Today: Divide and Conquer!

Algorithm Analysis

- Big-O Notation
- Model of computation

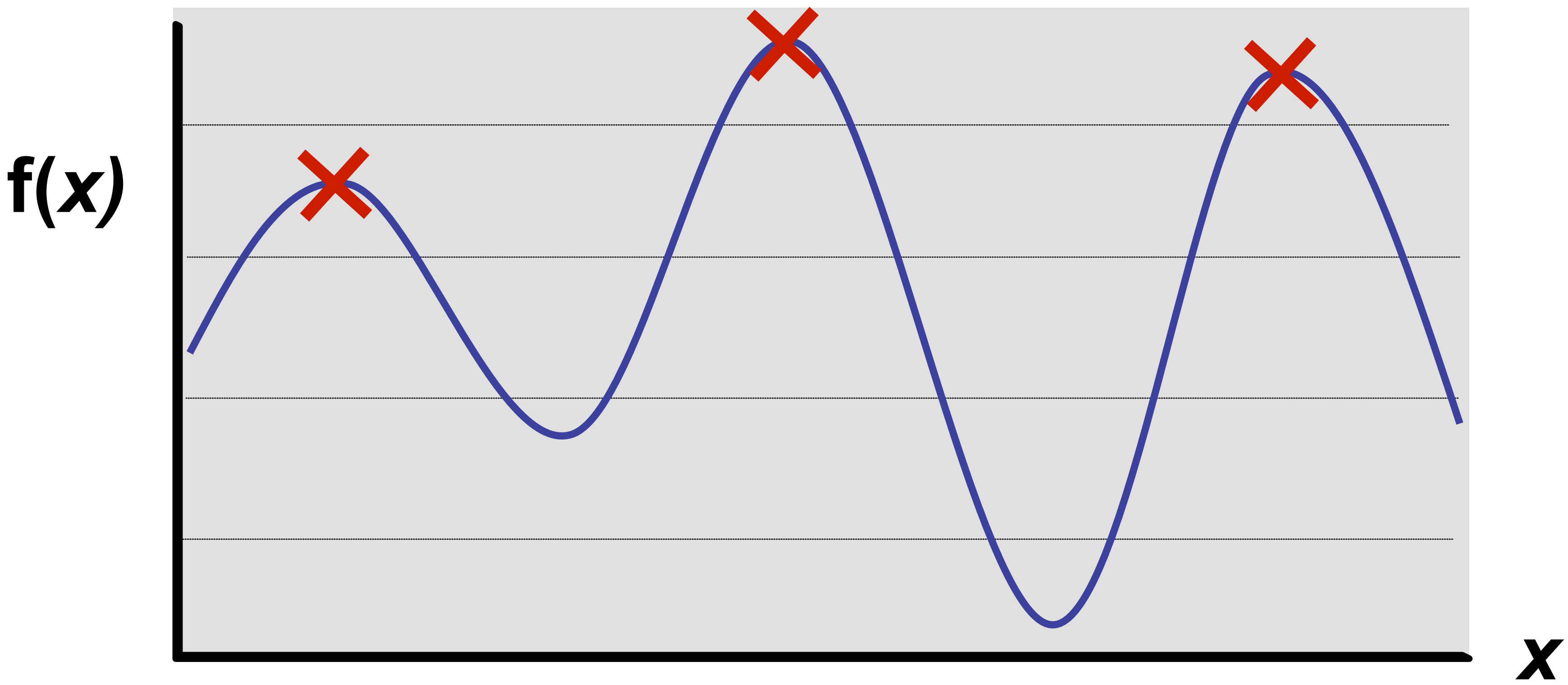
Searching

Peak Finding

- 1-dimension
- 2-dimensions

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

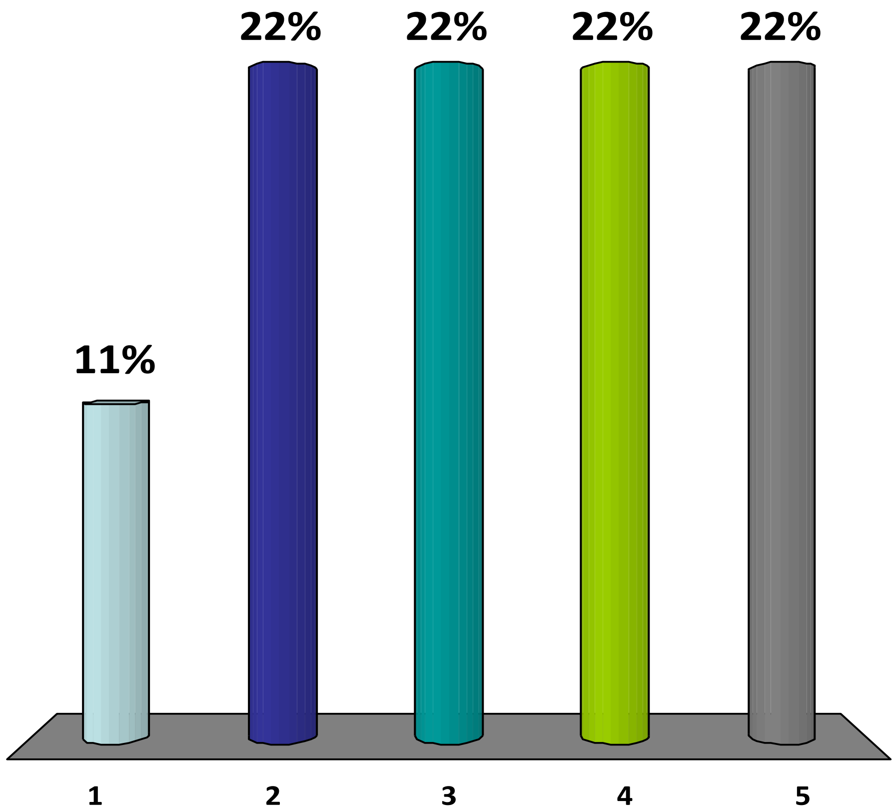
How long to find a global maximum?

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)

Response Counter



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: $O(n)$

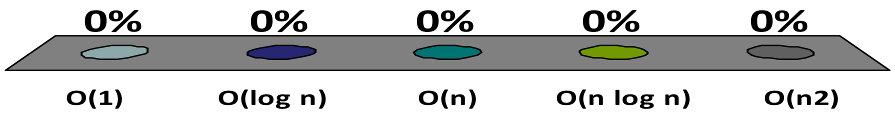
Global Maximum

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. $O(n^2)$

Response Counter



Global Maximum

Sorted array: A[1..n]

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

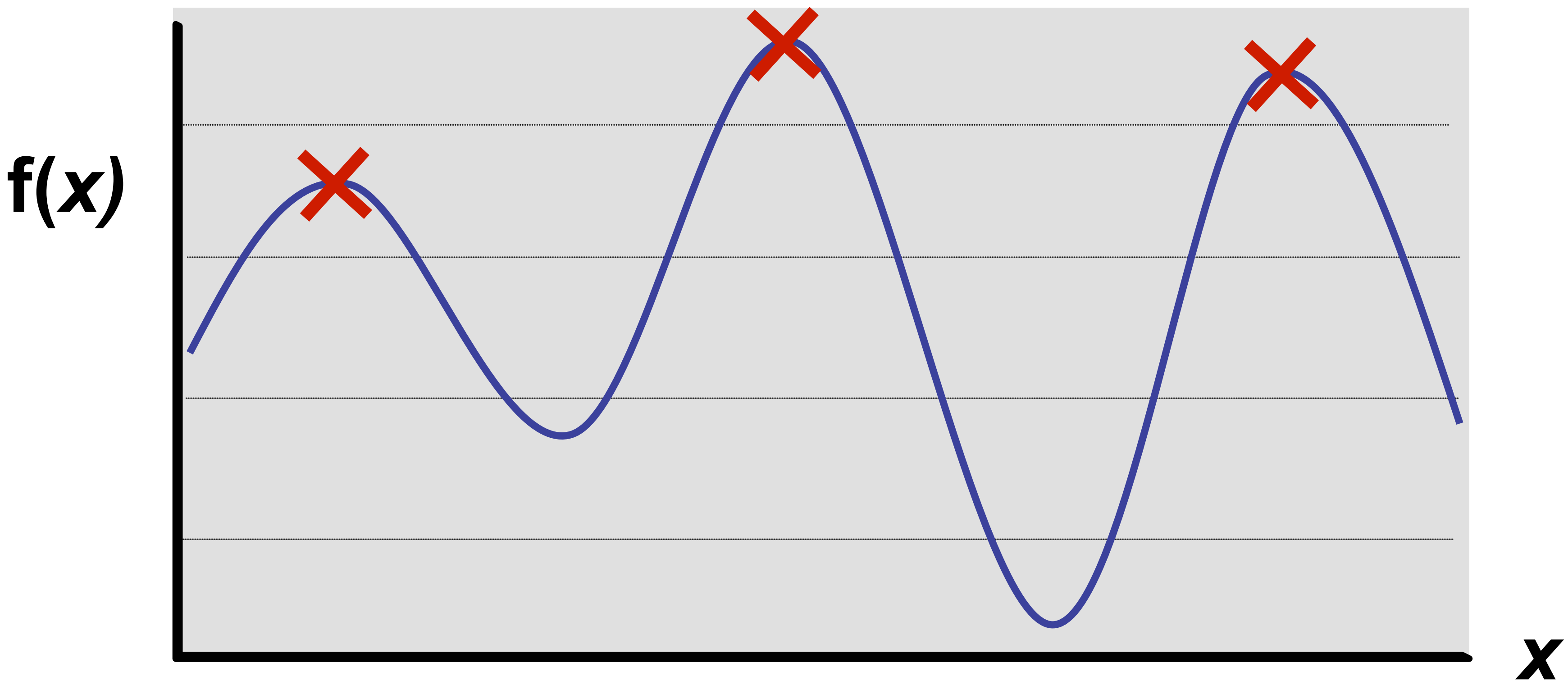
FindMax(A,n)

    return A[n]

Time Complexity: O(1)

Peak Finding

Input: Some function f(x)



Output: A local maximum

Peak Finding

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

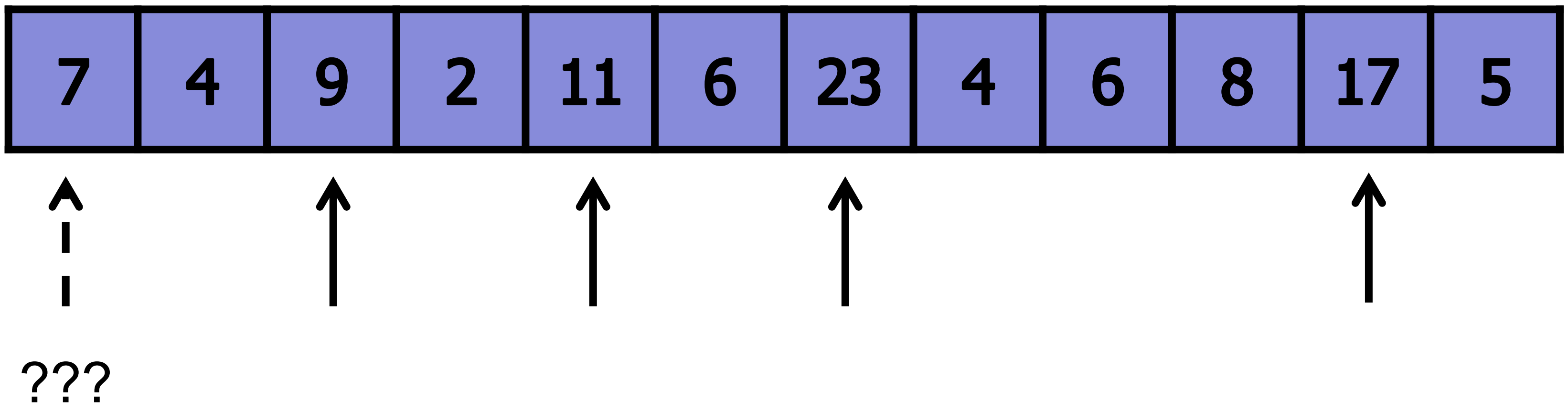
7	4	9	2	11	6	23	4	6	8	17	5

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]

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]

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]

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 $A[1..n]$

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

Running time: n

Simple improvement?

Peak Finding: Algorithm 1

Input: Some array A[1..n]

2	4	9	2	11	6	9	11	13	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**

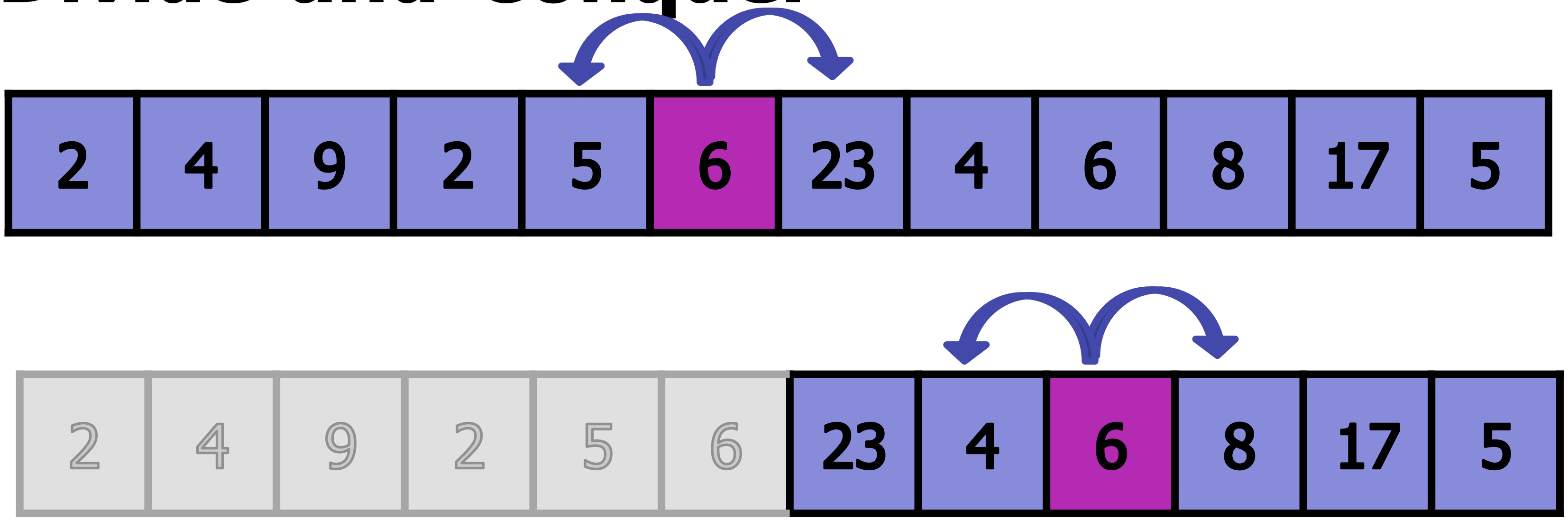
23 < 6? **NO**

Recurse!

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

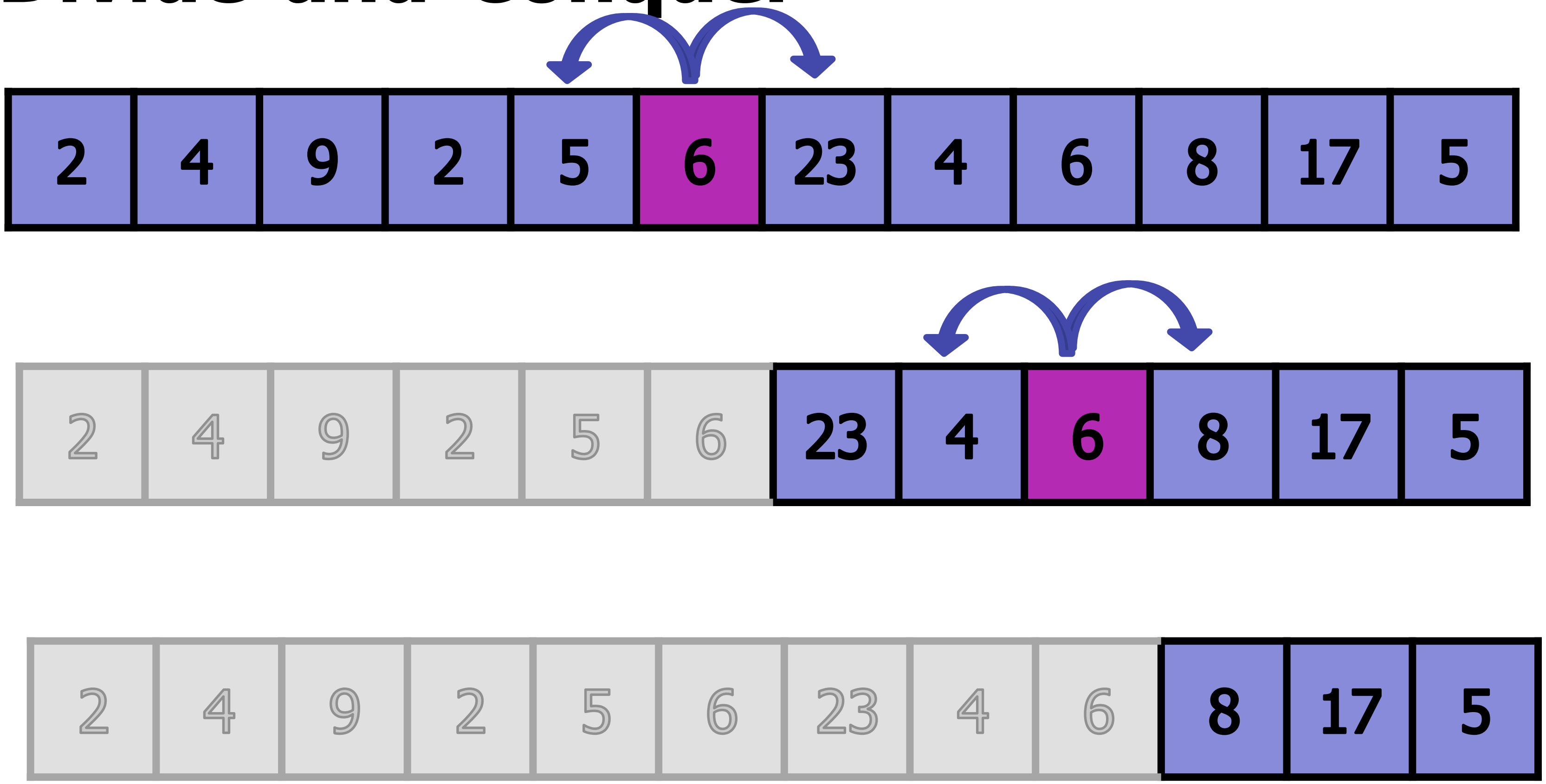
Peak Finding: Algorithm 2

Divide-and-Conquer



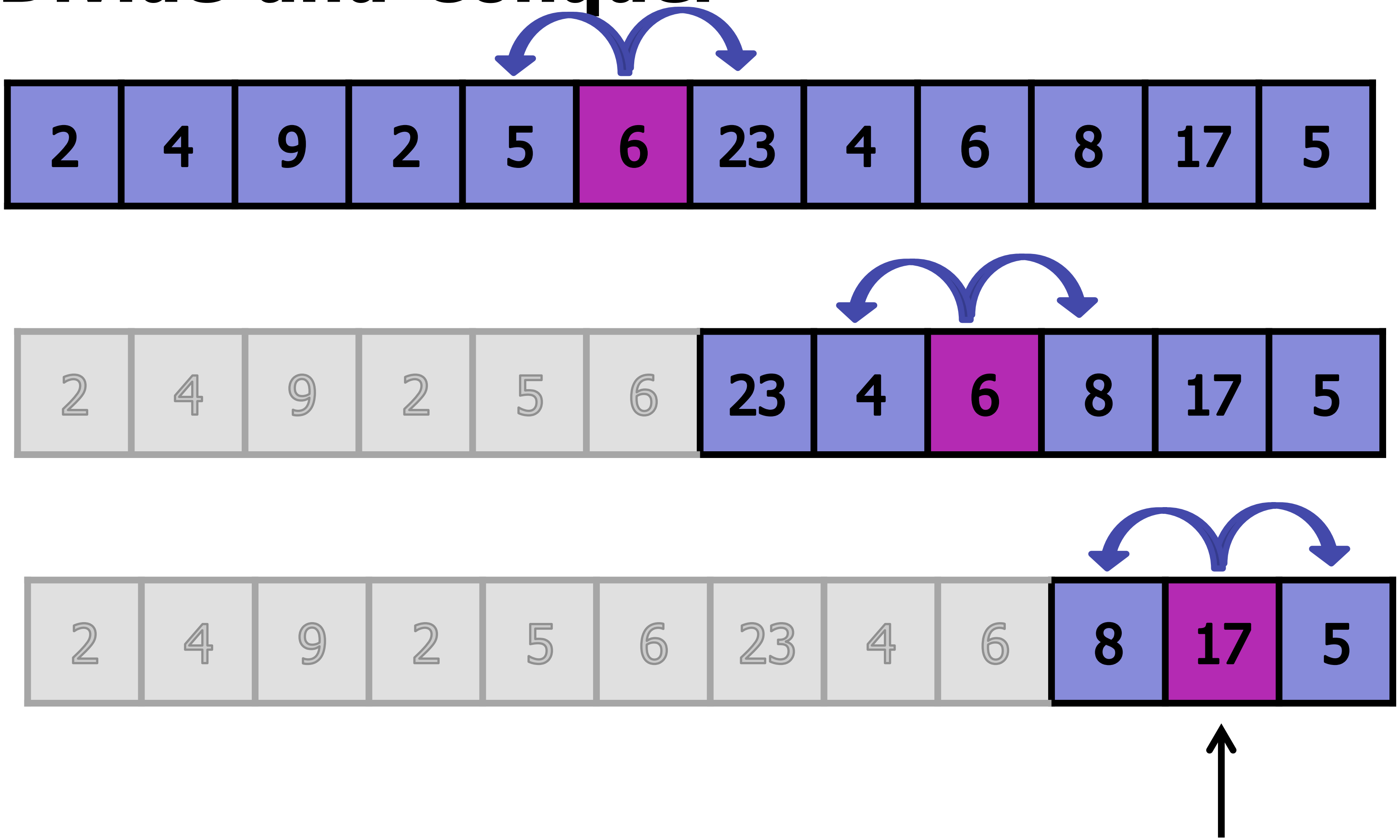
Peak Finding: Algorithm 2

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

Why?

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.

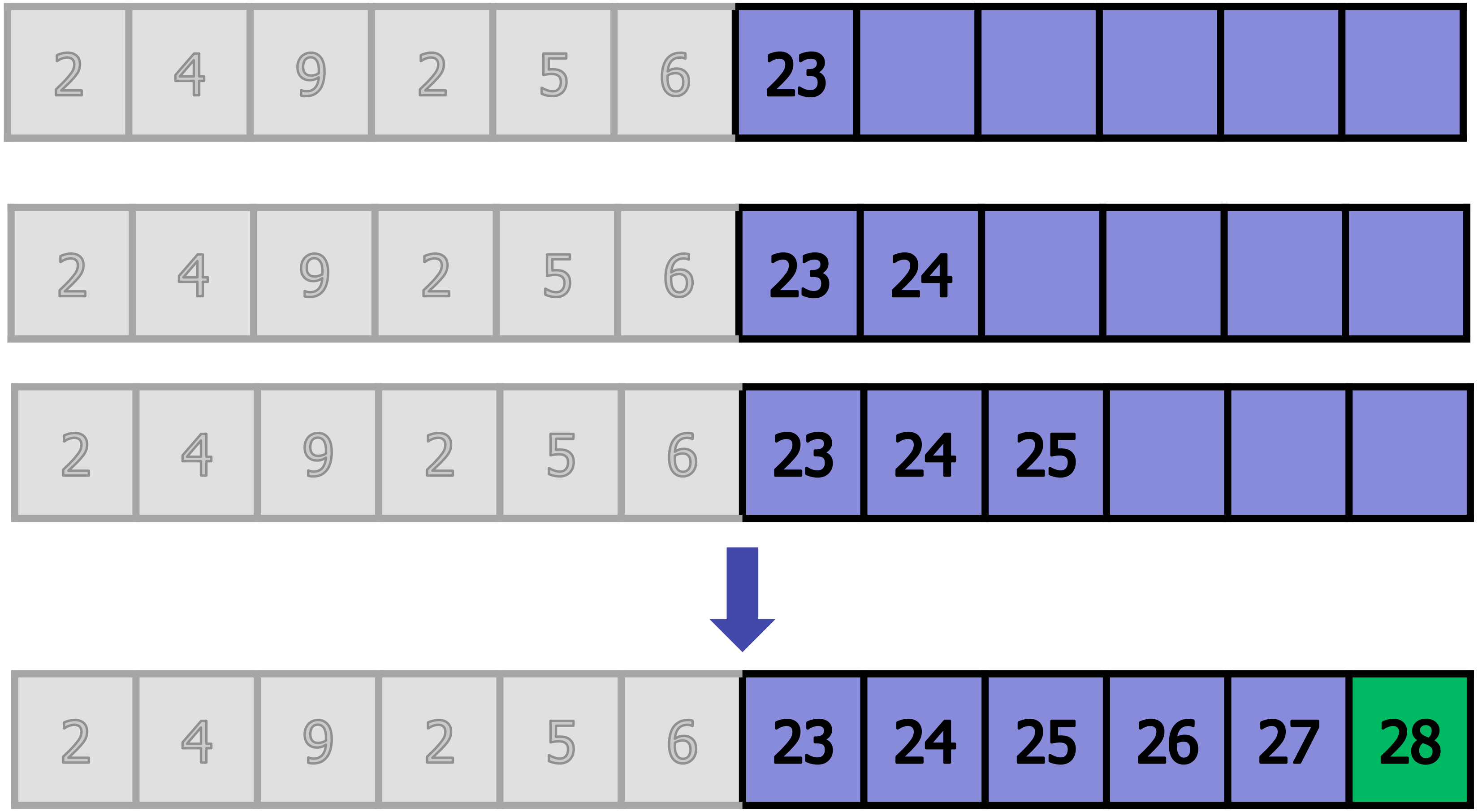
Explanation:

- 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 for comparing A[n/2] with neighbors

Time to find a peak in an array of size n

Recursion

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

Unrolling the recurrence:

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

Peak Finding: Algorithm 2

Unrolling the recurrence:

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

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

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

$$\ldots$$

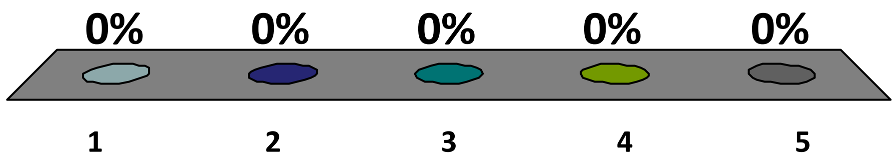
$$\ldots$$

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

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

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

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



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:

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