Onwards…

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 peak in the array of max elements.

Algorithm 1-2D

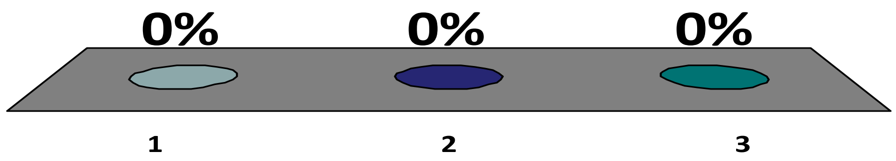
Step 1: Find global max for each column.

Step 2: Find peak in the max array.

Is this algorithm correct?

1. Yes
2. No
3. I’m confused…

Response Counter



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.

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 + 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

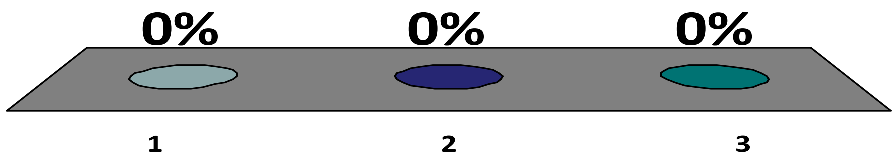
Step 1: Find 1D-peak for each column.

Step 2: Find peak in the max array.

Is this algorithm correct?

1. Yes
2. No
3. I’m confused…

Response Counter



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**

3 4 3 3 ← Find 1D peak.

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

2D: Algorithm 1

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.

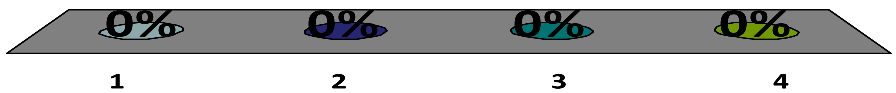
7	10	12	20	7	9	4	3	1	18	5	17	4
19	11	7	4	6	8	8	3	5	6	8	14	8
6	9	14	4	7	9	3	5	9	8	3	10	6

? ? ? ? ? **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)

Response Counter



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

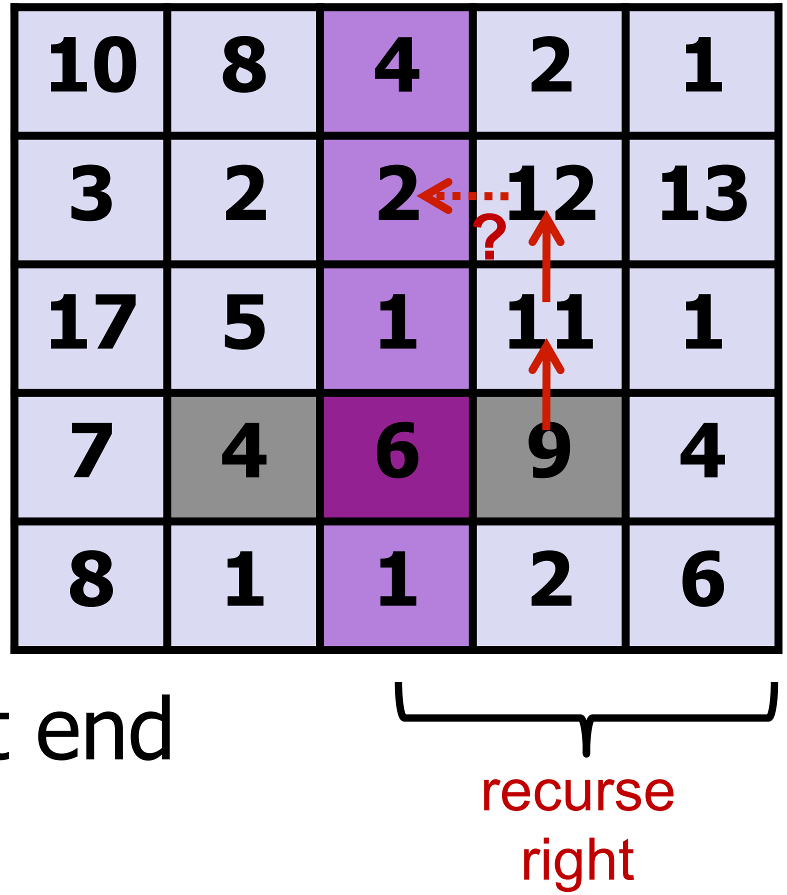
recurse right

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

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

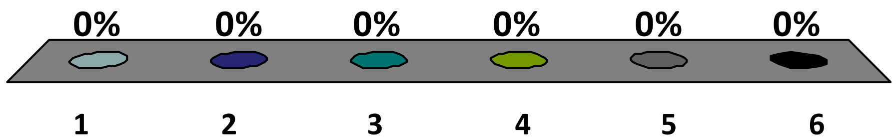
Recurrence Analysis

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

$T(n) = ??$

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

Response Counter



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

T(n) = O(n log m)

2D Algorithm 4

We want to do better than $O(n \log m)$...

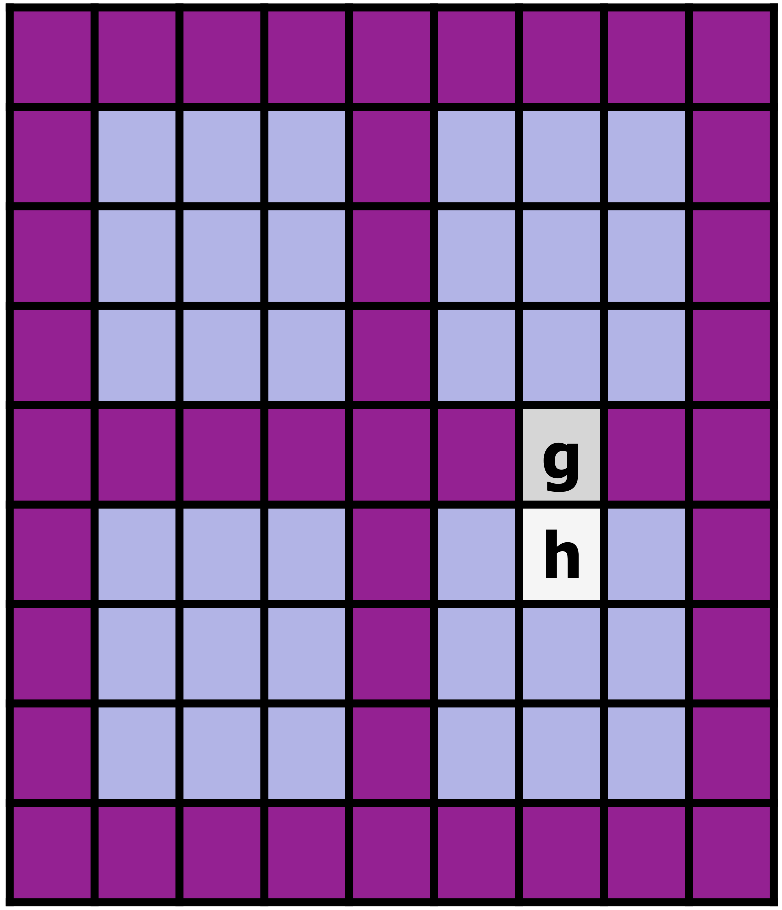
Any ideas??

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

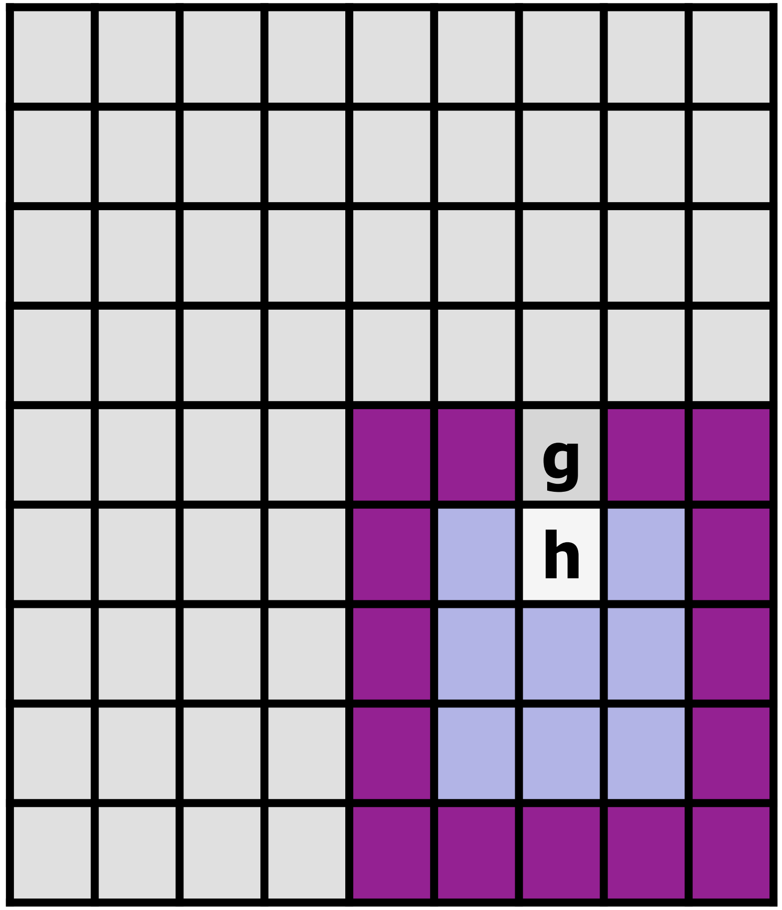
h > g

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

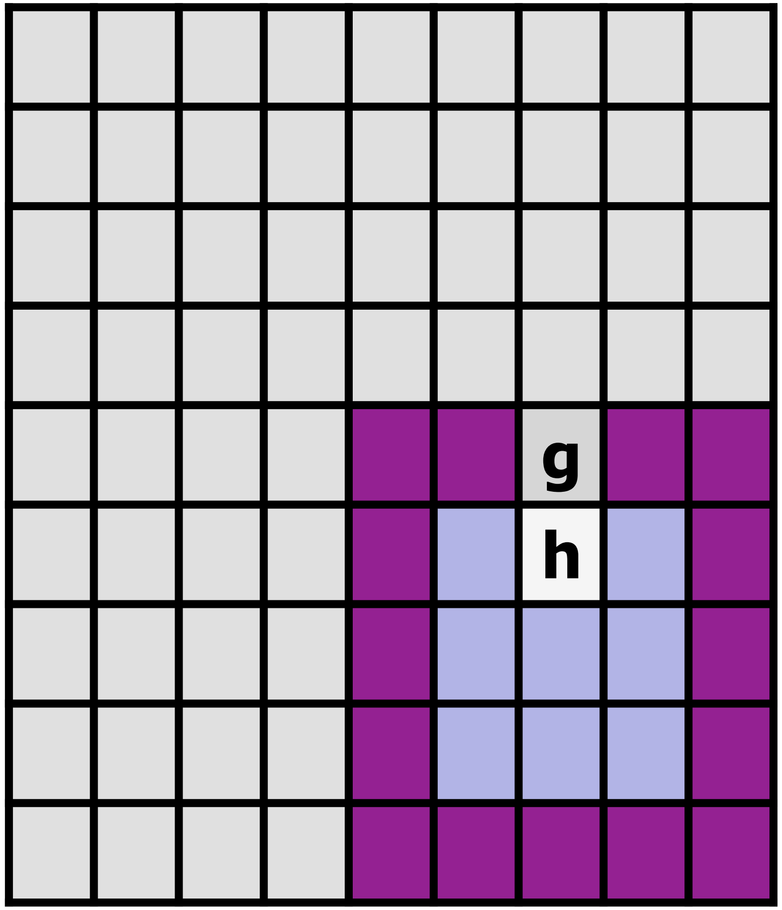
2D Algorithm 4

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.

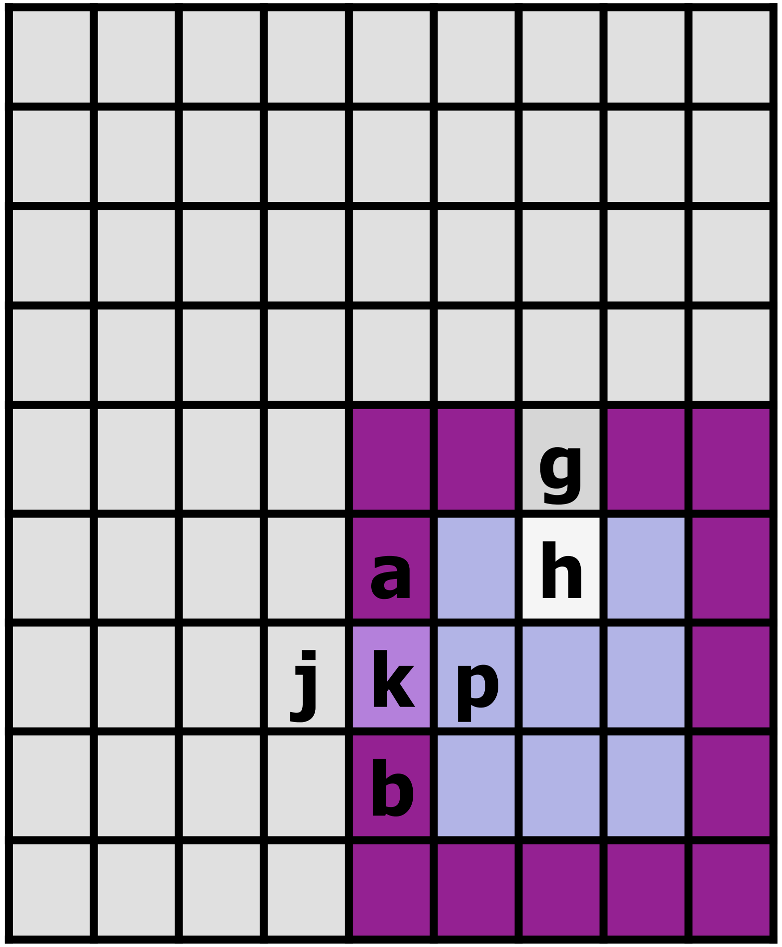
   Proof: as before.

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

   Example: j > k > p

   k > a

   k > b



2D Algorithm 4

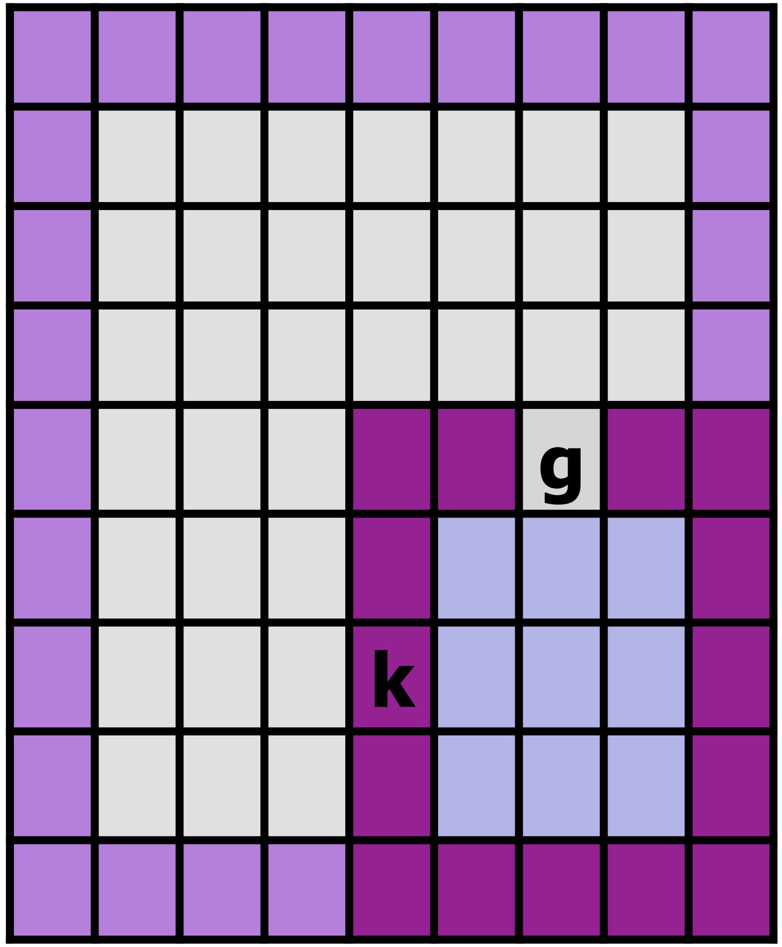
Correctness

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

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

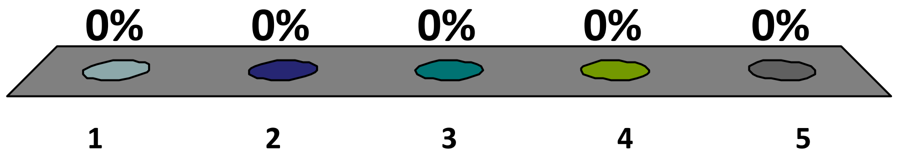
Recurrence Analysis

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

$T(n) = ??$

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

Response Counter



Recurrence Analysis

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

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)