

R02

# Divide and Conquer Peak Finding

50.004 Introduction to Algorithm

Gemma Roig

(slides adapted from Dr. Simon LUI)

ISTD, SUTD

# Peak Finding

# Peak Finding Problem (PFP): 1D array

- Consider an array  $A[1..n]$  :

10	13	5	8	3	2	1
----	----	---	---	---	---	---

- An element  $A[i]$  is a *peak* if it is not smaller than all 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.

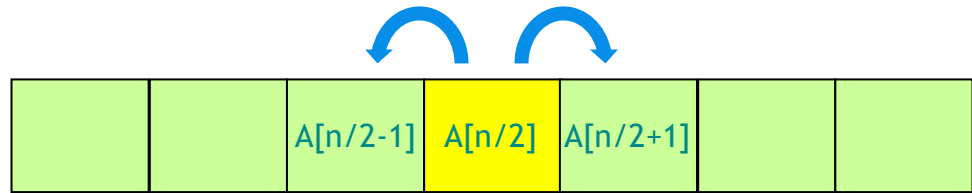
# Peak Finding

- Algorithm 1:
  - 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  $T(n)=\Theta(n)$

1	2	4	8	9	12	21
---	---	---	---	---	----	----



# Peak Finding



- Algorithm 2:
- Consider the middle element of the array and compare with neighbors
  - If  $A[n/2-1] > A[n/2]$   
then search for a peak among  $A[1] \dots A[n/2-1]$
  - Else, if  $A[n/2] < A[n/2+1]$   
then search for a peak among  $A[n/2+1] \dots A[n]$
  - Else  $A[n/2]$  is a peak!  
(since  $A[n/2-1] \leq A[n/2]$  and  $A[n/2] \geq A[n/2+1]$  )



# Algorithm II: Complexity

(worse case)

Time needed to find  
peak in array of length  $n$

Time for comparing  
 $A[n/2]$  with two  
neighbors

- We have

Recursion

$$T(n) = T(n/2) + O(1)$$

- Unraveling the recursion,

$$T(n) \leq T(n/2) + c \leq (T(n/2^2) + c) + c \leq \dots$$

# Algorithm II: Complexity

(worse case)

Time needed to find peak  
in array of length  $n$

Time for comparing  
 $A[n/2]$  with two  
neighbors

- We have

$$T(n) = T(n/2) + O(1)$$

- Unraveling the recursion,

$$T(n) \leq T(n/2) + c \leq (T(n/2^2) + c) + c \leq \dots$$

$$\leq T(\underbrace{n/2^{\log_2 n}}_{=1}) + \underbrace{c + c + \dots + c}_{\log_2 n} = O(\log n)$$

- $\log n$  is much much better than  $n$  !

# In class exercise

- Solve this with the Master Theorem

$$T(n) = T(n/2) + O(1)$$

$$1) \text{ if } f(n) = O(L^{1-\varepsilon}) = O(n^{\log_b a - \varepsilon})$$

$$\Rightarrow T(n) = \Theta(L) = \Theta(n^{\log_b a})$$

$$2) \text{ if } f(n) = \Theta(L) = \Theta(n^{\log_b a})$$

$$\Rightarrow T(n) = \Theta(L \log n) = \Theta(n^{\log_b a} \log n)$$

$$3) \text{ if } f(n) = \Omega(L^{1+\varepsilon}) = \Omega(n^{\log_b a + \varepsilon})$$

$$\Rightarrow T(n) = \Theta(f(n))$$



# Divide and Conquer

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

# Peak finding: 2D

- Consider a square 2D array  $A[1\dots n, 1\dots n]$  :

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

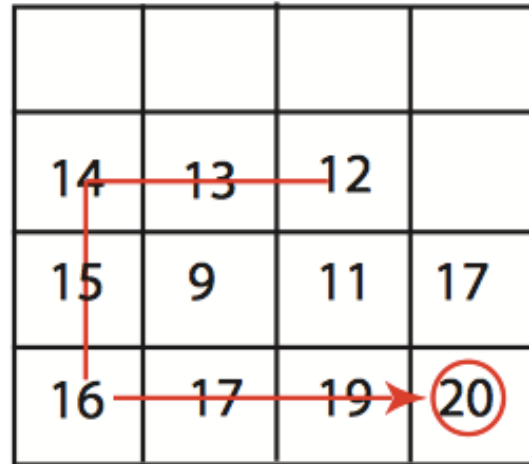
Size of problem =  $n$   
= # of rows or columns

- An element  $A[i]$  is a *2D peak* if it is not smaller than its (at most 4) neighbors.
- Problem: find *any* 2D peak.

# 2D Peak finding: Ideas?

- Algorithm 1: **brute-force** method (i.e. search for all square)

14	13	12	
15	9	11	17
16	17	19	20



- Complexity =  $\Theta(n^2)$

# Algorithm 2: use the 1D algorithm

- Algorithm 2:
  - (a) For each column  $j$ , find its *global* maximum  $B[j]$
  - (b) Apply 1D peak finder to find a peak of  $B[1...n]$  (say  $B[j]$ )
- Running time ?  
...is  $\Theta(n^2)$

- Proof of Algorithm Correctness (an informal way)
  - 12 is vertically the largest (trivial)
  - 12 is horizontally the largest, because
    - 12 is larger than 9 and 6
    - 9 and 6 are larger than all the elements in their column

$A[][]$  

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

$B[]$

12	9	6
----	---	---

# Algorithm 3: be “lazy” in the 1D algorithm

- Modify Algorithm 2.
- Use the 1D algorithm (P.5) to find the maximum in  $B[]$
- Total time become  $O(n \log n)$  !
  - We only need  $O(\log n)$  entries of  $B[j]$
  - Each  $B[j]$  can be computed in  $O(n)$

$A[][]$

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

$B[]$

12	9	6
----	---	---

# Exercise

- Find the complexity of the following formulas, using Master theorem

$$1. \ T(n) = 3T(n/2) + n^2$$

$$2. \ T(n) = 4T(n/2) + n^2$$

$$3. \ T(n) = T(n/2) + 2^n$$