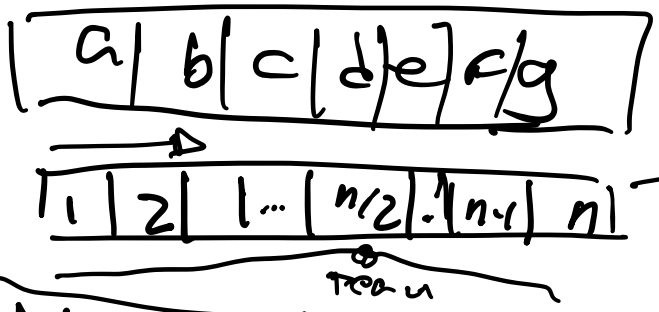
Peak finding

Pos 2 is peak if

$b \geq a \quad b \geq c$

Find the peak if it exist



$O(n)$

start from left
worst case n-1 loop through all cases

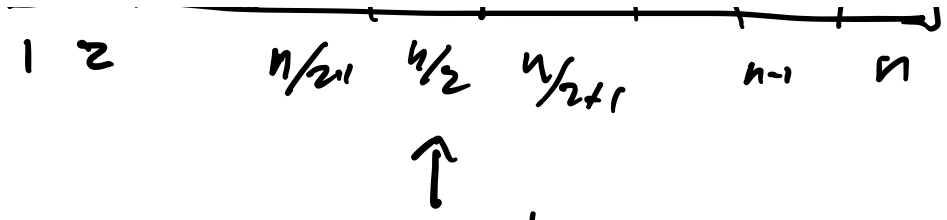
Algorithm thinking

- Start at some point
- Create generic solutions that allow us to solve more complex problems

Why study this

- to handle effectively big amount of data

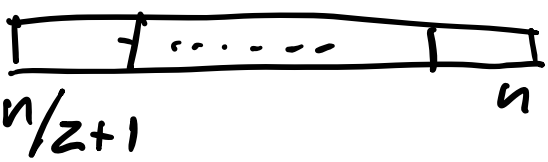
Divide and Conquer



look at n/2 pos

- If $a[n/2] < a[n/2-1]$ then look at the left half $(1 \ldots n/2-1)$ to find the peak

- else, look for the peak on the other half



- Otherwise, stop, you found the peak

Note: this is a recursive algorithm

$T(n)$ = work algorithm does on a set of data of size n

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

Best case $= T(1) = O(1)$

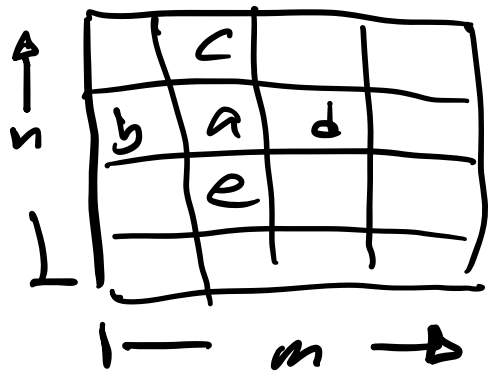
$T(n) = O(1) + \cdots + O(1)$

worst case $= T(n) = O(\log_2 n)$

Excersice:

Code both algorithms and Benchmark how long both take

2D Version

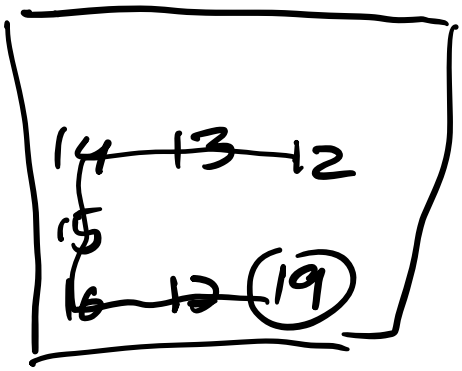


a is a 2D Peak If

- $a \geq b$; $a \geq d$
- $a \geq c$; $a \geq e$

Greede Ascent algorithm

↳ Finds a direction to search for the peak (decides where to start)

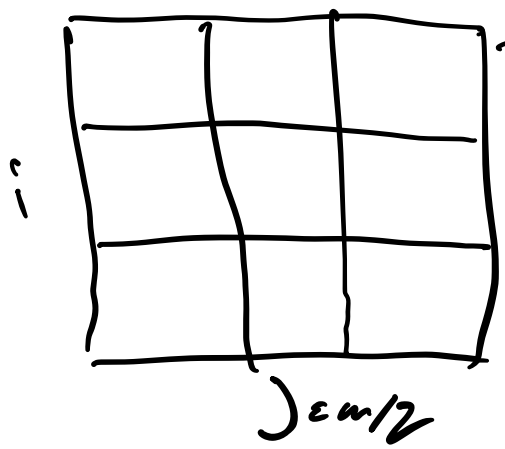


→ Find the largest number & the left/right doesn't go from the starting point

worst Case

$O(nm) = O(n^2)$ if $m = n$

Divide and conquer in 2D



- Pick a middle col $j = m/2$
- Find global max in col j at (i, j)
- Compare $(i, j-1)$ - (i, j) $(i, j+1)$

Pick left col if
$(i, j-1) > (i,)$

If $(i, j) \geq (i, j-1), (i, j+1)$

Then we found a peak

Complexity

$$T(n, m) = T(n, m/2) + O(n)$$

$$T(n, m) = O(n) + \cdots (O(n)$$

$$\log_2 m$$

$$= O(n \log_2 m)$$