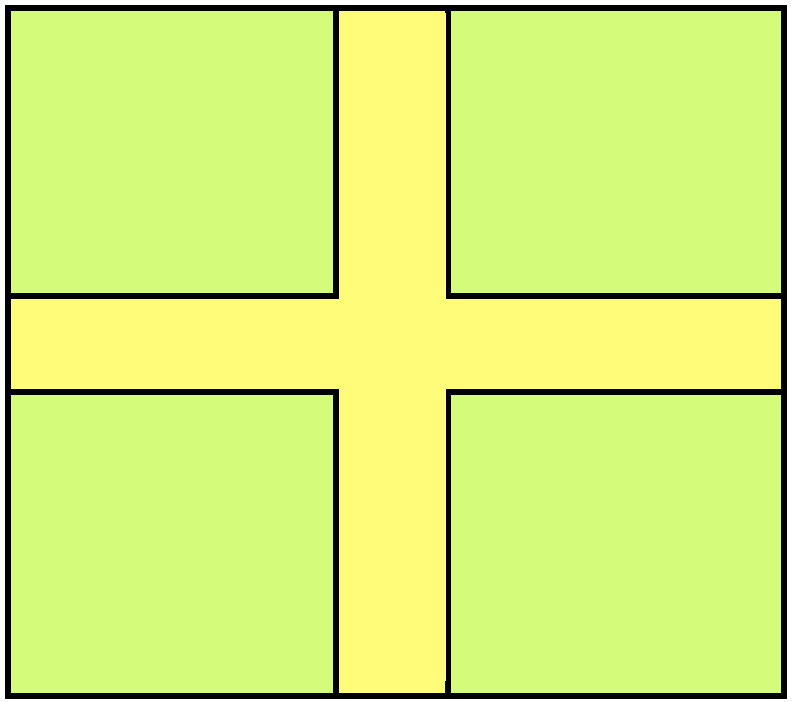
$$+\Theta\left(\frac{n+m}{4}\right)$$

$$+ \ldots + \Theta(1)$$

$$= \Theta(n+m) \quad !$$

Towards a linear-time algorithm

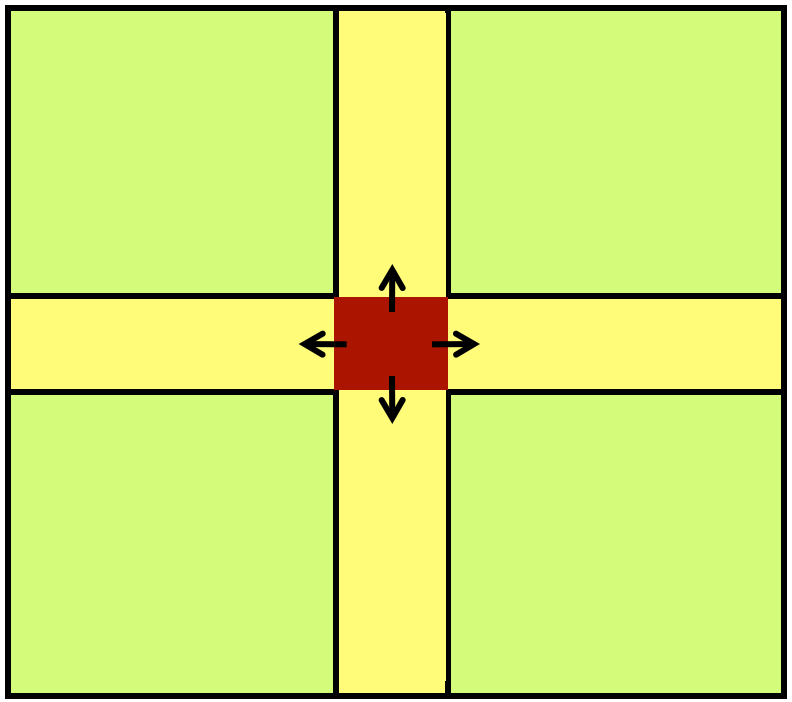
What elements are useful to check?



- suppose we find global max on the cross

Towards a linear-time algorithm

What elements are useful to check?

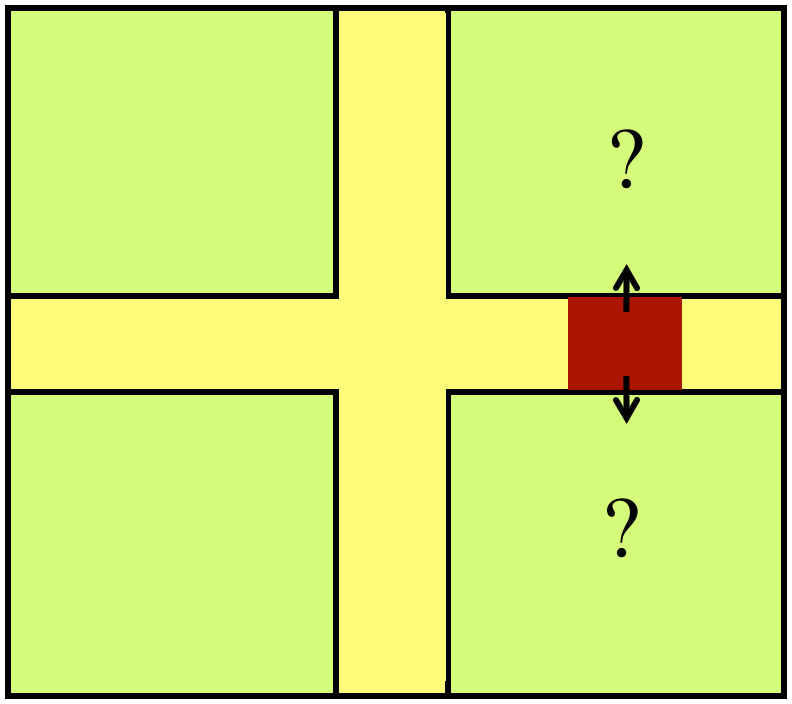


- suppose we find global max on the cross

- if middle element done!

Towards a linear-time algorithm

What elements are useful to check?



- find global max on the cross
- if middle element done!
- o.w. two candidate sub-squares
- determine which one to pick by looking at its neighbors not on the cross (as in Algorithm II)

Claim: The sub-square chosen by the above procedure (if any), always contains a peak of the large square.

OK, what else is needed for an O(n+m) algorithm?
Hmmm…

First Problem Set Out Today !

- Refer to class website for further information!
- Good Luck!
- I.e., GOOD WORK!