CS2020
Data Structures and Algorithms

Welcome!

Administrativia

Discussion Groups:

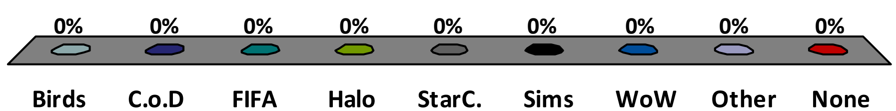
- Close to finalized.
- Wed. 4-6pm MOVING 2-4pm
- Talk to me if there are problems.

Problem Sets:

- #1: Due Thursday.
- #2: Released today.

What is your favorite video game?

1. Angry Birds
2. Call of Duty
3. FIFA
4. Halo
5. Starcraft
6. The Sims
7. World of Warcraft
8. Other
9. I don’t play video games.



Today: Divide and Conquer!

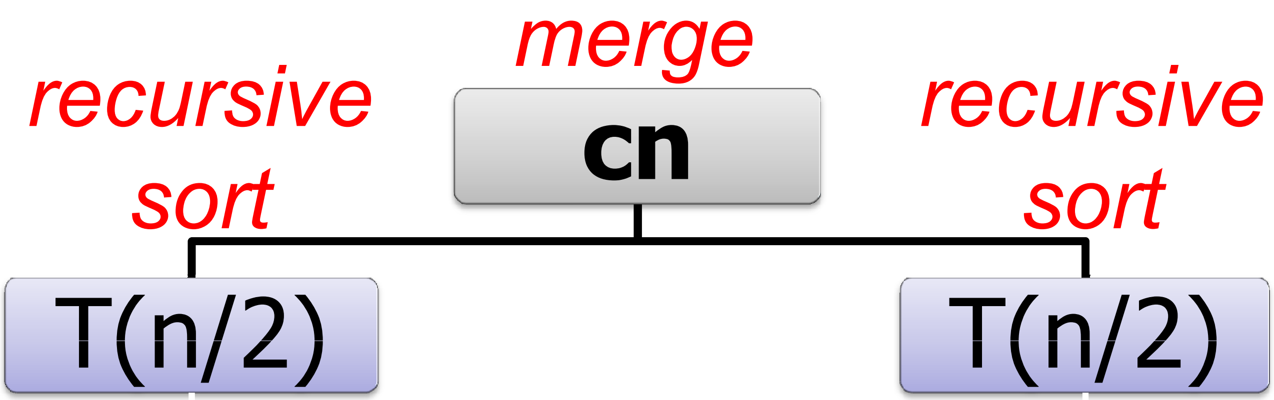
Peak Finding

- 1-dimension
- 2-dimensions

A few other examples?

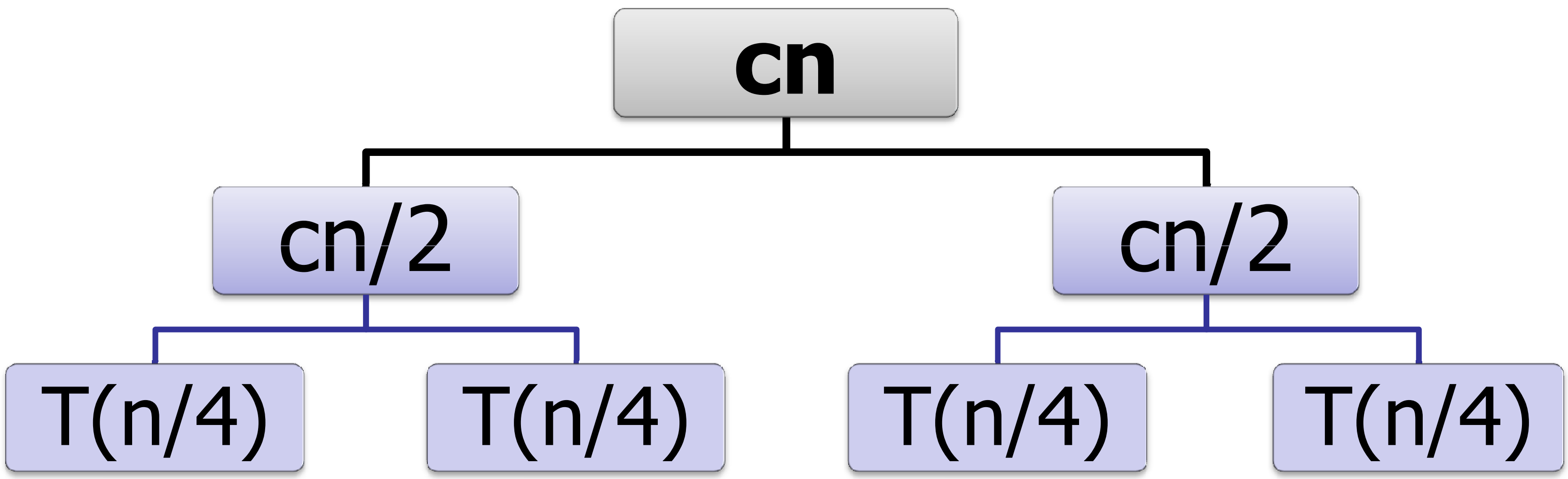
Merge-Sort Analysis (Review)

$$T(n) = 2T(n/2) + cn$$



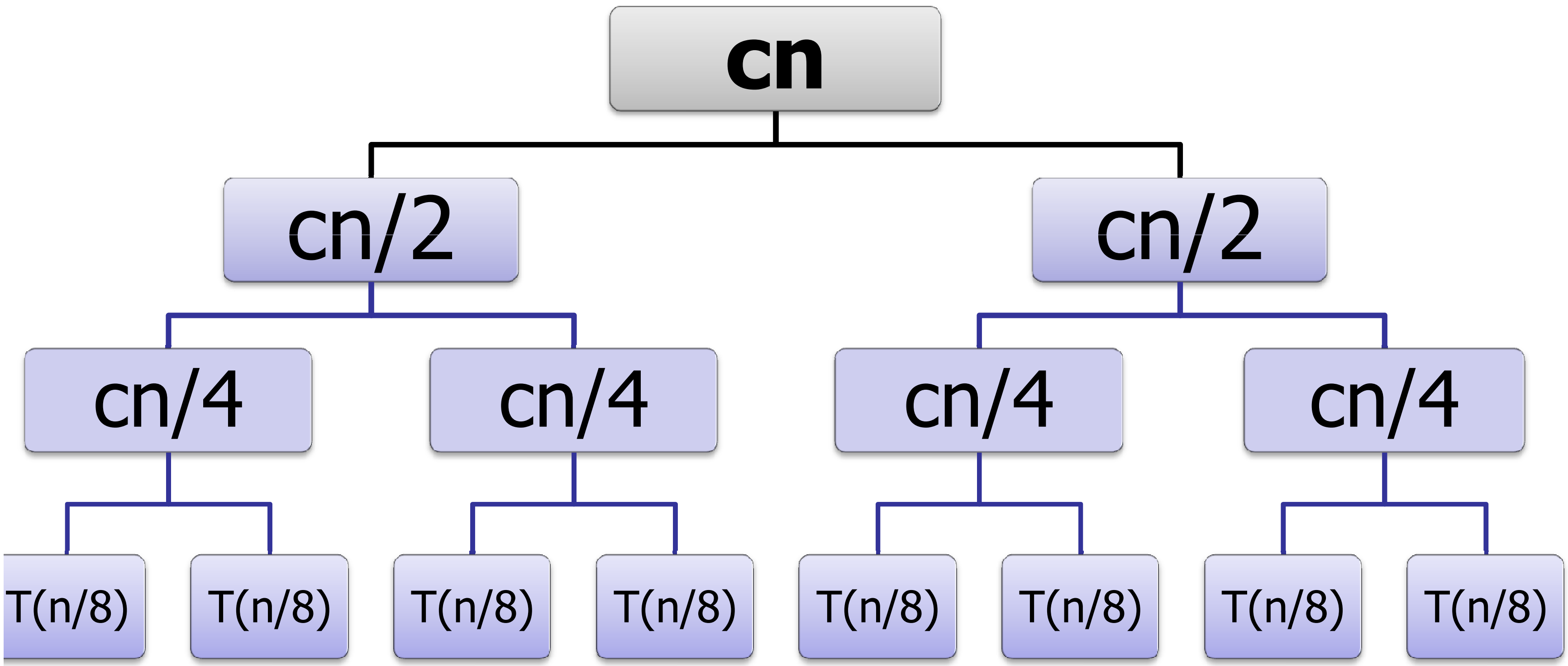
Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$



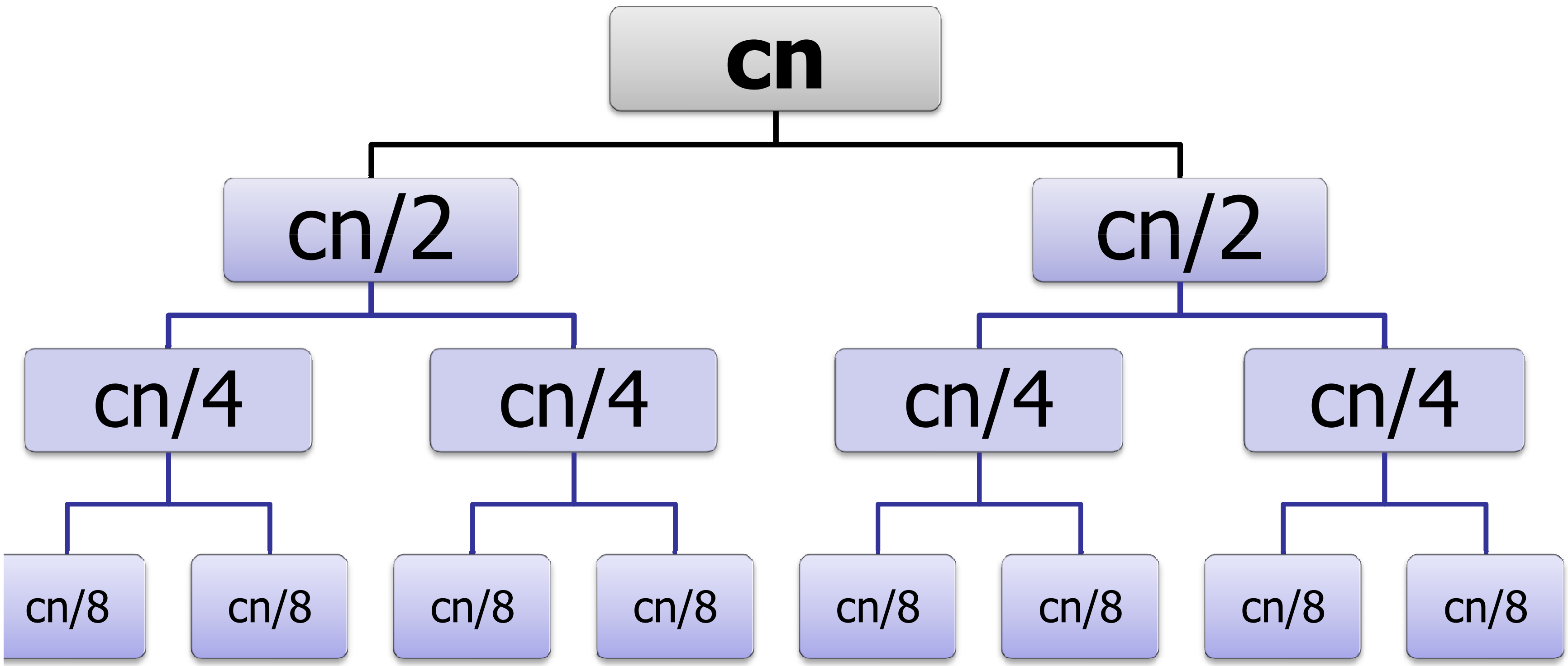
Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$



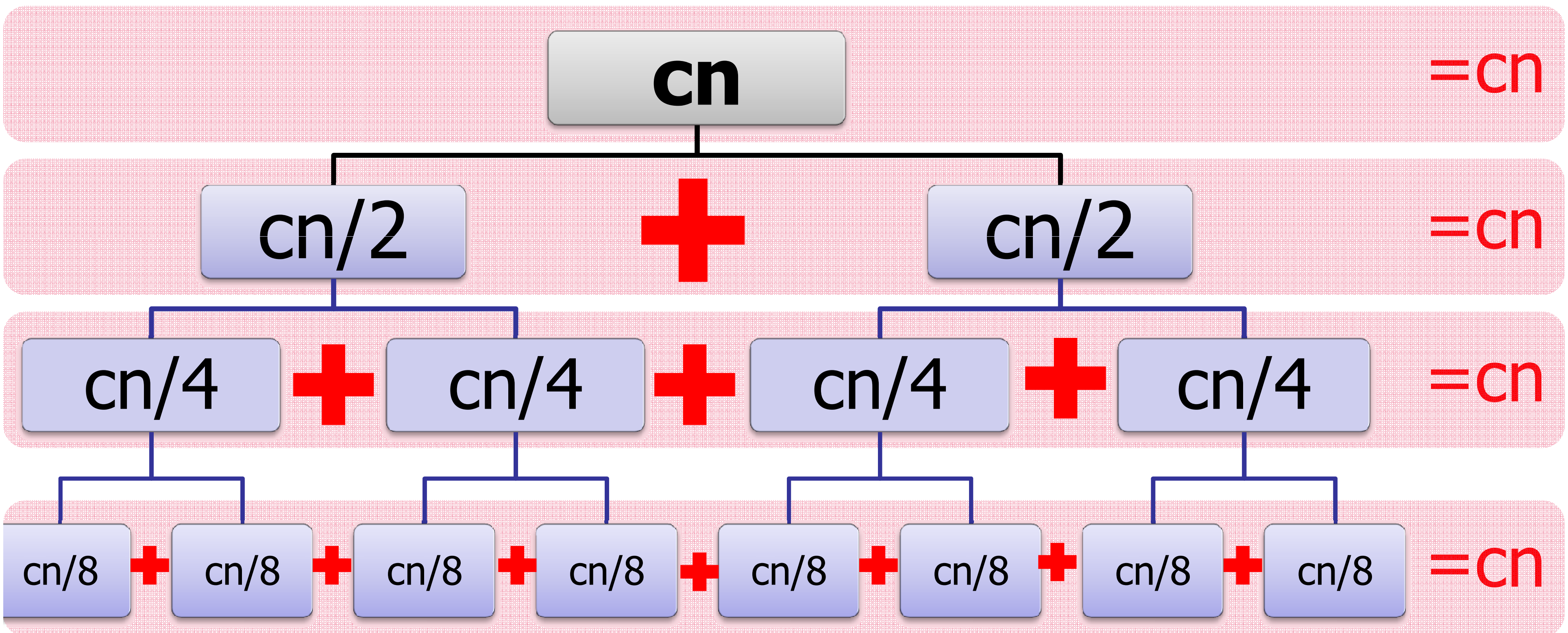
Merge-Sort Analysis(Review)

$T(n) = 2T(n/2) + cn$



Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$



cn log n

Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$

Level	Number
0	1
1	2
2	4
3	8
4	16
…	…
h	??

$$\text{Number} = 2^{\text{Level}}$$

Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$

Level	Number
0	1
1	2
2	4
3	8
4	16
…	…
h	n

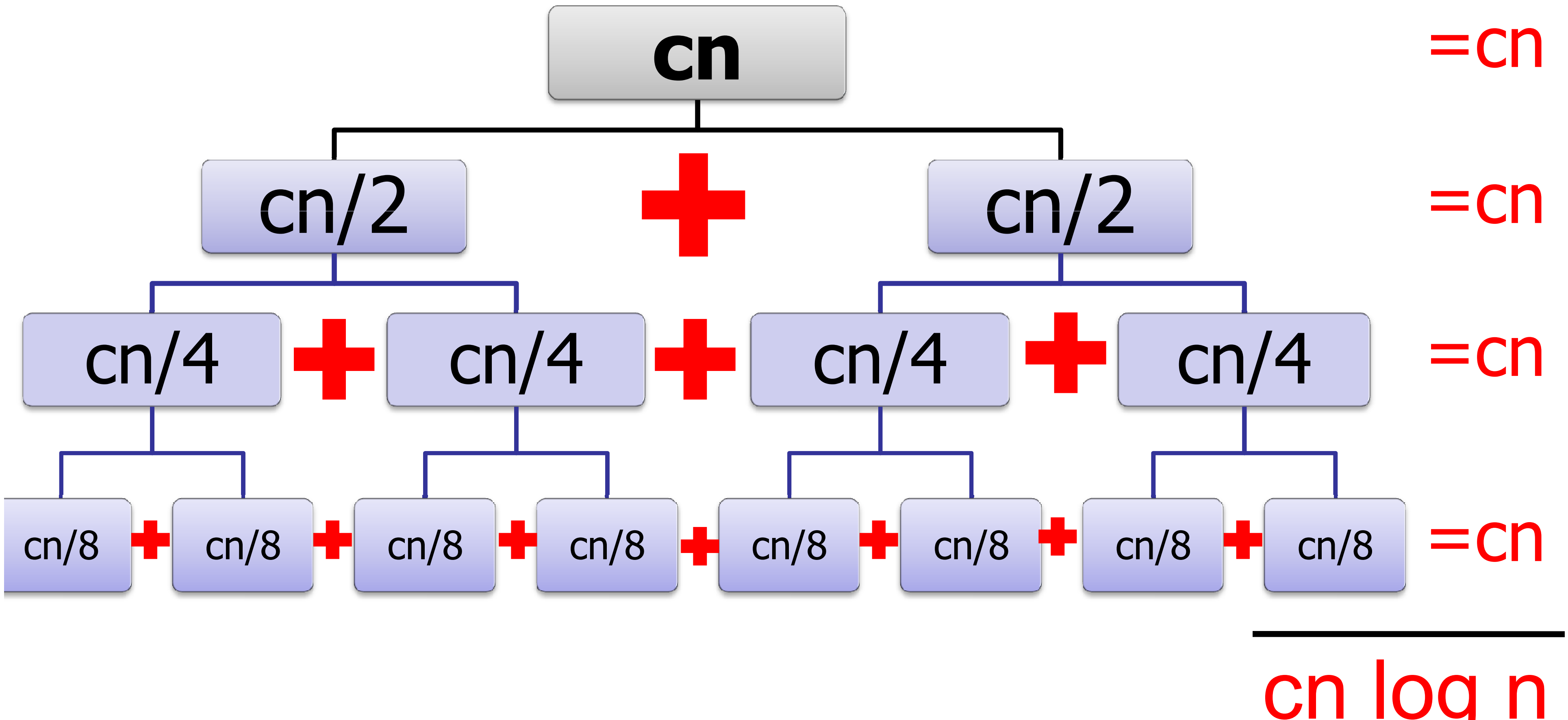
$$\text{Number} = 2^{\text{Level}}$$

$$n = 2^h$$

$$\log n = h$$

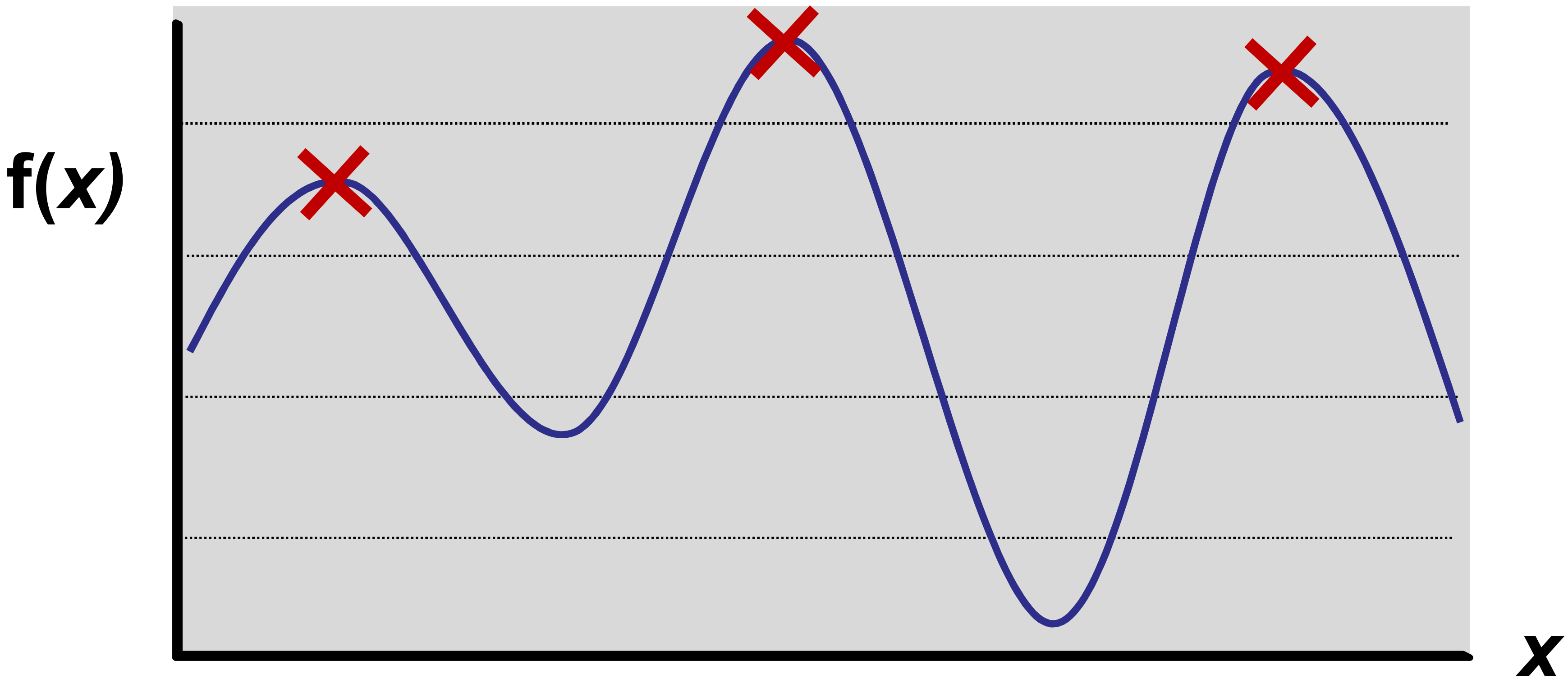
Merge-Sort Analysis (Review)

$T(n) = 2T(n/2) + cn$



Peak Finding

Input: Some function f(x)



Output: A local maximum (or minimum)

Peak Finding

Optimization problems:

- Find a good solution to a problem.
- Find a design that uses less energy.
- Find a way to make more money.
- Find a good scenic viewpoint.
- Etc.

Why local maximum?

- Finds a *good enough* solution.
- Local maxima are close to the global maximum?
- Much, much faster.

Global Maximum

Input: Array A[1..n]

Output: maximum element in A

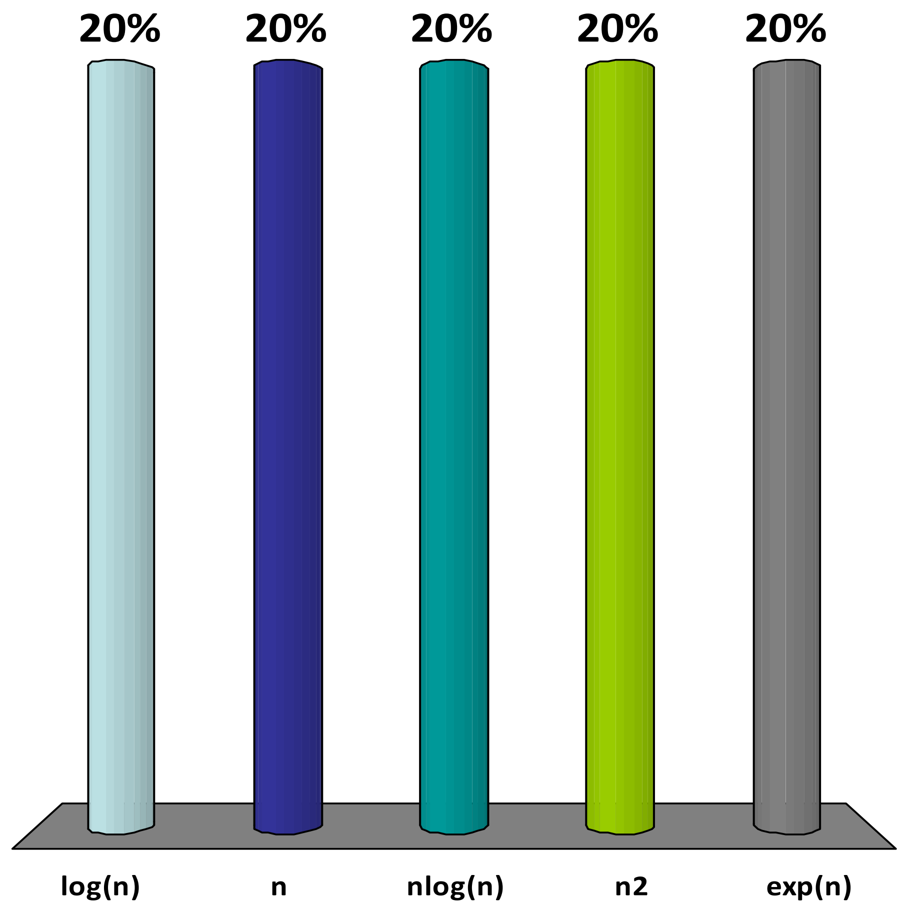
How long to find a global maximum?

Input: Array A[1..*n*]

Output: maximum element in A

1. O(log n)
2. O(n)
3. O(n log n)
4. $O(n^2)$
5. $O(2^n)$

0 of 60



Global Maximum

Unsorted array: A[1..n]

7	4	9	2	11	6	23	4	28	8	17	5

FindMax(A,n)
    max = A[1]
    for i = 1 **to** n **do**:
        if (A[i] > max) **then** max = A[i]

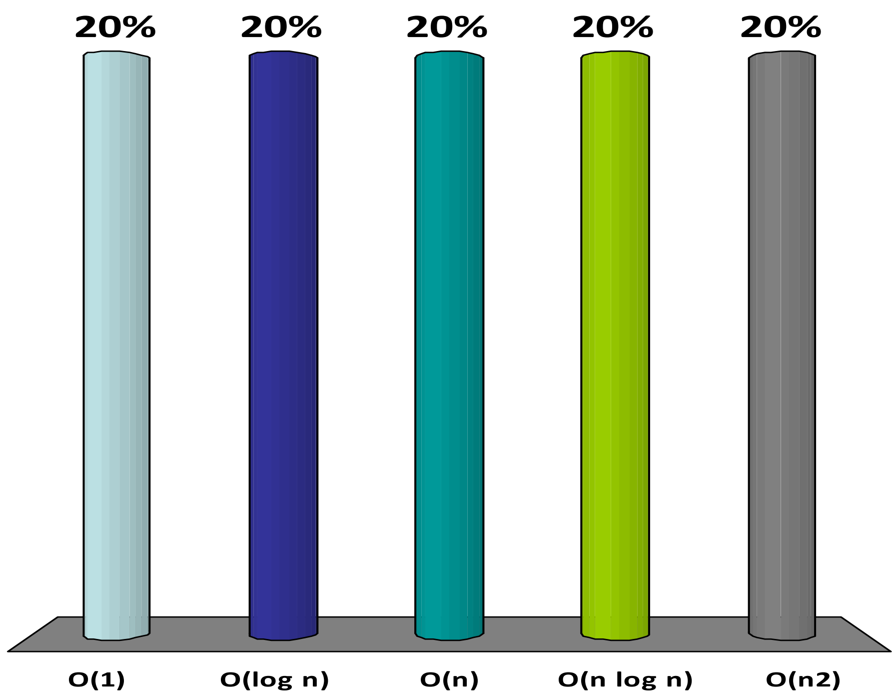
Time Complexity: ??

Global Maximum

Sorted array: A[1..n]

How long to find the maximum?

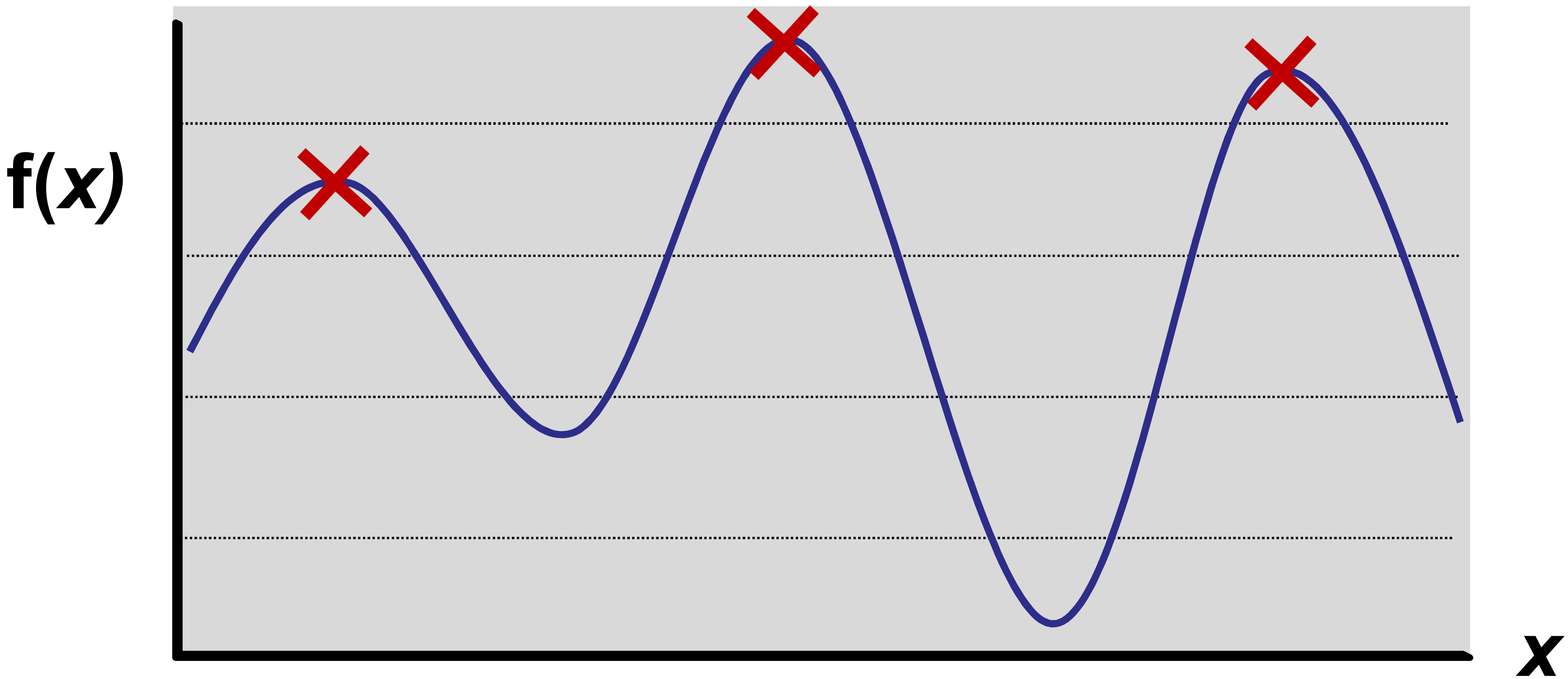
1. O(1)
2. O(log n)
3. O(n)
4. O(n log n)
5. O(n^2)



0 of 60

Peak Finding

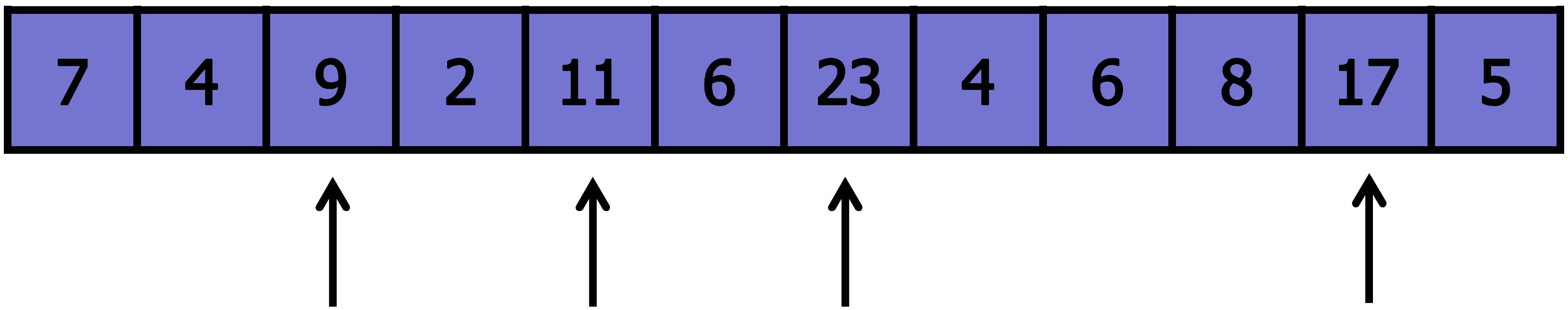
Input: Some function f(x)



Output: A local maximum

Peak Finding

Input: Some ~~function~~ array A[1..n]

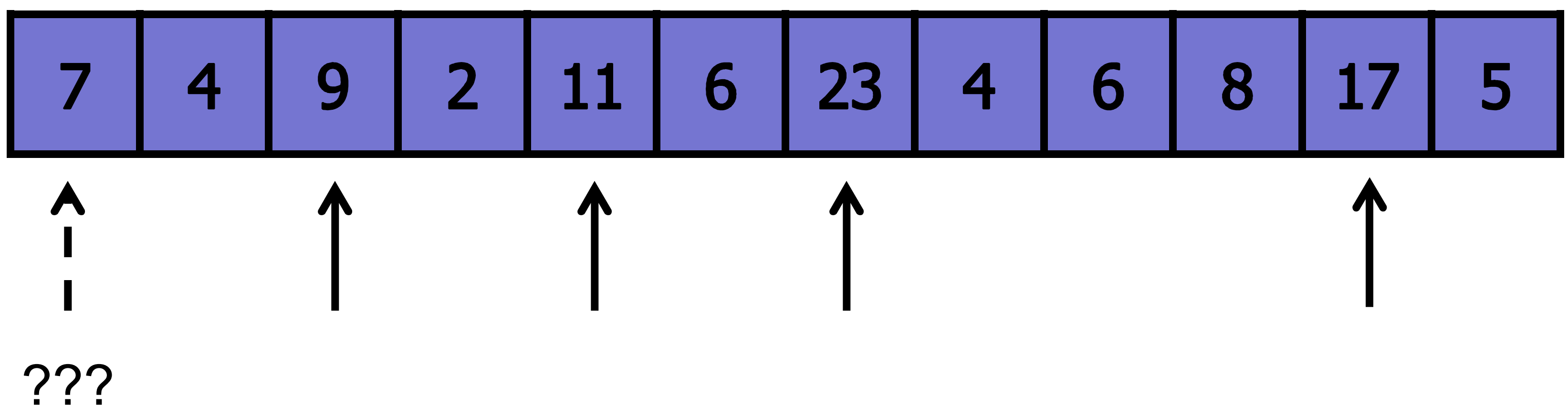


Output: a local maximum in A

$$A[i-1] \leq A[i] \quad \textbf{and} \quad A[i+1] \leq A[i]$$

Peak Finding

Input: Some ~~function~~ array A[1..n]

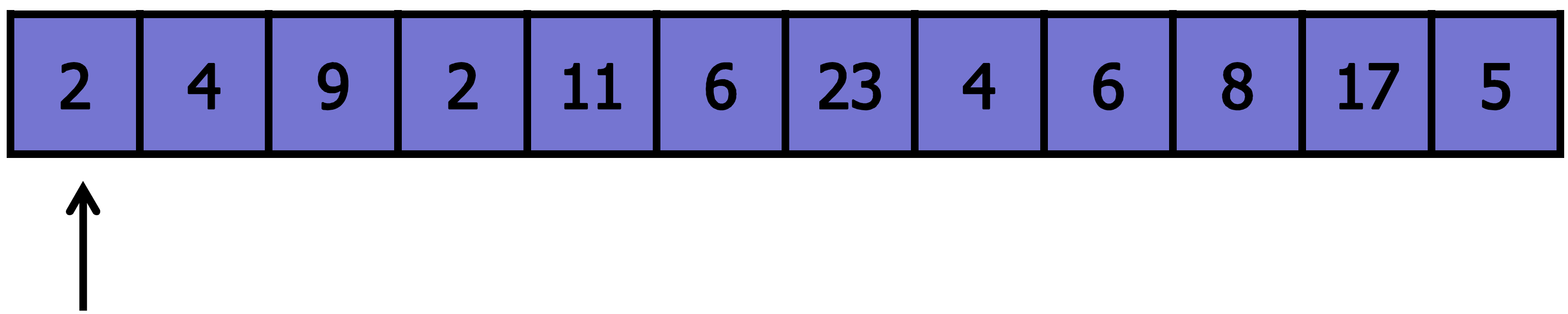


Output: a local maximum in A

$$A[i-1] \le A[i] \quad \textbf{and} \quad A[i+1] \le A[i]$$

Peak Finding: Algorithm 1

Input: Some array A[1..n]



FindPeak

- Start from A[1]
- Examine every element
- Stop when you find a peak.

Peak Finding: Algorithm 1

Input: Some array A[1..n]

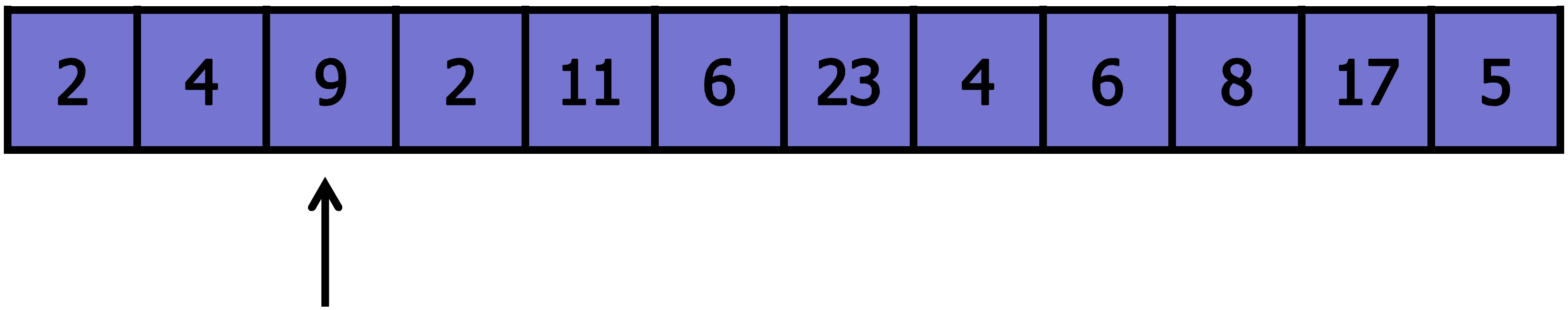
2	4	9	2	11	6	23	4	6	8	17	5

FindPeak

- Start from A[1]
- Examine every element
- Stop when you find a peak.

Peak Finding: Algorithm 1

Input: Some array A[1..n]

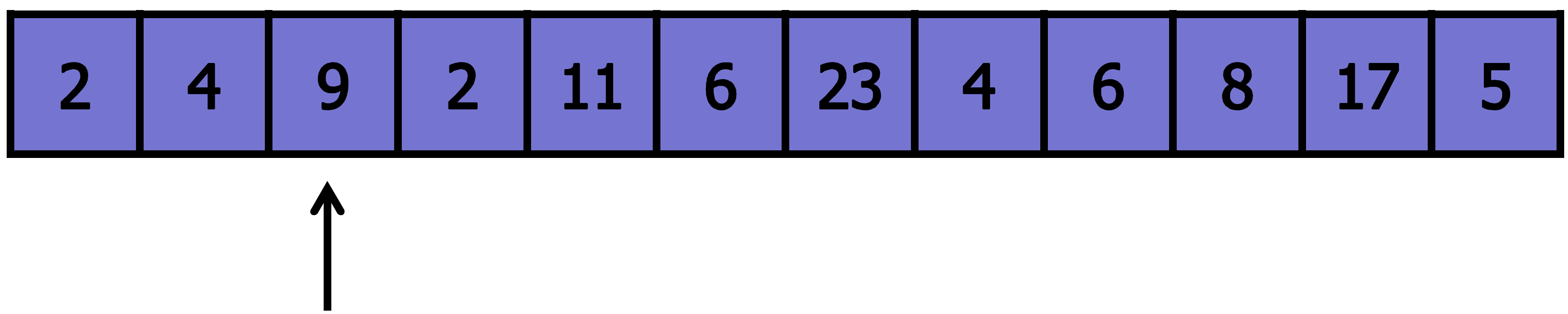


FindPeak

- Start from A[1]
- Examine every element
- Stop when you find a peak.

Peak Finding: Algorithm 1

Input: Some array $A[1..n]$



Running time: n

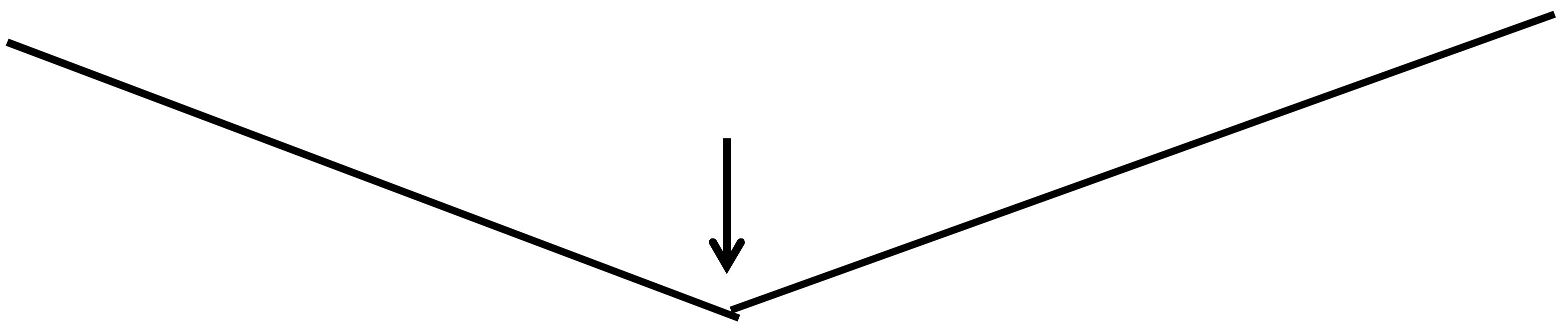
Simple improvement?

Peak Finding: Algorithm 1

Input: Some array A[1..n]

2	4	9	2	11	6	23	4	6	8	17	5

Start in the middle!



Worst-case: n/2

Peak Finding: Algorithm 2

Divide-and-Conquer

2	4	9	2	5	6	23	4	6	8	17	5

Start in the middle

5 < 6? **OK**

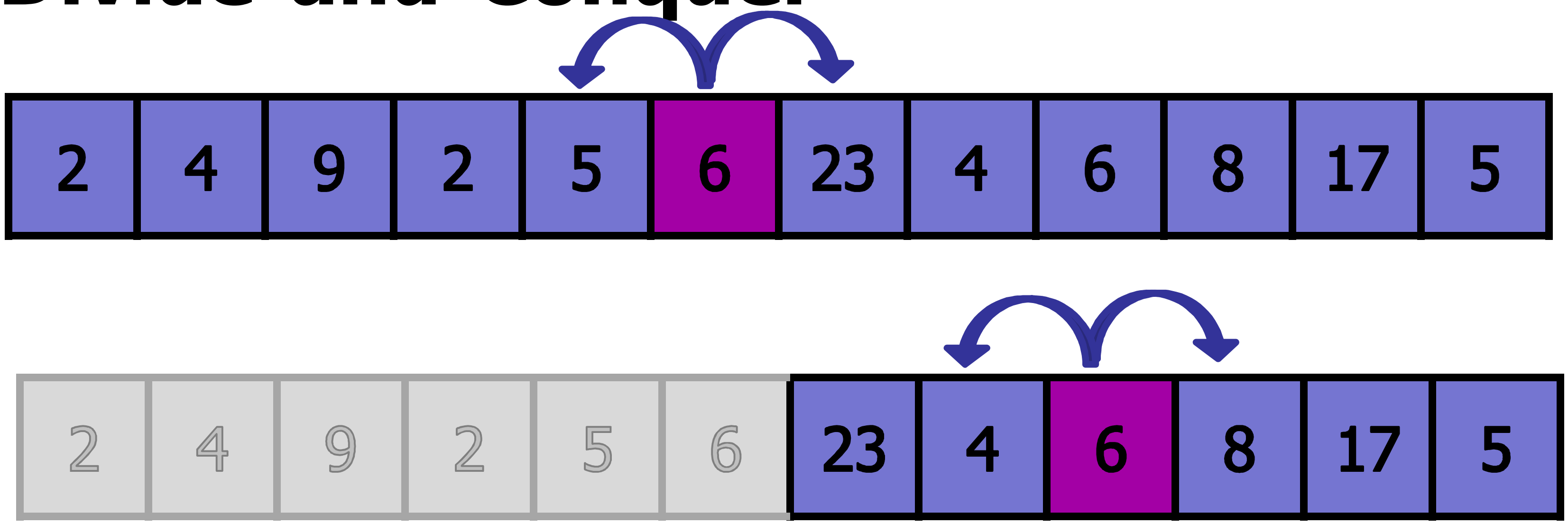
23 < 6? **NO**

Recurse!

2	4	9	2	5	6	23	4	6	8	17	5

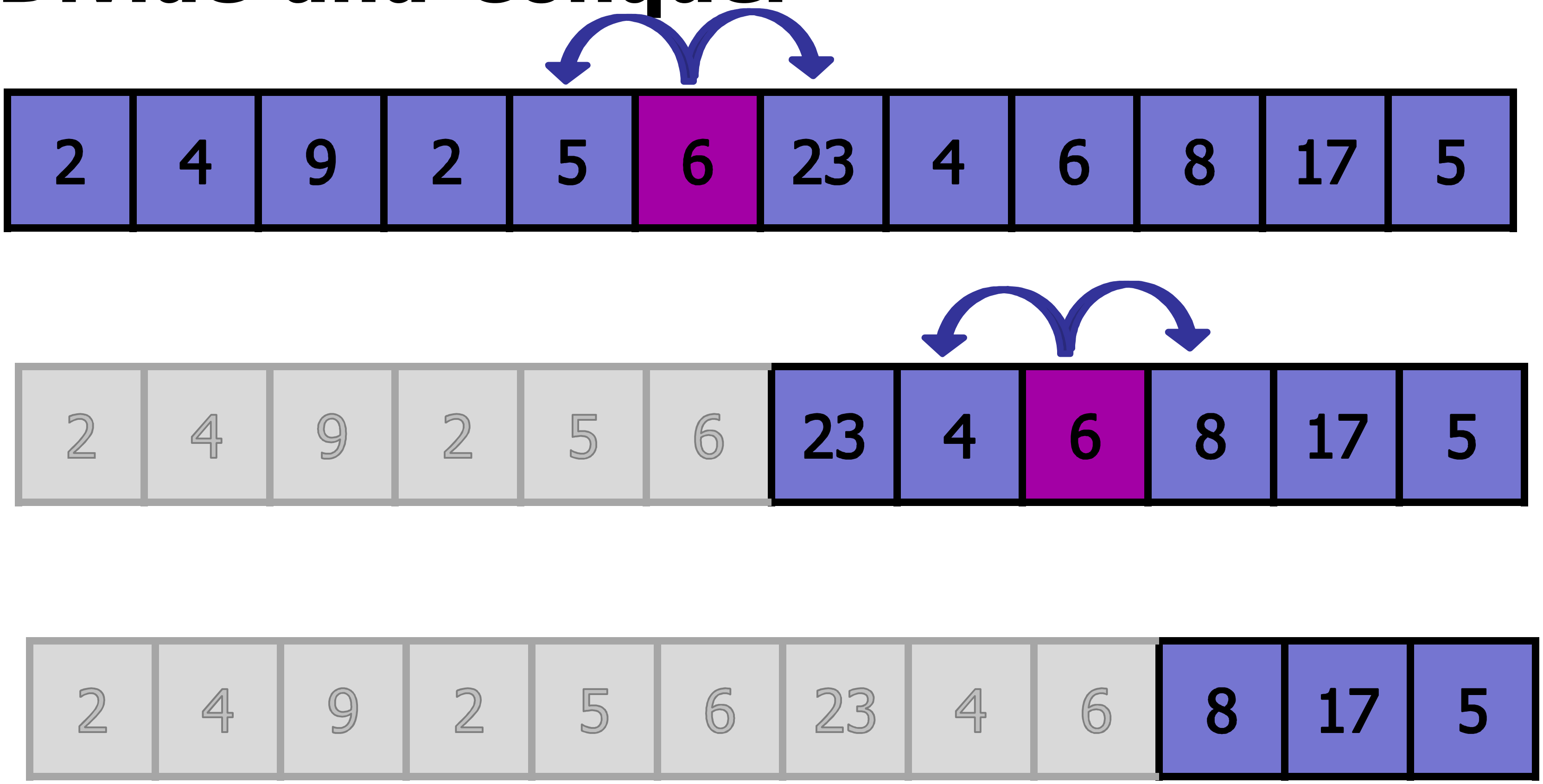
Peak Finding: Algorithm 2

Divide-and-Conquer



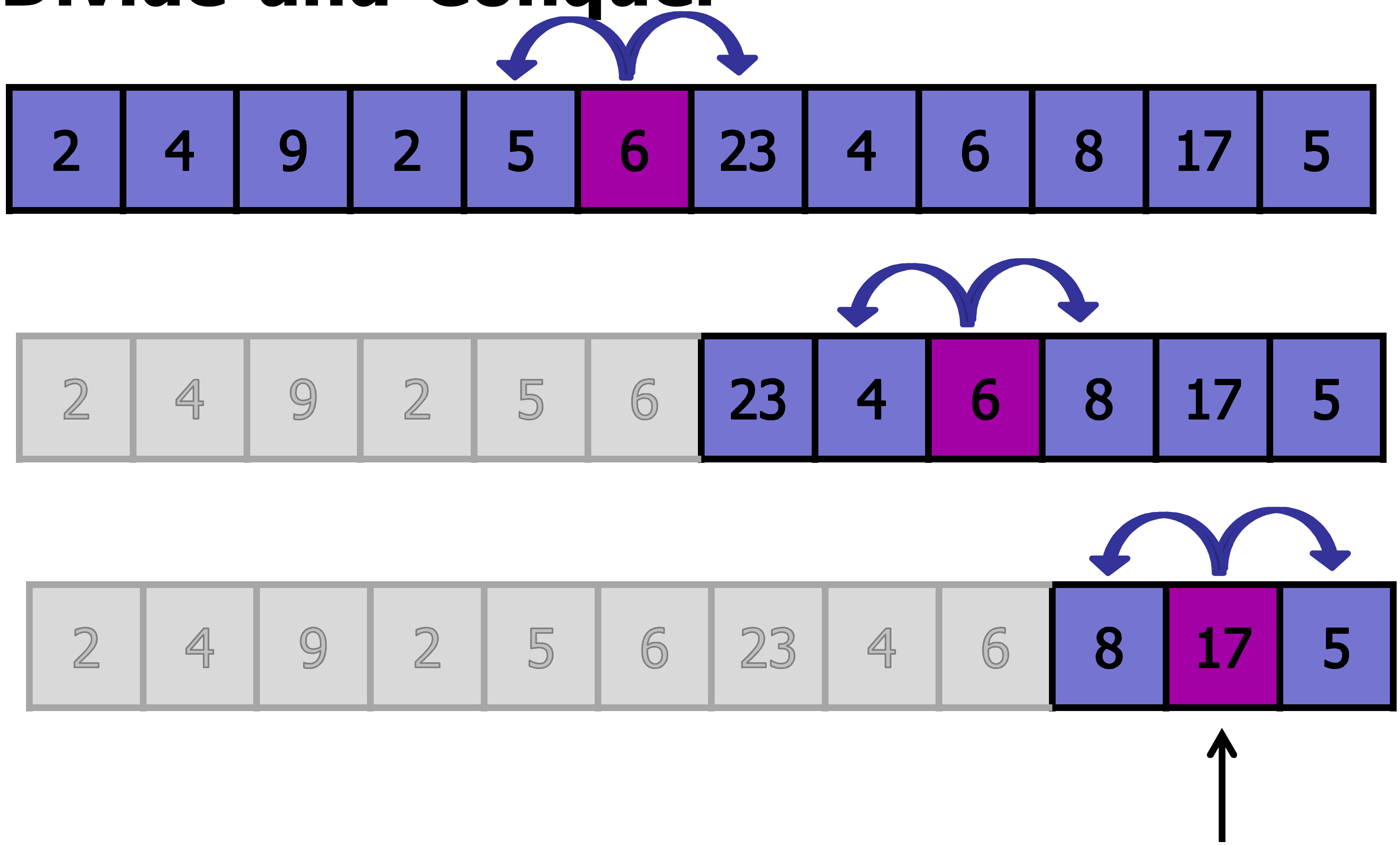
Peak Finding: Algorithm 2

Divide-and-Conquer



Peak Finding: Algorithm 2

Divide-and-Conquer



We found a peak!

Peak Finding: Algorithm 2

Input: Some array A[1..n]

2	4	9	2	11	6	23	4	6	8	17	5

FindPeak(A, n)

    if A[n/2] is a peak **then return** n/2

    else if A[n/2+1] > A[n/2] **then**

        Search for peak in right half.

    else if A[n/2–1] > A[n/2] **then**

        Search for peak in left half.

Peak Finding: Algorithm 2

Proof ?

2	4	9	2	11	6	23	4	6	8	17	5

FindPeak(A, n)

if A[n/2] is a peak **then return** n/2

else if A[n/2+1] > A[n/2] **then**

Search for peak in right half.

else if A[n/2–1] > A[n/2] **then**

Search for peak in left half.

Peak Finding: Algorithm 2

Key property:

– If we recurse in the right half, then there exists a peak in the right half.

2	4	9	2	5	6	23	4	6	8	17	5

Peak Finding: Algorithm 2

Key property:

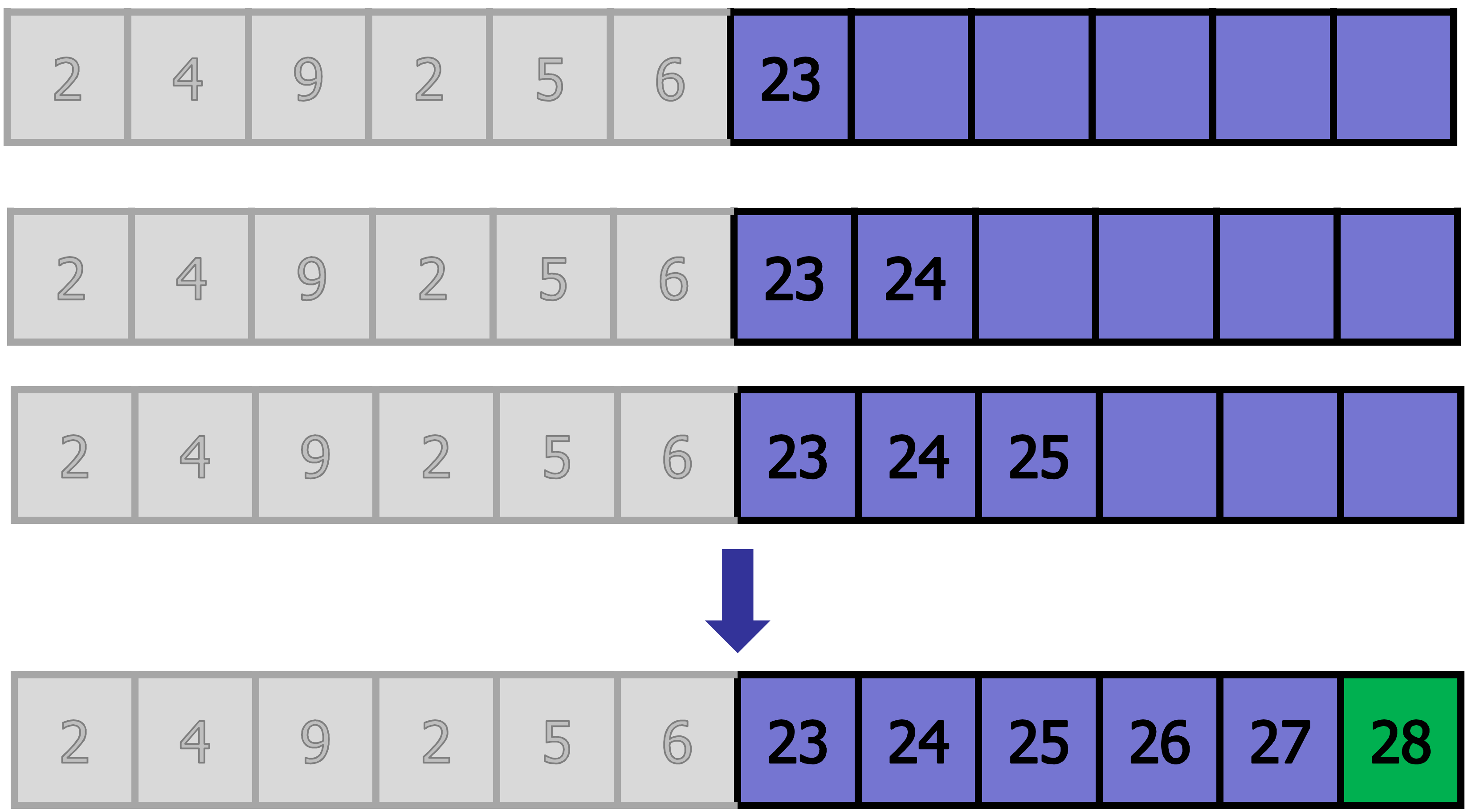
- If we recurse in the right half, then there exists a peak in the right half.

- Proof:
  - Assume no peaks in the right half.
  - Given: A[middle] < A[middle + 1]
  - Since no peaks, A[middle+1] < A[middle+2]
  - Since no peaks, A[middle+2] < A[middle+3]
  - …
  - Since no peaks, A[n-1] < A[n] ← PEAK

Peak Finding: Algorithm 2

Recurse on right half, since 23 > 6.

– Assume no peaks in right half.



Peak Finding: Algorithm 2

Running time?

2	4	9	2	11	6	23	4	6	8	17	5

```
FindPeak(A, n)
    if A[n/2] is a peak then return n/2
    else if A[n/2+1] > A[n/2] then
        Search for peak in right half.
    else if A[n/2–1] > A[n/2] then
        Search for peak in left half.
```

Peak Finding: Algorithm 2

Running time:

Time to find a peak in an array of size n

Recursion

Time for comparing A[n/2] with neighbors

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

Unrolling the recurrence:

$$T(n) = \theta(1) + \theta(1) + \ldots + \theta(1) = O(\log n)$$

Peak Finding: Algorithm 2

Unrolling the recurrence:

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

$$= T(n/4) + \theta(1) + \theta(1)$$

$$= T(n/8) + \theta(1) + \theta(1) + \theta(1)$$

$$\ldots$$

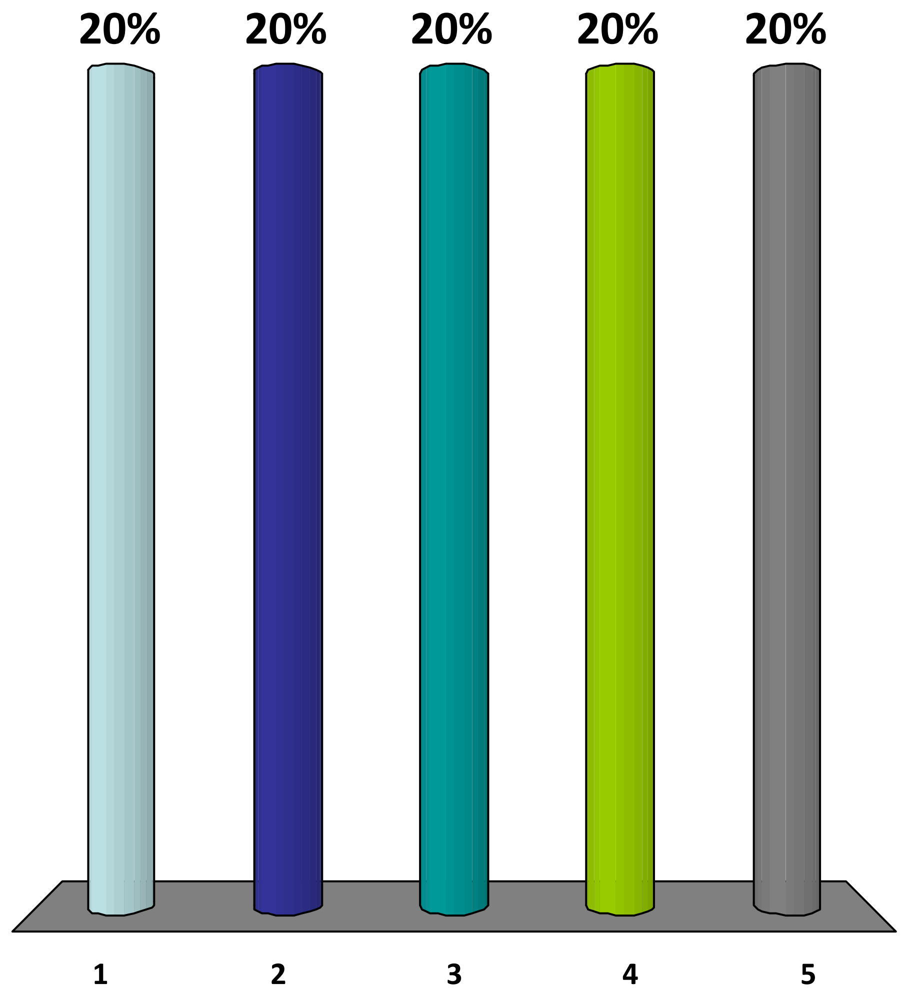
$$\ldots$$

$$= T(1) + \theta(1) + \ldots + \theta(1) =$$

$$= \theta(1) + \theta(1) + \ldots + \theta(1) =$$

How many times can you divide a number ***n*** in half before you reach 1?

1. n/4
2. √n
3. $\log_2(n)$
4. arctan(1+√5/2n)
5. I don't know.



Peak Finding: Algorithm 2

Unrolling the recurrence:

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

$$= T(n/4) + \theta(1) + \theta(1)$$

$$= T(n/8) + \theta(1) + \theta(1) + \theta(1)$$

…

…

$$= T(1) + \theta(1) + \ldots + \theta(1) =$$

$$= \theta(1) + \theta(1) + \ldots + \theta(1) =$$

Peak Finding: Algorithm 2

Running time:

Time to find a peak in an array of size n

Recursion

Time for comparing A[n/2] with neighbors

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

Unrolling the recurrence:

$$T(n) = \underbrace{\theta(1) + \theta(1) + \ldots + \theta(1)}_{\log(n)} = O(\log n)$$

Preview

After the break:

The 2nd dimension!

Peak Finding 2D (the sequel)

Given: 2D array A[1..n, 1..m]

10	8	5	2	1
3	2	1	5	7
17	5	1	4	1
7	9	4	6	4
8	1	1	2	6

Output: a peak that is not smaller than the (at most) 4 neighbors.

2D: Algorithm 1

Step 1: Find global max for each column

3	4	5	2
2	1	2	5
1	9	1	2
7	5	3	3

7 9 5 3 ← Find 1D peak.

Step 2: Find <u>peak</u> in the array of max elements.

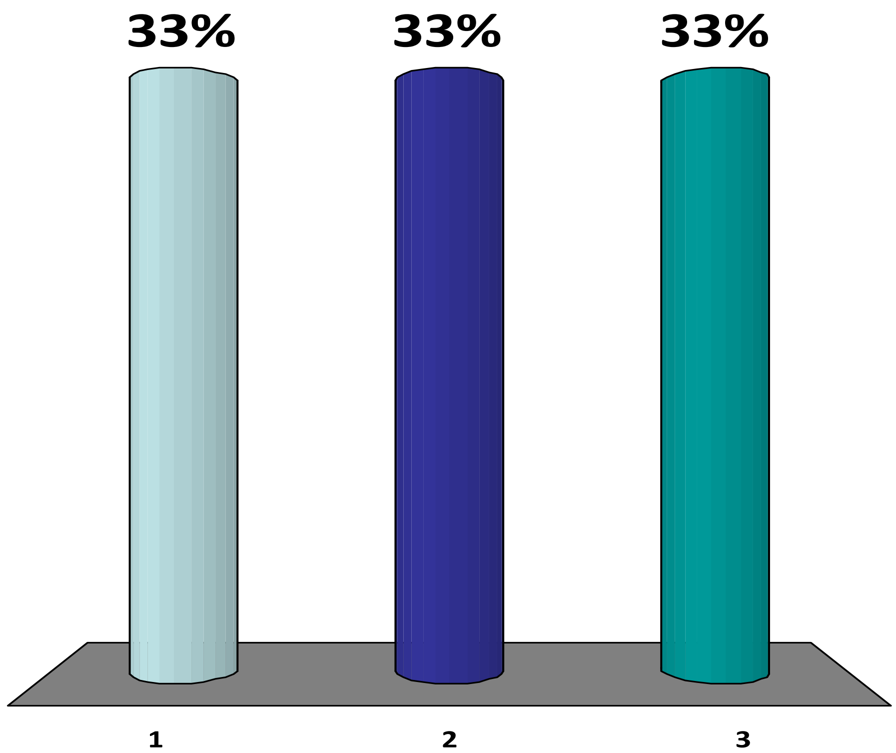
Algorithm 1-2D

Step 1: Find global max for each column.

Step 2: Find peak in the max array.

Is this algorithm correct?

1. Yes
2. No
3. I’m confused…



0 of 60

2D: Algorithm 1

Step 1: Find global max for each column

3	4	5	2
2	1	2	5
1	9	1	2
7	5	3	3

7 9 5 3 ← Find 1D peak.

Step 2: Find peak in the array of max elements.

Running time: O(mn + m log(m))

2D: Algorithm 2

Step 1: Find a (local) peak for each column

3	4	5	2
2	1	2	5
1	9	1	2
7	5	3	3

7 9 5 3 ← Find 1D peak.

Step 2: Find peak in the array of peaks.

Algorithm 2-2D

Step 1: Find 1D-peak for each column.

Step 2: Find peak in the max array.

Is this algorithm correct?

1. Yes
2. No
3. I’m confused…

2D: Algorithm 2

Step 1: Find a global max for each column

3	4	5	2
2	1	2	5
1	9	1	2
7	5	3	3

? ? ? ? ← Find 1D peak.

Step 2: Find peak in the array of peaks.

7	10	12	20	7	9	4	3	1	18	5	17	4
19	11	7	4	6	8	8	3	5	6	8	14	8
6	9	14	4	7	9	3	5	9	8	3	10	6

? ? ? ? ? ? ? ? ? ? ? ? ?

Find 1D Peak:

Step 1: Check middle element.

Step 2: Recurse left/right half.

7	10	12	20	7	9	4	3	1	18	5	17	4
19	11	7	4	6	8	8	3	5	6	8	14	8
6	9	14	4	7	9	3	5	9	8	3	10	6
?	?	?	?	?	8	10	12	?	?	?	?	?

Find 1D Peak:

Step 1: Check middle element.

Step 2: Recurse left/right half.

7	**10**	**12**	**20**	**7**	**9**	**4**	**3**	**1**	**18**	**5**	**17**	**4**
19	**11**	**7**	**4**	**6**	**8**	**8**	**3**	**5**	**6**	**8**	**14**	**8**
6	**9**	**14**	**4**	**7**	**9**	**3**	**5**	**9**	**8**	**3**	**10**	**6**

? ? ? ? ? 8 10 12 ? 6 8 9 ?

Find 1D Peak:

Step 1: Check middle element.

Step 2: Recurse left/right half.

7	10	12	20	7	9	4	3	1	18	5	17	4
19	11	7	4	6	8	8	3	5	6	8	14	8
6	9	14	4	7	9	3	5	9	8	3	10	6

?	?	?	?	?	8	10	12	?	6	8	9	4

Find 1D Peak:

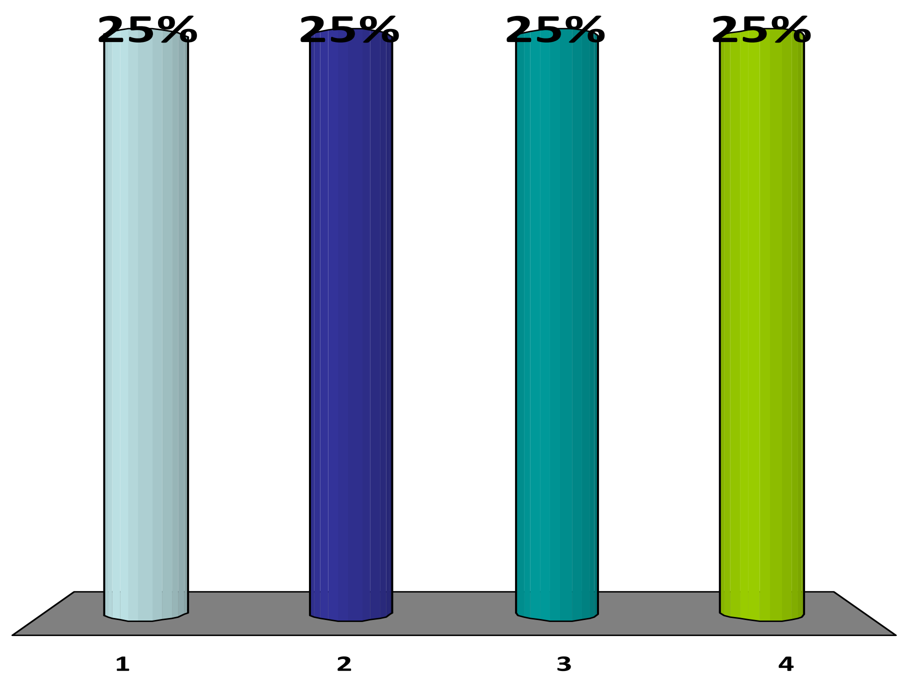
- Step 1: Check middle element.
- Step 2: Recurse left/right half.

7	10	12	20	7	9	4	3	1	18	5	17	4
19	11	7	4	6	8	8	3	5	6	8	14	8
6	9	14	4	7	9	3	5	9	8	3	10	6

? ? ? ? ? **8** **10** **12** ? **6** **8** **9** **4**

How many columns do we need to examine?

1. O(m)
2. O(√m)
3. O(log m)
4. O(1)



0 of 60

2D: Algorithm 2

Find peak in the array of peaks:

- Use 1D Peak Finding algorithm
- For each column examined by the algorithm, find the maximum element in the column.

Running time:

- 1D Peak Finder Examines O(log m) columns
- Each column requires O(n) time to find max
- Total: **O(n log m)**

(Much better than O(nm) of before.)

2D Algorithm 3

Any ideas??

2D Algorithm 3

Divide-and-Conquer

1. Find MAX element of middle column.
2. If found a peak, DONE.
3. Else:
   - If left neighbor is larger, then recurse on left half.
   - If right neighbor is larger, then recurse on right half.

10	8	4	2	1
3	2	2	12	13
17	5	1	11	1
7	4	6	9	4
8	1	1	2	6

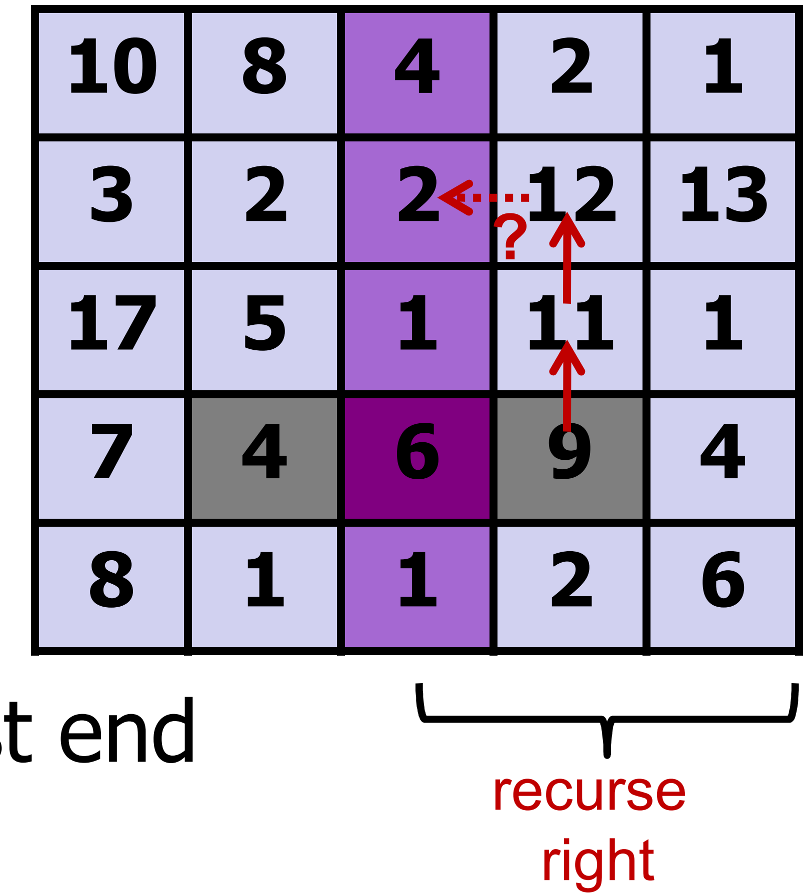
recurse right

2D Algorithm 3

Correctness

1. Assume no peak on right half.
2. Then, there is some increasing path:

   $9 \rightarrow 11 \rightarrow 12 \rightarrow \ldots$



3. Eventually, the path must end at a max.
4. If there is no max in the right half, then it must cross to the left half… Impossible!

2D Algorithm 3

Divide-and-Conquer

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

Recurse *once* on array of size [n, m/2]

Do n work to find max element in column.

10	8	4	2	1
3	2	2	12	13
17	5	1	11	1
7	4	6	9	4
8	1	1	2	6

recurse right

Recurrence Analysis

$$
\begin{aligned}
T(n, m) &= T(n, m/2) + n \\
&= T(n, m/4) + n + n \\
&= T(n, m/8) + n + n + n \\
&= T(n, m/16) + n + n + n + n \\
&= \ldots
\end{aligned}
$$

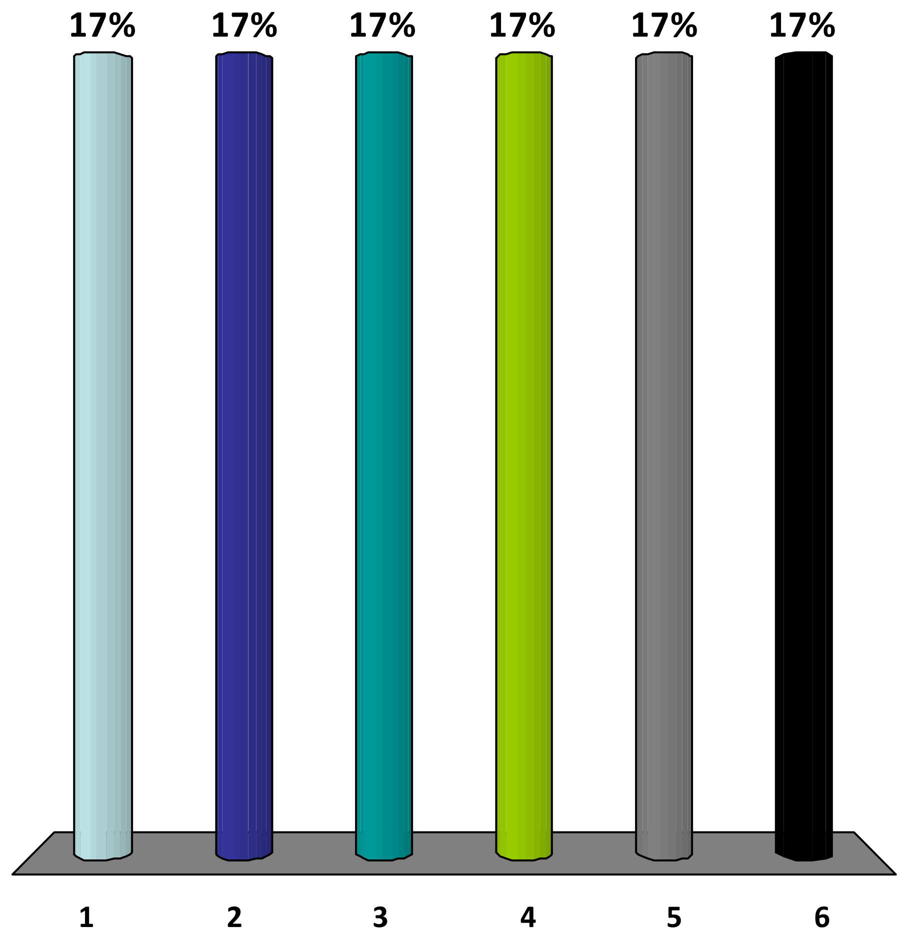
Recurrence Analysis

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

$T(n) = ??$

1. $O(\log n)$
2. $O(\log m)$
3. $O(nm)$
4. $O(n \log m)$
5. $O(m \log n)$
6. $O(n! \cos(\Pi/m))$

0 of 60



2D Algorithm 3

Divide-and-Conquer

1. Find MAX element of middle column.
2. If found a peak, DONE.
3. Else:
   – If left neighbor is larger, then recurse on left half.
   – If right neighbor is larger, then recurse on right half.

T(n) = O(n log m)

10	8	4	2	1
3	2	2	12	13
17	5	1	11	1
7	4	6	9	4
8	1	1	2	6

recurse right

2D Algorithm 4

We want to do better than O(n log m)…

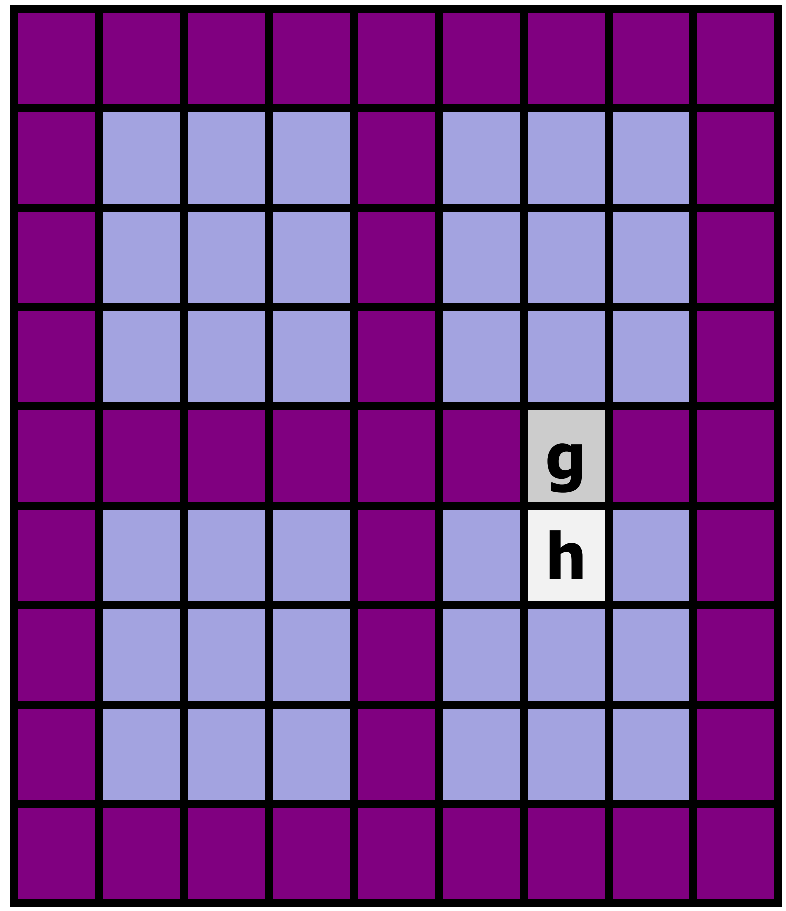
Any ideas??

2D Algorithm 4

Divide-and-Conquer

1. Find MAX element on border + cross.
2. If found a peak, DONE.
3. Else:

   Recurse on quadrant containing element bigger than MAX.



Example: MAX = g

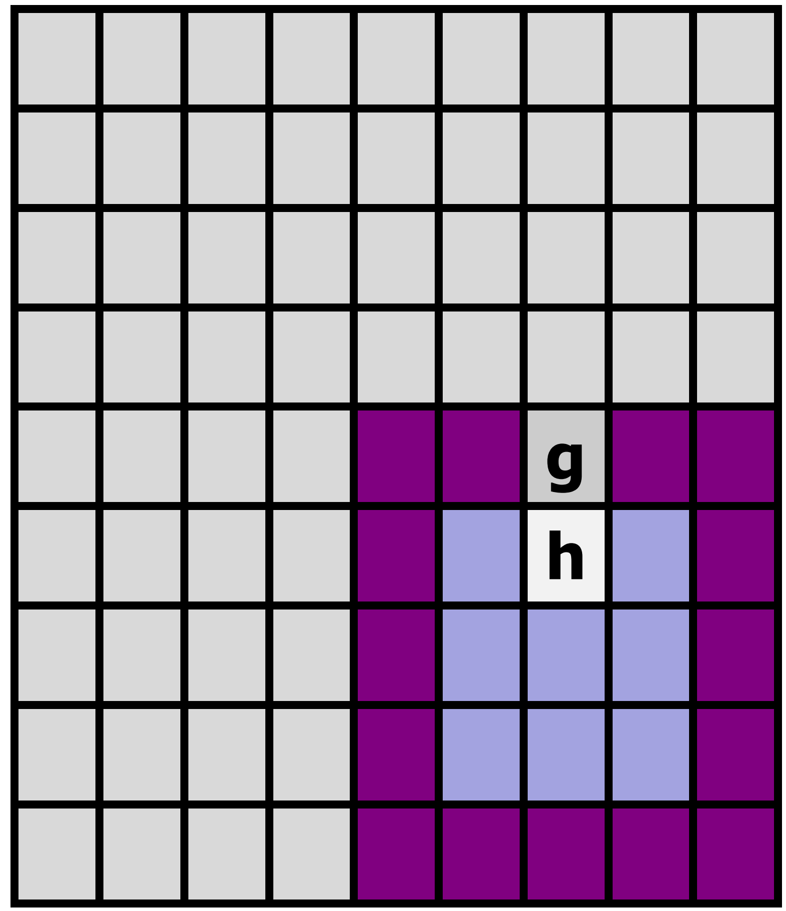
h > g

2D Algorithm 4

Divide-and-Conquer

1. Find MAX element on border + cross.
2. If found a peak, DONE.
3. Else:

   Recurse on quadrant containing element bigger than MAX.



Example: MAX = g

h > g

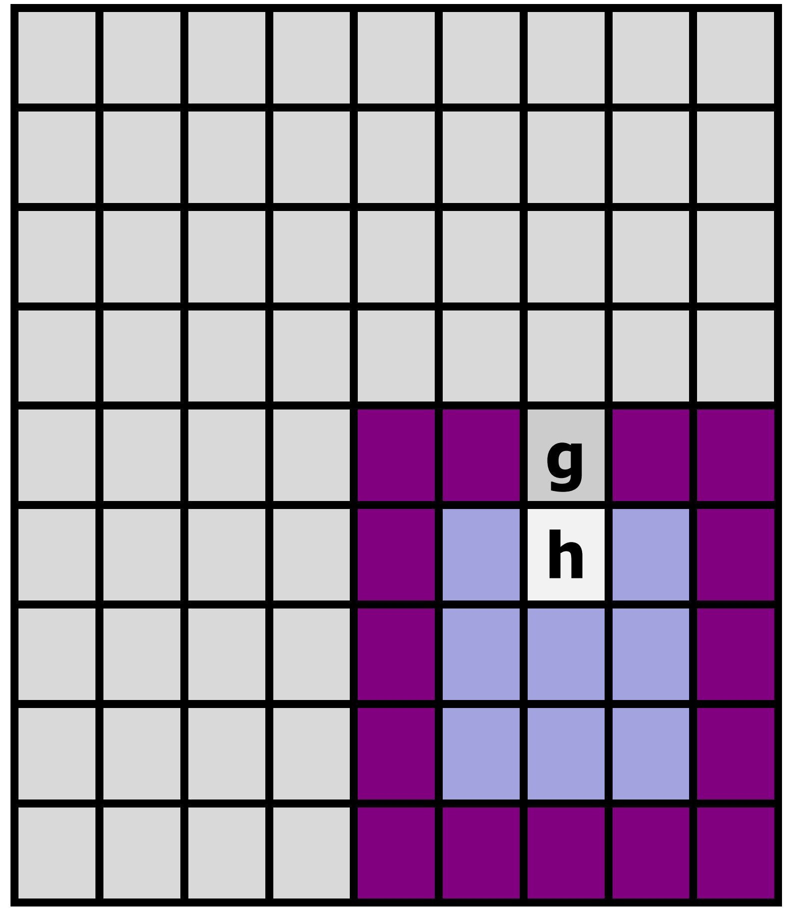
2D Algorithm 4

Correctness

1. The quadrant contains a peak.

   Proof: as before.

2. Every peak in the quadrant is NOT a peak in the matrix.



2D Algorithm 4

Correctness

1. The quadrant contains a peak.

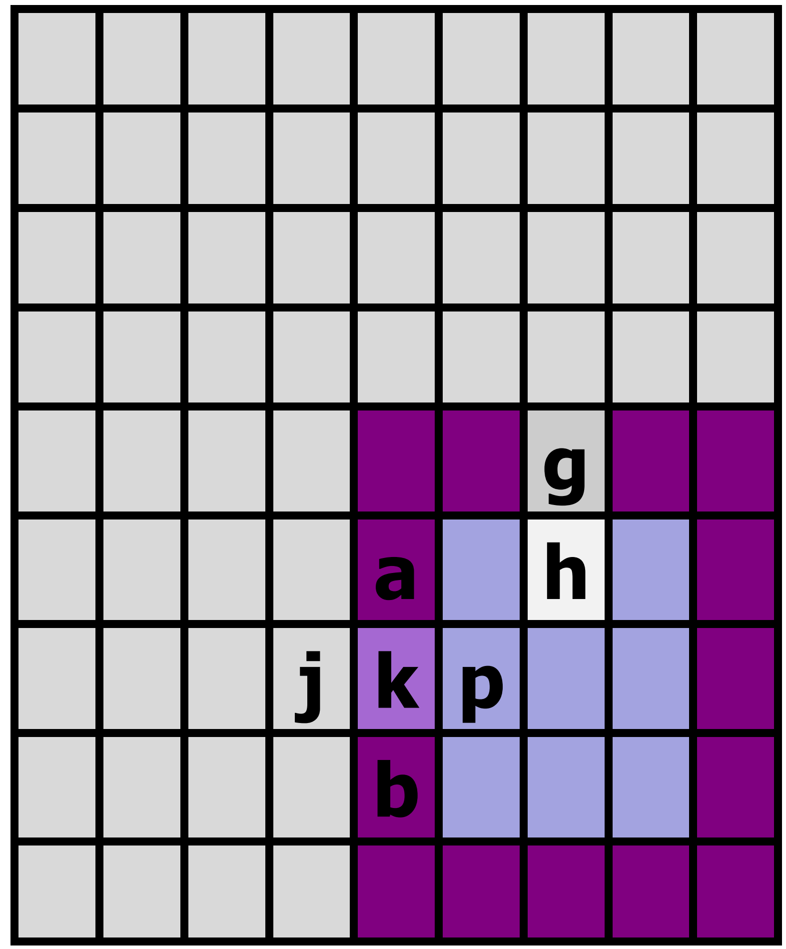
   Proof: as before.

2. Every peak in the quadrant is NOT a peak in the matrix.

   Example: j > k > p

   k > a

   k > b



2D Algorithm 4

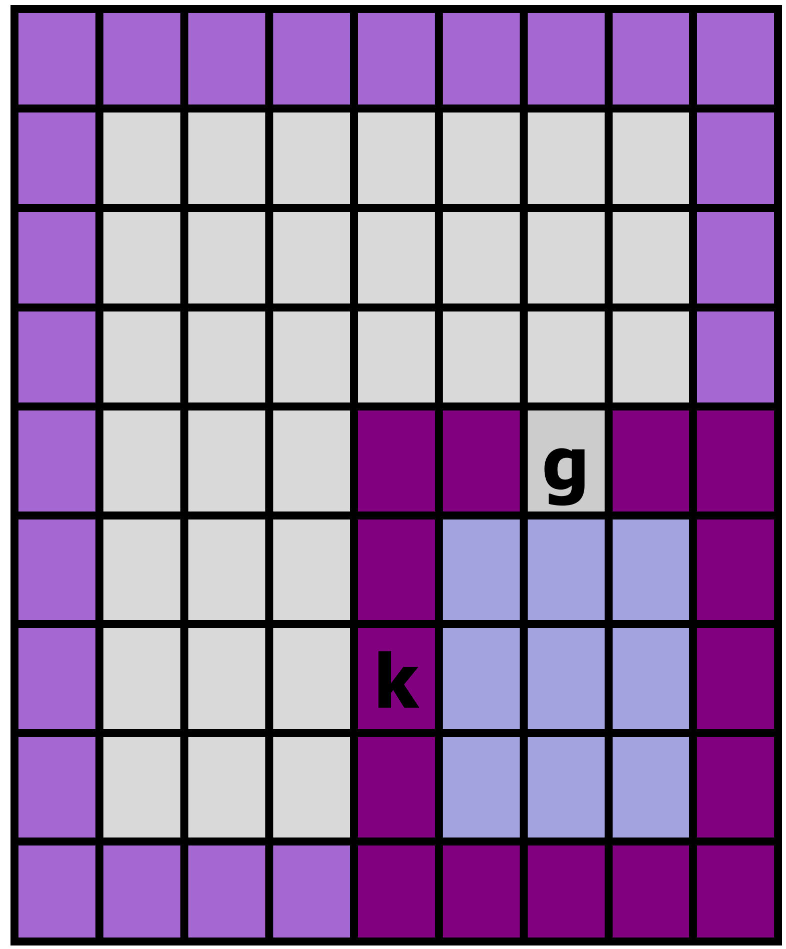
Correctness

Key property:

Find a peak at least as large as every element on the boundary.

Proof:

If recursing finds an element at least as large as g, and g is as big as the biggest element on the boundary, then the peak is as large as every element on the boundary.



2D Algorithm 4

Divide-and-Conquer

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

Recurse *once* on array of size [n/2, m/2]

Do 6(n+m) work to find max element.

Recurrence Analysis

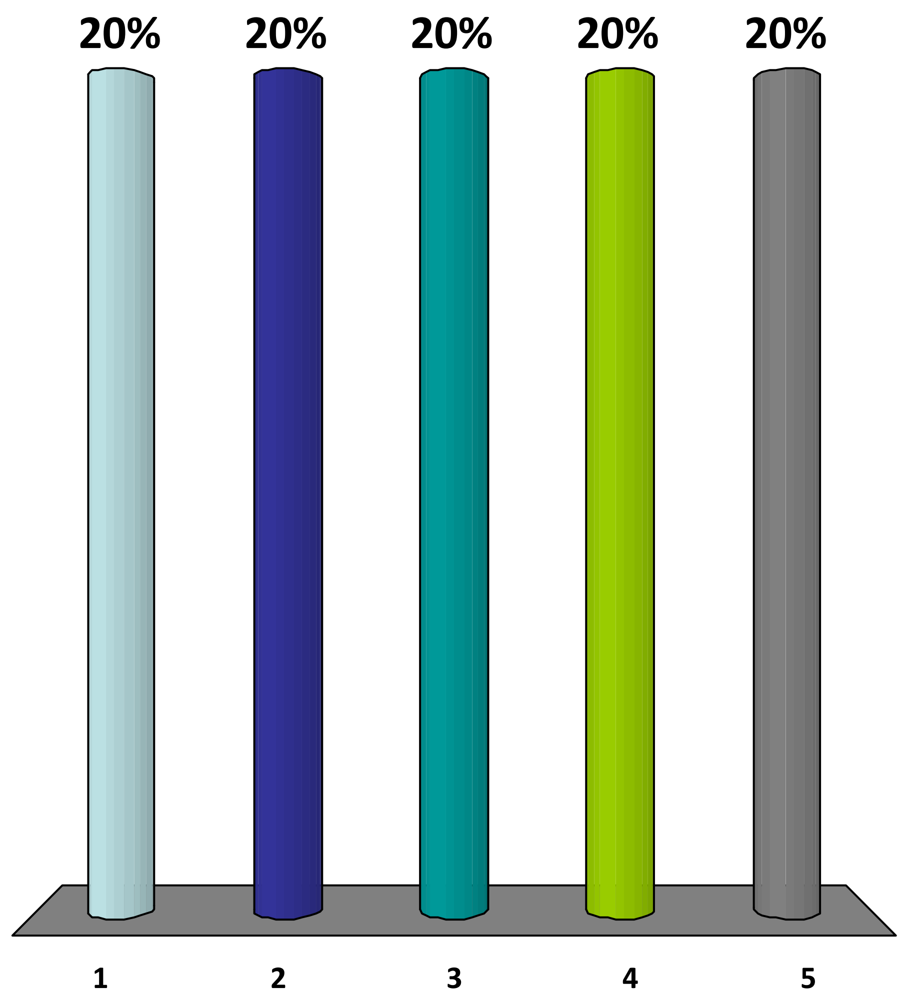
$$
\begin{aligned}
T(n, m) &= T(n/2, m/2) + cn + cm \\
&= T(n/4, m/4) + cn/2 + cm/2 + n + m \\
&= T(n/8, m/8) + cn/4 + cm/4 + \ldots \\
&= \ldots
\end{aligned}
$$

Recurrence Analysis

$$T(n, m) = T(n/2, m/2) + cn + cm$$

T(n) = ??

1. O(log n)
2. O(nm)
3. O(n log m)
4. O(m log n)
5. O(n+m)



0 of 60

Recurrence Analysis

$$
\begin{aligned}
T(n, m) &= T(n/2, m/2) + cn + cm \\
&= T(n/4, m/4) + cn/2 + cm/2 + n + m \\
&= T(n/8, m/8) + cn/4 + cm/4 + \ldots \\
&= \ldots \\
\\
&= cn(1 + 1/2 + 1/4 + \ldots) + \\
&\quad\ cm(1 + 1/2 + 1/4 + \ldots) \\
&< 2cn + 2cm \\
&= O(n + m)
\end{aligned}
$$

Summary

1D Peak Finding

- Divide-and-Conquer
- O(log n) time

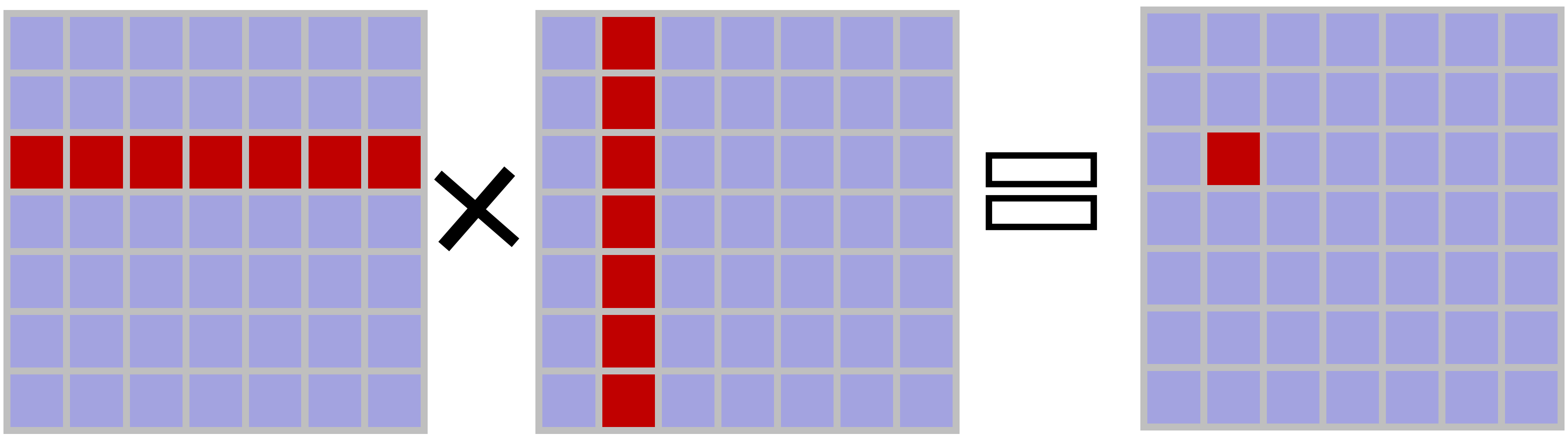
2D Peak Finding

- Simple algorithms: O(n log m)
- Careful Divide-and-Conquer: O(n + m)

Matrix Multiplication

Given: two matrices A[n,n] and B[n,n]

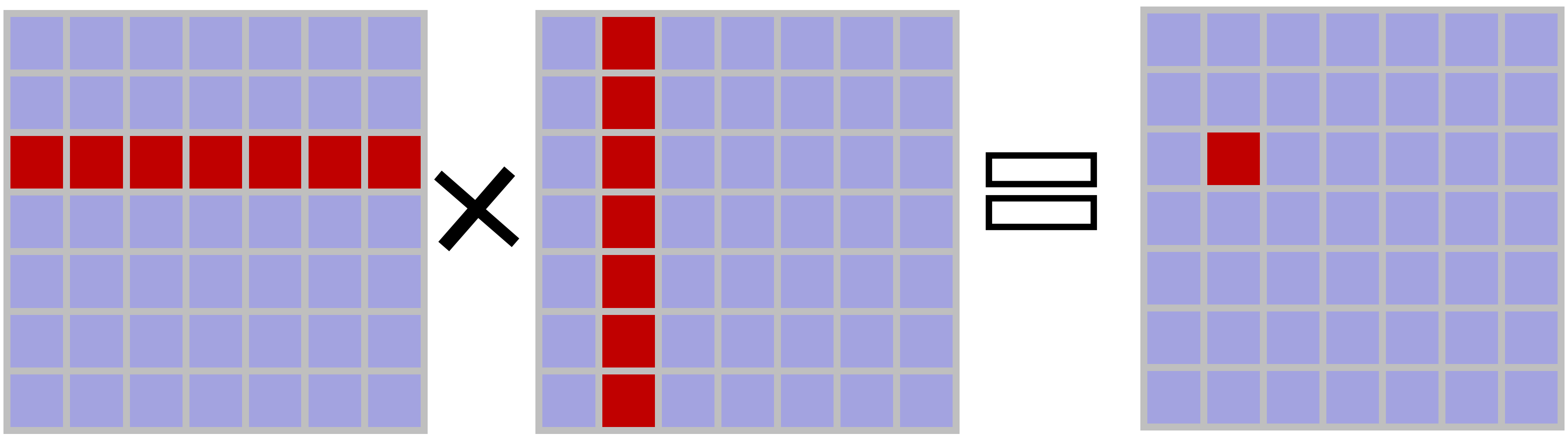
Calculate: matrix C = AB



Matrix Multiplication

Given: two matrices A[n,n] and B[n,n]

Calculate: matrix C = AB



$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

Matrix Multiplication

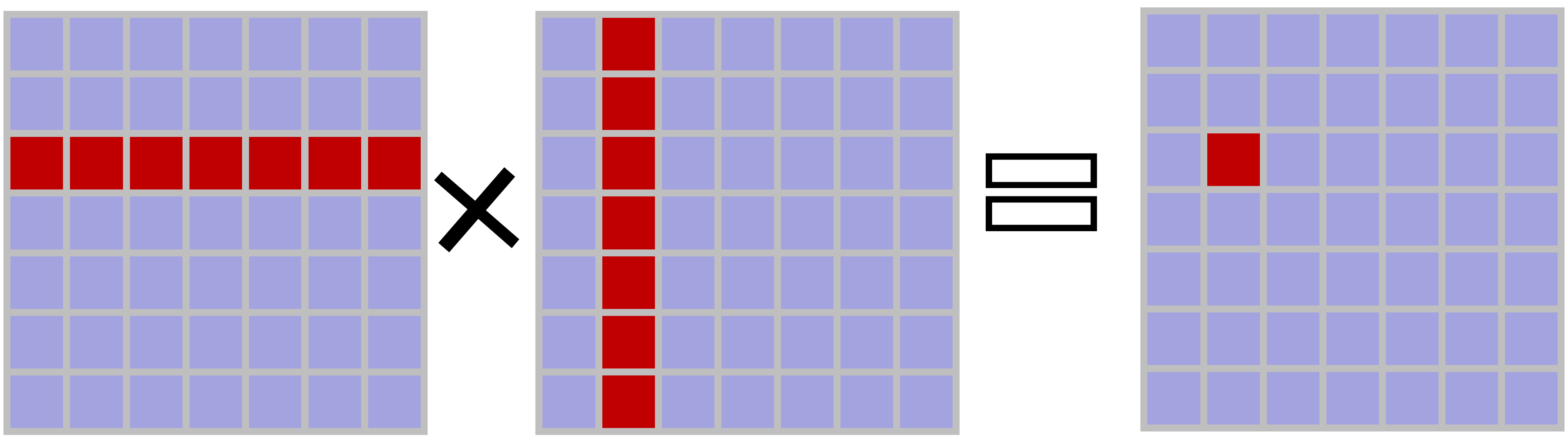
Multiply(A,B)

for $i = 1$ **to** n **do**

for $j = 1$ **to** n **do**

$C_{ij} = 0$

for $k = 1$ **to** n **do** $C_{ij} \mathrel{+}= A_{ik} * B_{kj}$



Matrix Multiplication

Multiply(A,B)

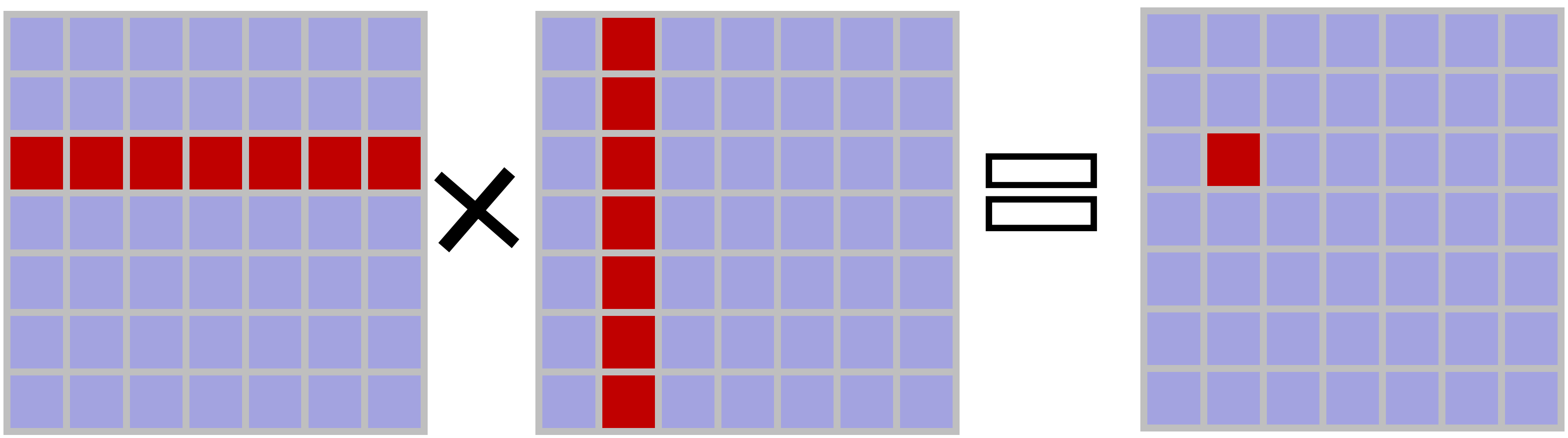
for $i = 1$ **to** n **do**

for $j = 1$ **to** n **do**

$C_{ij} = 0$

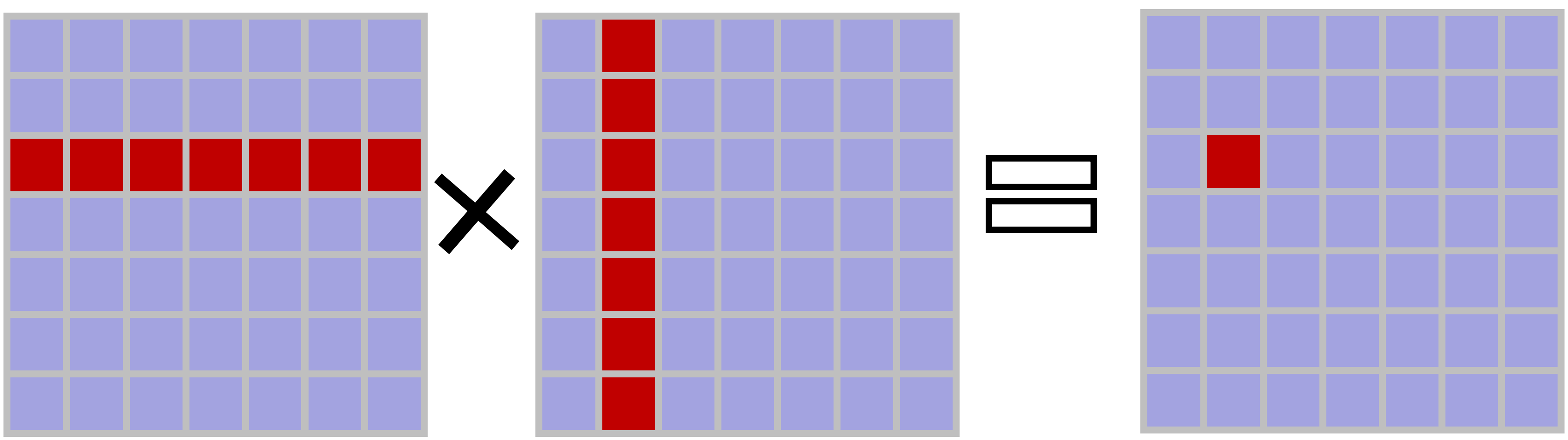
for $k = 1$ **to** n **do** $C_{ij} \mathrel{+}= A_{ik} * B_{kj}$

$O(n^3)$



Matrix Multiplication

Ideas for improvement?



Matrix Multiplication

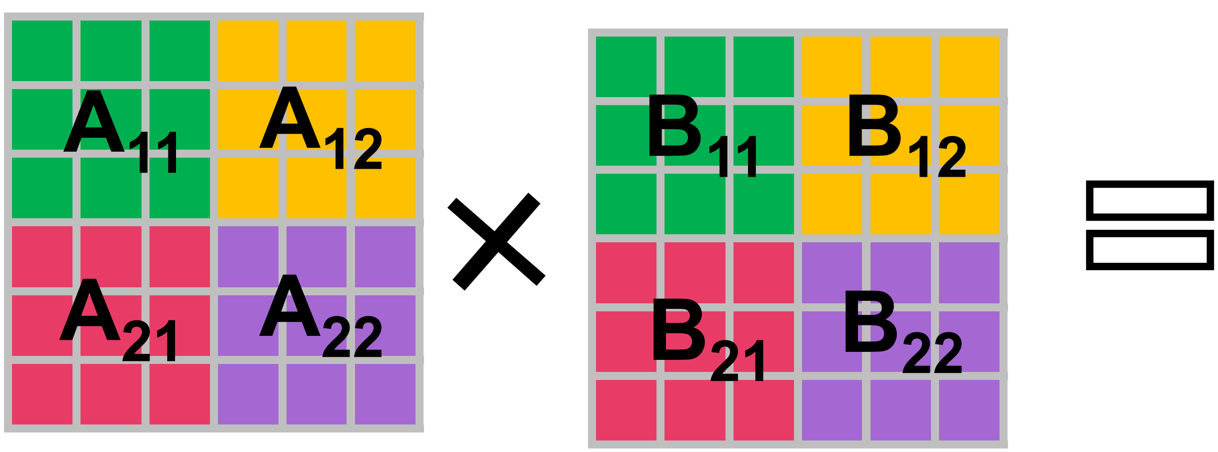
Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



Matrix Multiplication

Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$T(n) = 8T(n/2) + O(n^2)$$

Substitution Method

Solve:

$$T(n) = 8T(n/2) + kn^2$$

Guess:

$$T(n) = n^3 - kn^2$$

Substitution Method

Solve:

$T(n) = 8T(n/2) + kn^2$

Guess:

$T(n) = n^3 - kn^2$

Test: $8T(n/2) + kn^2$

$$
\begin{aligned}
T(n/2) &= (n/2)^3 - k(n/2)^2 \\
&= n^3/8 - kn^2/4
\end{aligned}
$$

Substitution Method

Solve:

$$T(n) = 8T(n/2) + kn^2$$

Guess:

$$T(n) = n^3 - kn^2$$

Test: $8T(n/2) + kn^2$

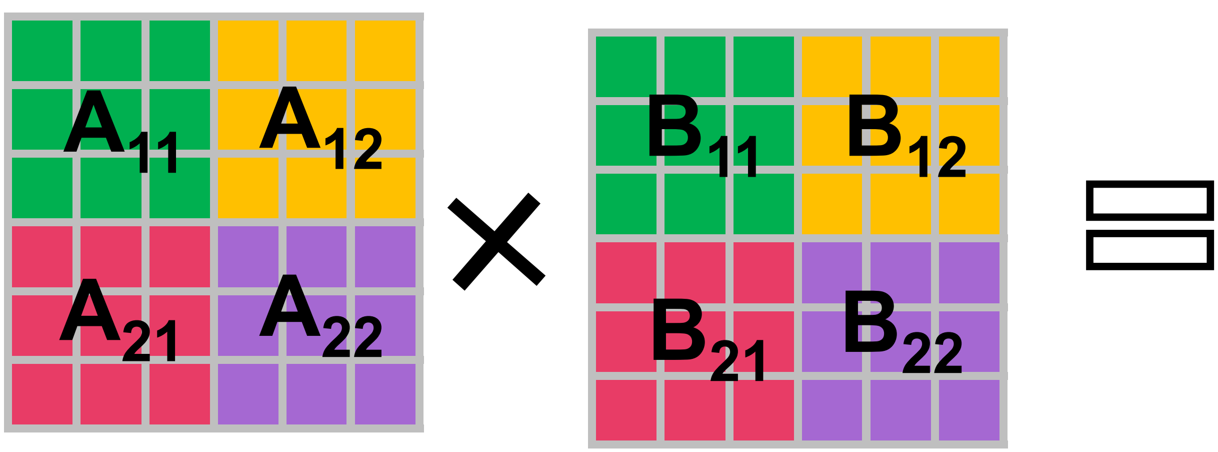
$$T(n/2) = (n/2)^3 - k(n/2)^2$$
$$= n^3/8 - kn^2/4$$

$$8T(n/2)+kn^2 = 8(n^3/8 - kn^2/4)+kn^2$$
$$= n^3 - 2kn^2+kn^2 = T(n)$$

Matrix Multiplication

Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



Matrix Magic

Define:

$M_1 = (A_{11}+A_{22})(B_{11} + B_{22})$

$M_2 = (A_{21}+ A_{22})B_{11}$

$M_3 = A_{11}(B_{12} - B_{22})$

$M_4 = A_{22}(B_{21} - B_{11})$

$M_5 = (A_{11}+A_{12})B_{22}$

$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

Notice: **7** multiplications!!

Matrix Magic

Calculate:

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Really!!

Magic!!

Matrix Multiplication

Strassen's Method:

$$T(n) = 7T(n/2) + \theta(n^2)$$

About 18 matrix additions/subtractions

Not 8!

Matrix Multiplication

Strassen's Method:

$T(n) = 7T(n/2) + \theta(n^2)$

$T(n) \cong n^{\log(7)} \cong n^{2.81}$

(Faster when N > 32, approximately)

Best known to date:

$T(n) \cong O(n^{2.376})$

(Theoretical use only.)

Summary

1D Peak Finding

- Divide-and-Conquer
- O(log n) time

2D Peak Finding

- Simple algorithms: O(n log m)
- Careful Divide-and-Conquer: O(n + m)

Matrix multiplication: Strassens Method