MT1: FFTs + DP
MT2: Randomized + Flows
>MT2: Flows + LP + Approx

Conflict Dec 16 8-11 am

Given a 2D array
Find smallest entry in each row

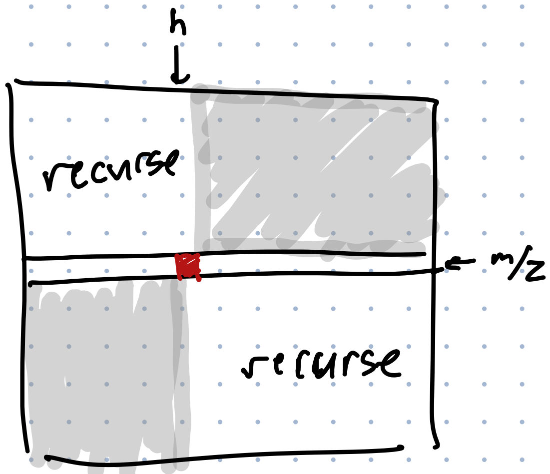
O(mn) is optimal in general

12	21	38	76	27
74	**14**	14	29	60
21	**8**	25	10	71
68	45	29	**15**	76
97	8	12	**2**	6

Monotone =
leftmost min elements in earlier ~~columns~~ rows above/left of leftmost min elems of later rows

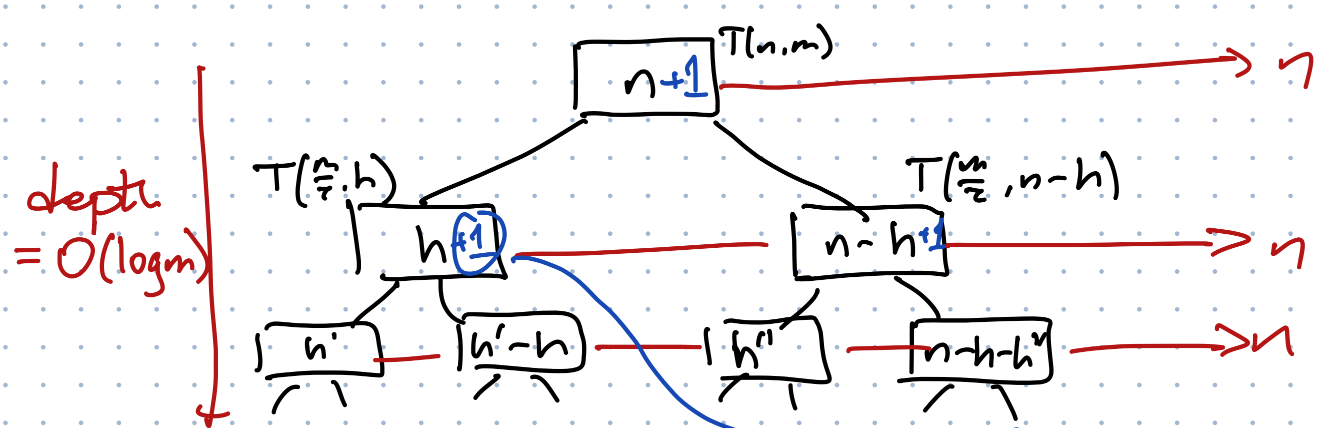
Top-down D+C:

Find min of middle row



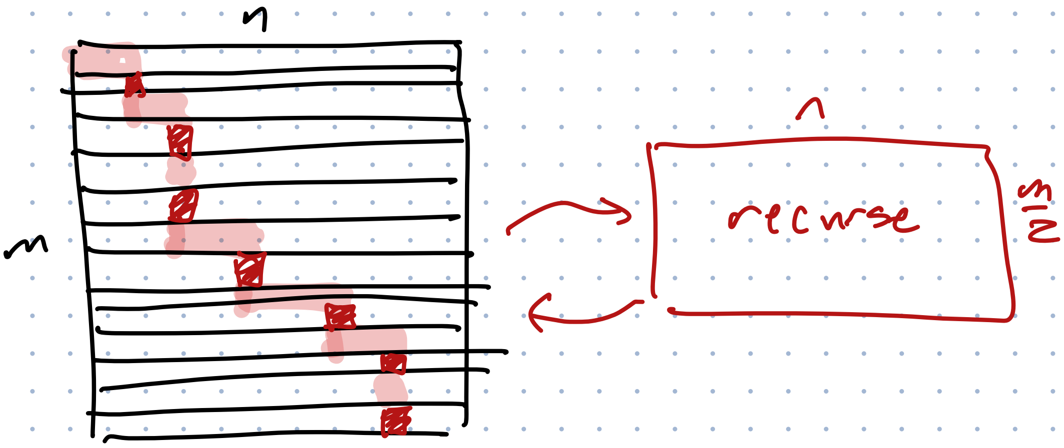
rows cols

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



$$\boxed{O(n\log m + m)}$$

Bottom-up:



Search $O(m+n)$ cells

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

$$= O(m + n\log m)$$

Can we get rid of this?

Totally monotone

= Every 2x2 subarray is monotone

NOT TM

$$\begin{bmatrix} 12 & 21 & 38 & 76 & 27 \\ 74 & 14 & 14 & 29 & 60 \\ 21 & 8 & 25 & 10 & 71 \\ 68 & 45 & 29 & 15 & 76 \\ 97 & 8 & 12 & 2 & 6 \end{bmatrix}$$

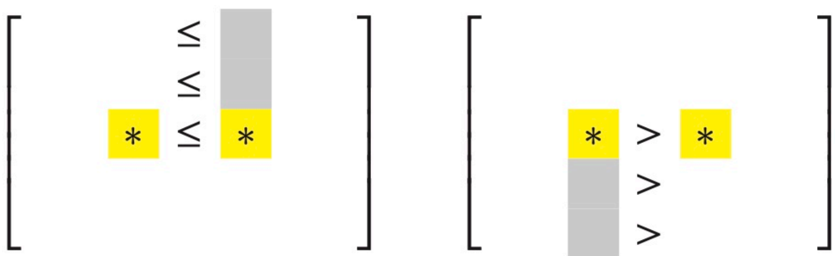
TM!

$$\begin{bmatrix} 12 & 21 & 38 & 76 & 89 \\ 47 & 14 & 14 & 29 & 60 \\ 21 & 8 & 20 & 10 & 71 \\ 68 & 16 & 29 & 15 & 76 \\ 97 & 8 & 12 & 2 & 6 \end{bmatrix}$$

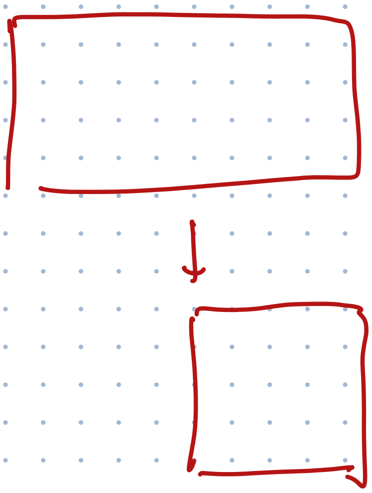
Shor
Moran
Aggarwal
Wilber
Klawe

"SMAWK"

One comparison "kills" lots of entries in one column



Assume array is wide $\iff n < m$ ⟶ reduce to $m \times m$ array



```
REDUCE(M[1..m, 1..n]):
  t ← 1
  S[t] ← 1
  for k ← 1 to n
      while t > 0 and M[t, S[t]] ≥ M[t, k]
          t ← t − 1          ⟨⟨pop⟩⟩
      if t < m
          t ← t + 1
          S[t] ← k           ⟨⟨push k⟩⟩
  return S[1..t]
```

?

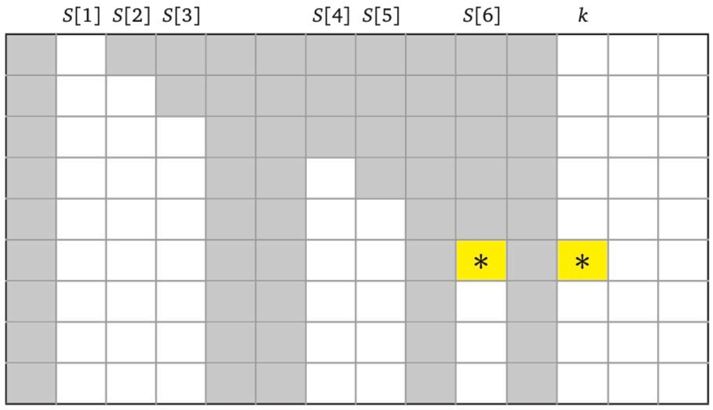
O(n) time

Figure D.10. The SMAWK algorithm to REDUCE wide arrays

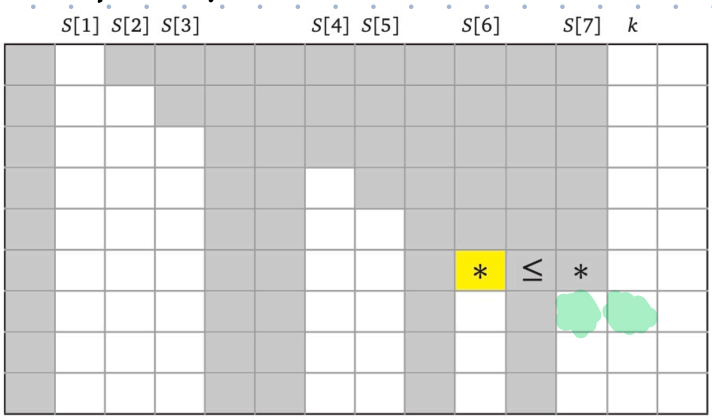
$S[1..t]$ = stack of column indices sorted in increasing order

— For all $1 \le j \le t$ top $j-1$ elements in column j are dead

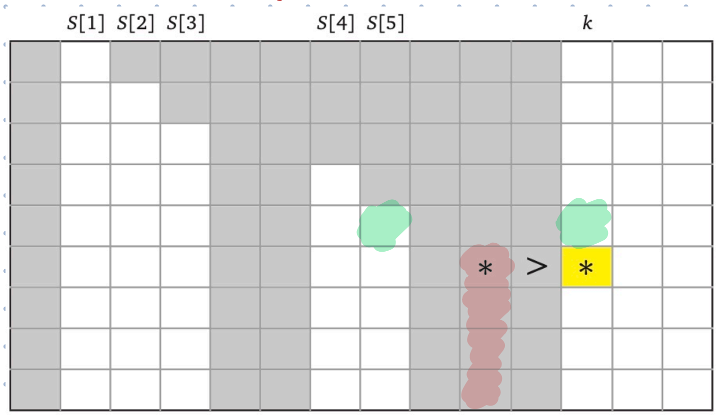
— If $j < k$ and j is not in S, column j is dead



push k, $k \leftarrow k+1$

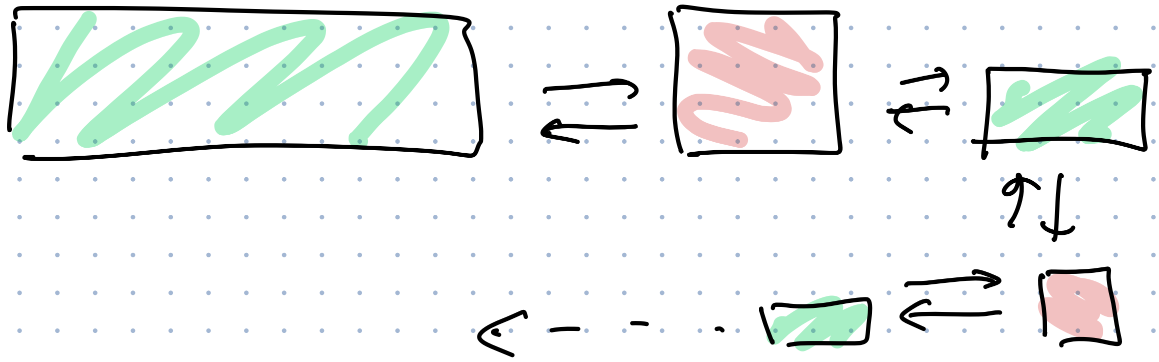


$t \leftarrow t-1$



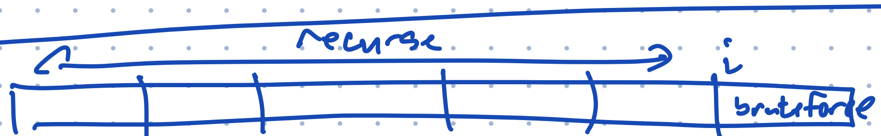
Find all row-minima in an $m \times n$ totally monotone array:

1. if $m < 10^{100}$ $\longrightarrow$ brute force $O(n)$ time
2. if $m < n$ $\longrightarrow$ reduce to $m \times m$ array in $O(n)$ time + recurse
3. if $m \geq n$ $\longrightarrow$ filter (every other row)
   recurse
   fill in odd rows $O(n+m)$ time



$$T(m,n) = \begin{cases} O(1) & \text{if } m \le O(1) \\ O(n) + T(m,m) & \text{if } m < n \\ O(m\cancel{n}) + T(\frac{n}{2}, m) & \text{if } m > n \end{cases}$$

$$\boxed{= O(m+n)}$$



$A[1..n]$

partition A into K segments/intervals $I_1 \ldots I_K$

$$\text{cost} = \sum_j \big(\underbrace{\textstyle\sum \text{elements in interval } I_j}\big)^2$$

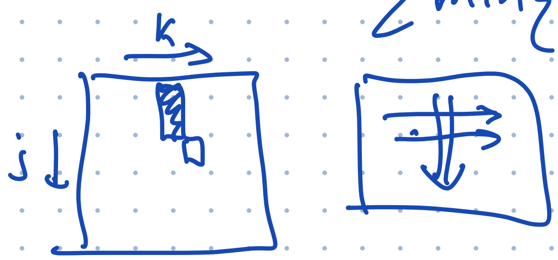
Indices $I[1..K]$

$$\Big(\sum_{i=I[j-1]+1}^{I[j]} A[i]\Big)^2$$

$$\text{cost}(i,j) = \sum_{k=i}^{j} A[k] \qquad \text{cost}(I[j-1]+1, I[j])^2$$

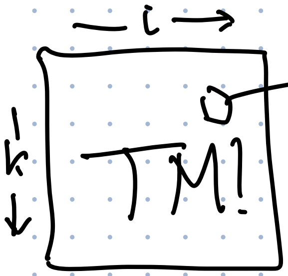
OptCost(j,k) = optimal cost of splitting $A[1..j]$ into k intervals

$$\text{OptCost}(j,k) = \Big\{ \min\Big\{ \boxed{\text{cost}(i,j)^2 + \text{OptCost}(i,k)} \;\Big|\; 1 \le i \le j \Big\}$$



For $j \leftarrow 1$ to n
    For $k \leftarrow 0$ to K
        For $i \leftarrow 1$ to j

$O(n^2K)$ time



$cost(i,j)^2 + OptCost(i, k-1))$

with SMAWK $\rightarrow$ $\boxed{O(nK) \text{ time}}$