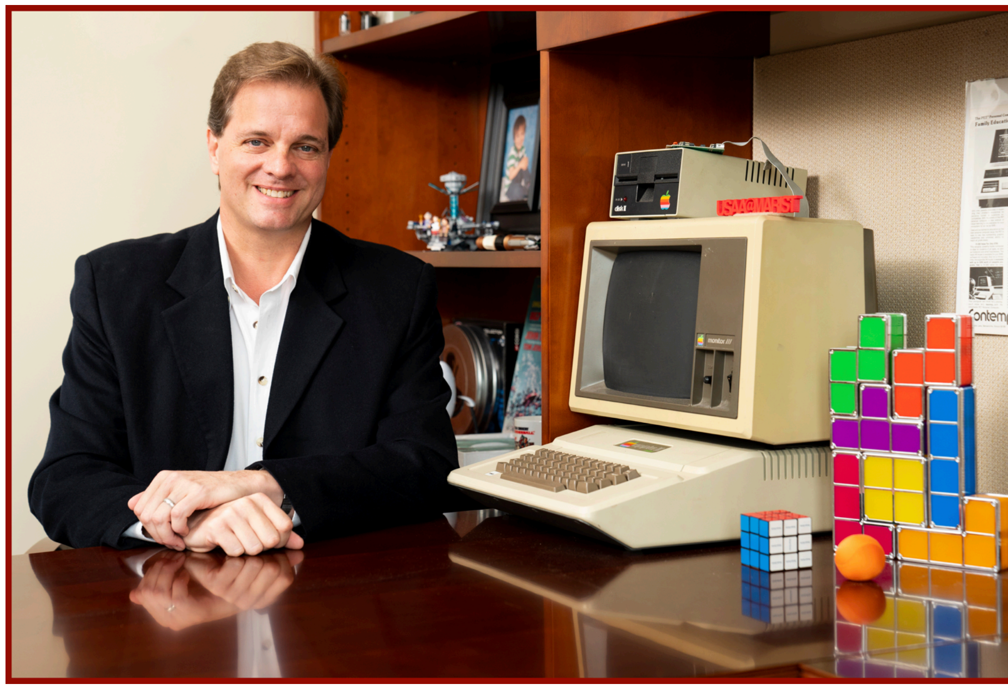
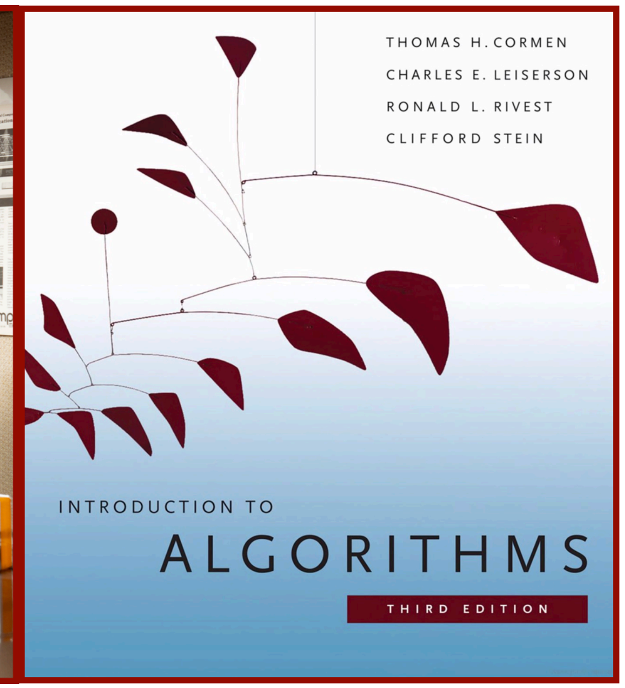
Sorting - part two





Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu

Divide and Conquer

Take a big problem and divide it into two smaller problems.

Take a those problems and divide them into two smaller problems.

Take a those problems and divide them into two smaller problems.

Take a those problems and divide them into two smaller problems.

Take a those problems and divide them into two smaller problems.

Take a those problems and divide them into two smaller problems.

Take a those problems and divide them into two smaller problems.

… until the problems get small enough that they are solved.

Then …

combine the smaller solutions into larger solutions

combine the smaller solutions into larger solutions

combine the smaller solutions into larger solutions

combine the smaller solutions into larger solutions

combine the smaller solutions into larger solutions

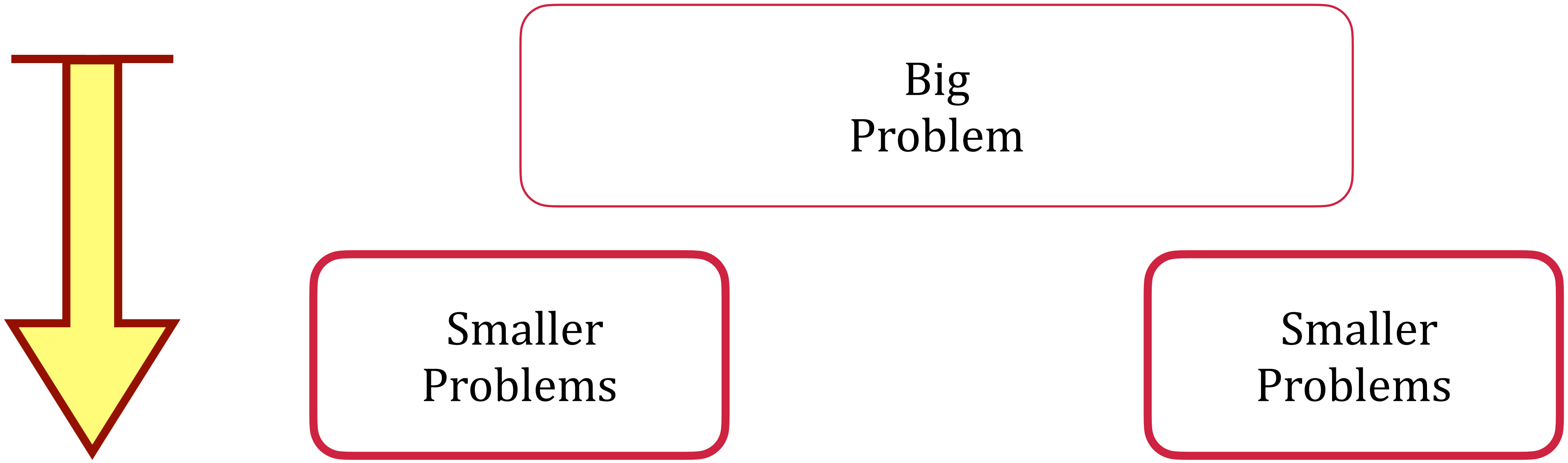
combine the smaller solutions into larger solutions

combine the smaller solutions into the complete solution

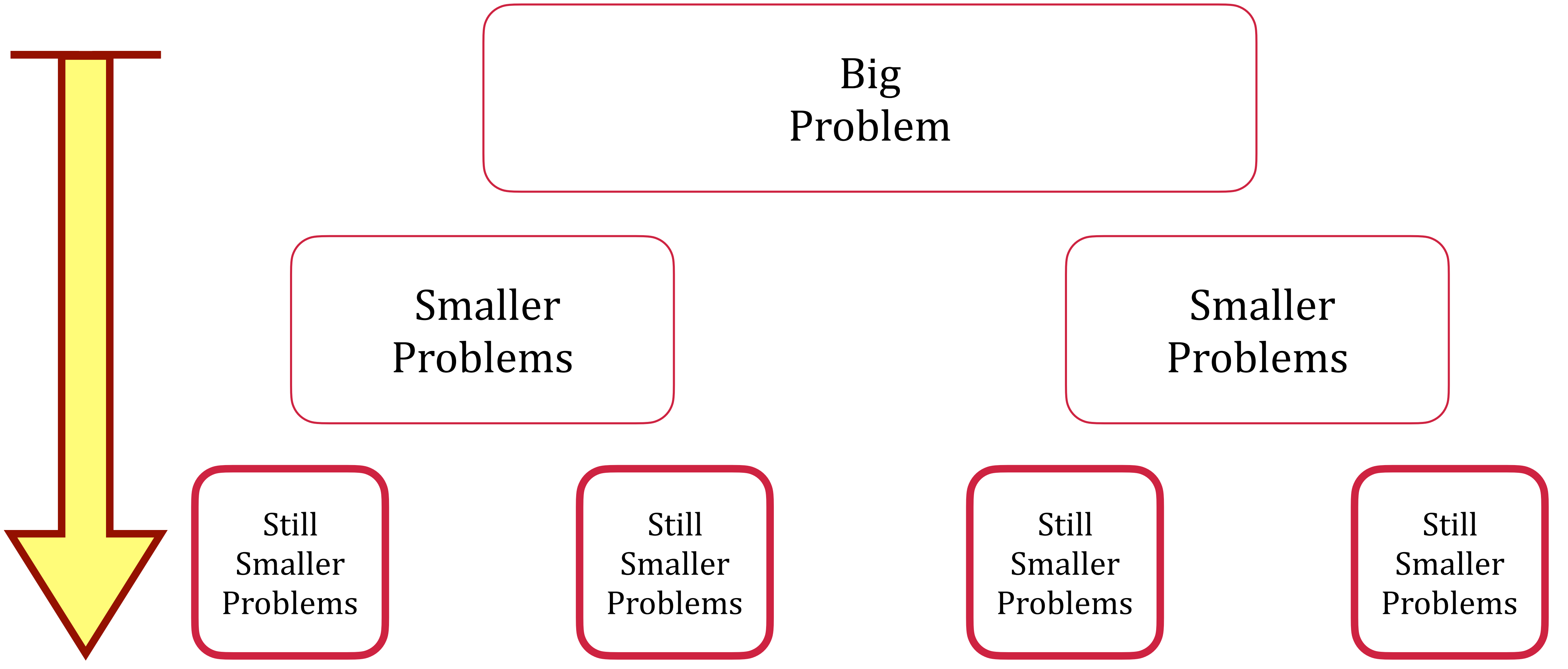
Divide and Conquer

Big
Problem

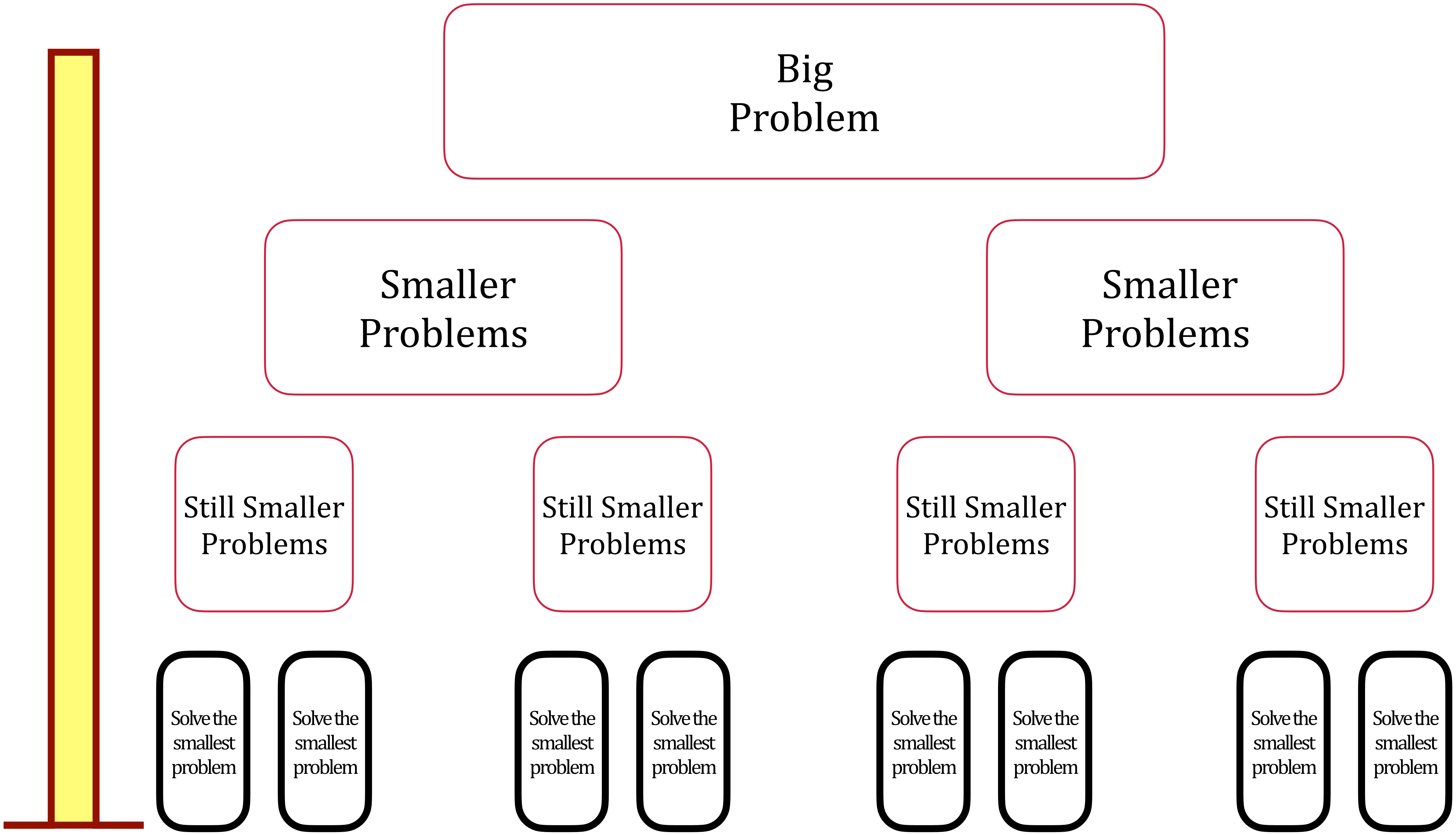
Divide and Conquer



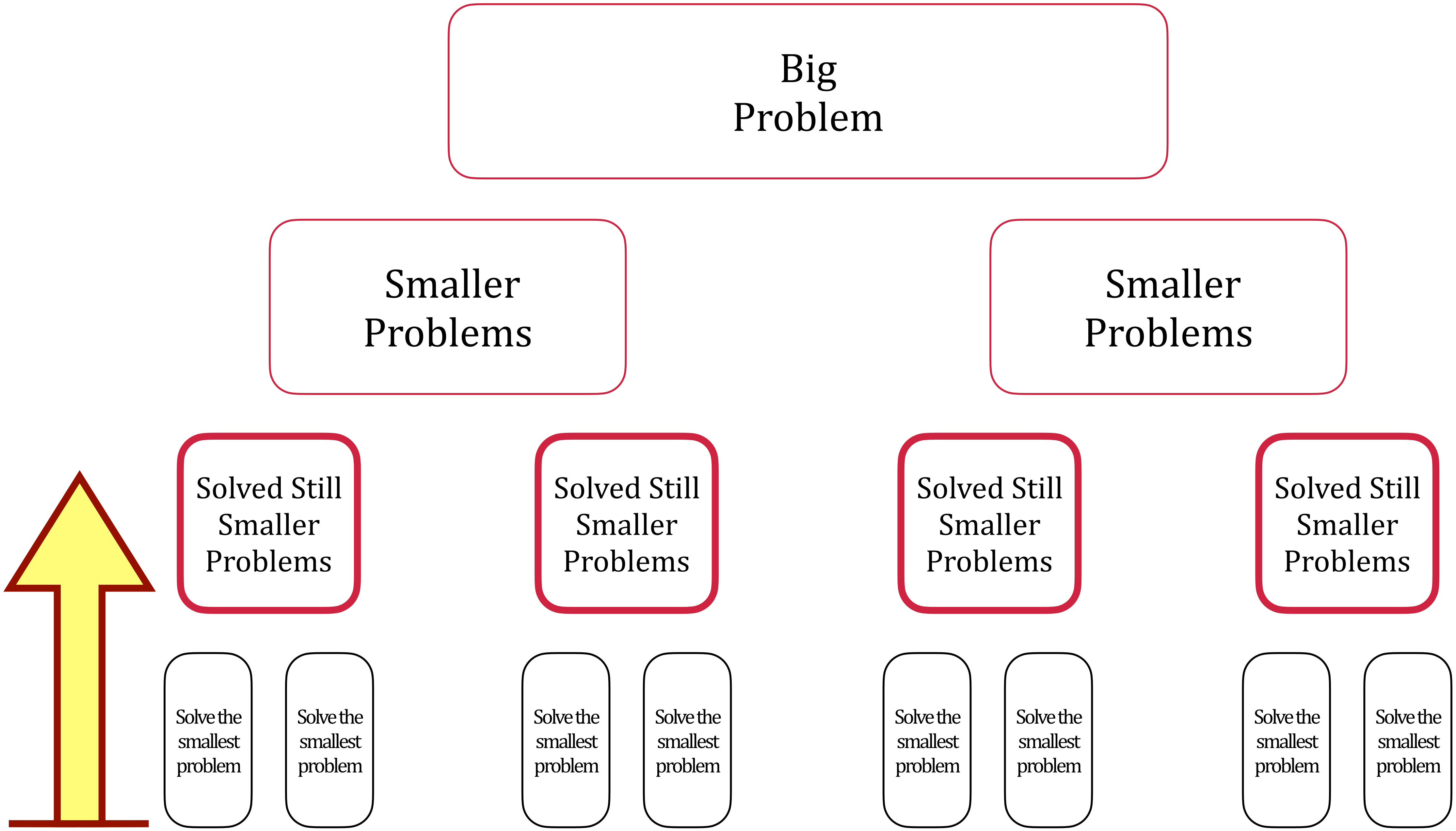
Divide and Conquer



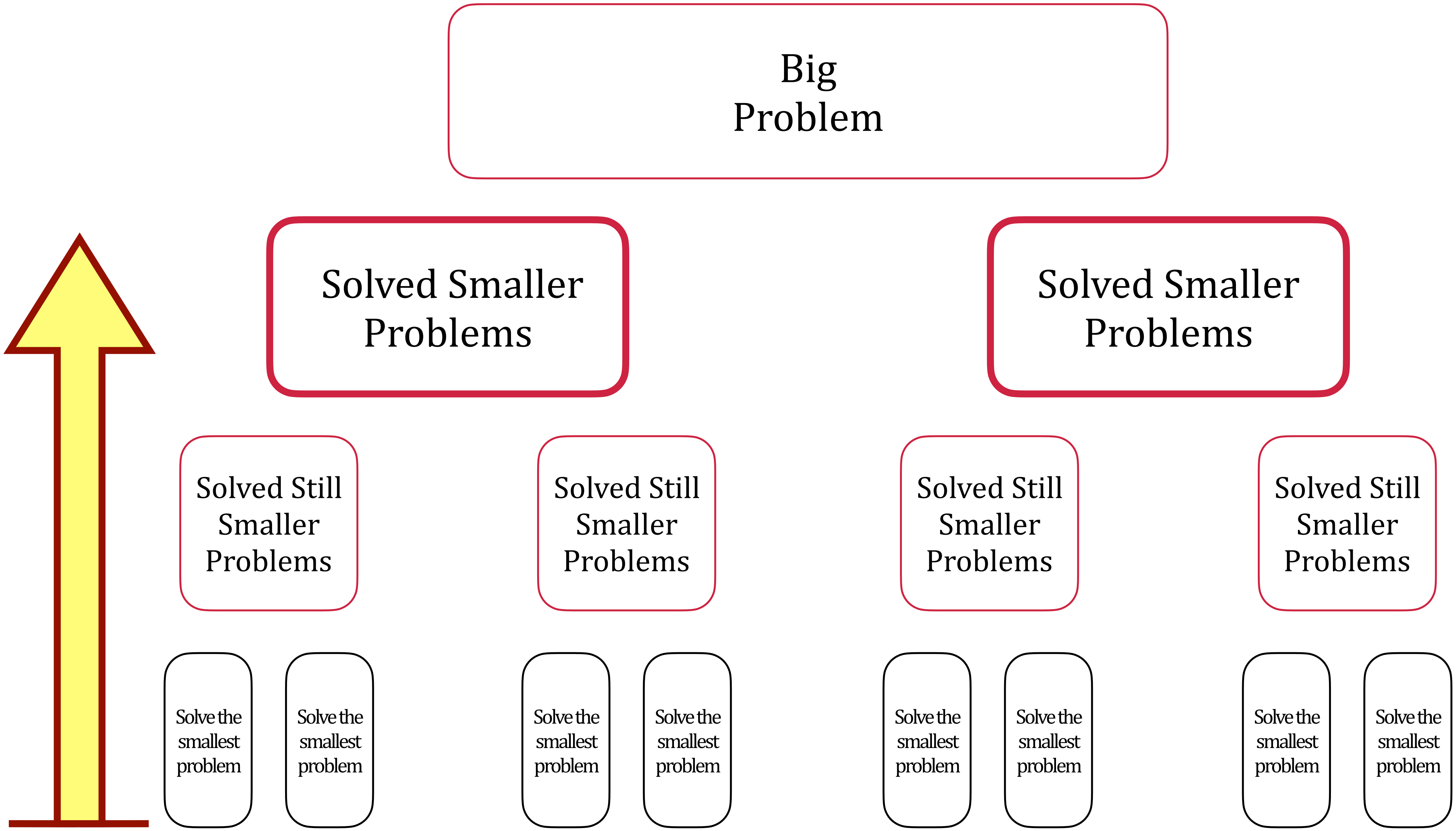
Divide and **Conquer**



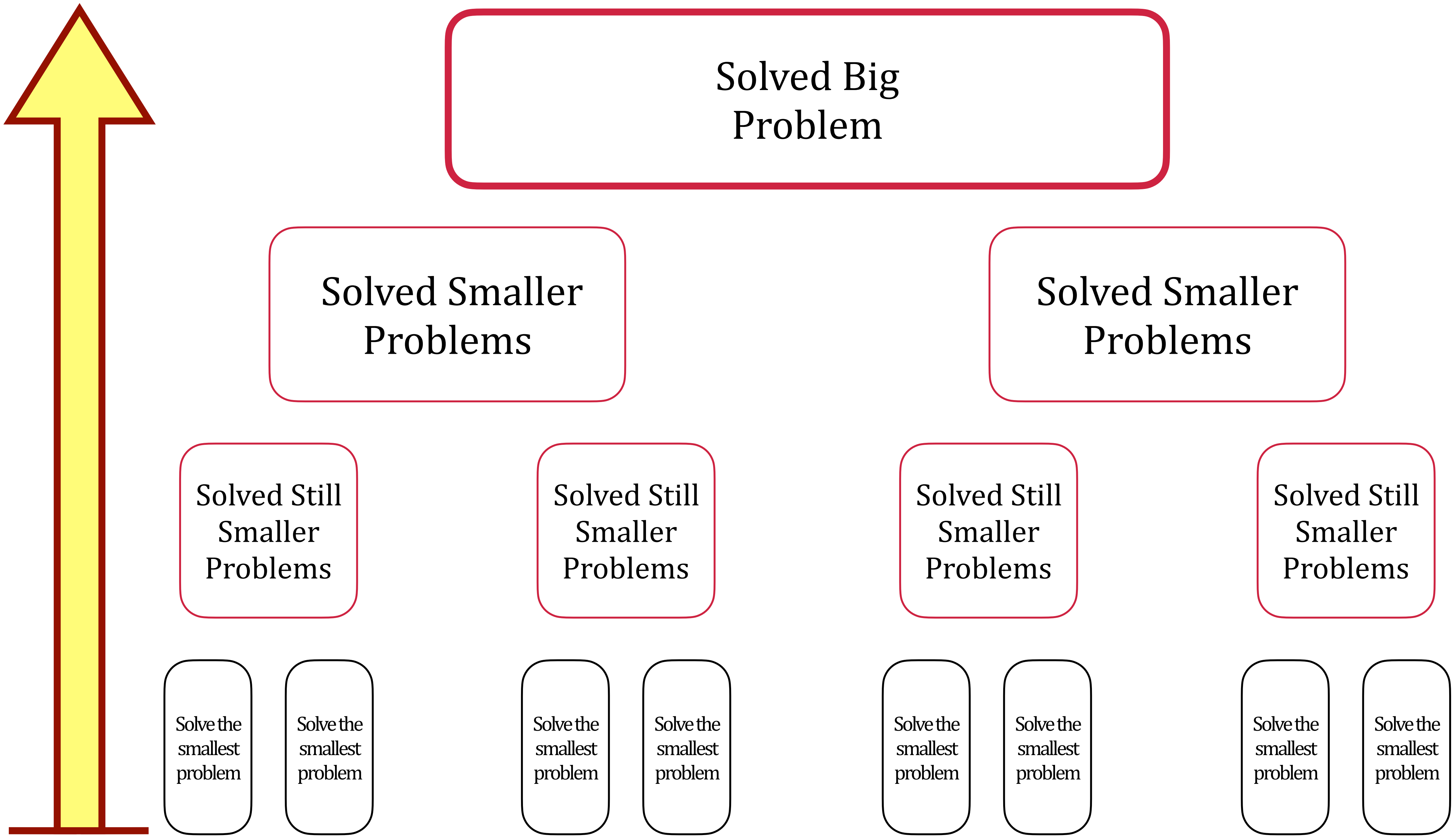
Divide and **Conquer**



Divide and **Conquer**



Divide and **Conquer**



Divide and Conquer :: Merge Sort

Divide and Conquer :: Merge Sort

Given an array that you what to sort . . .

Recursively **divide** the array into sub-arrays half the size until you have arrays of size 1. Note: an array of size 1 is sorted.

Then **conquer** by merging the (technically sorted) single-element arrays into progressively larger sorted sub-arrays as the recursion “unwinds”.

Divide and Conquer :: Merge Sort

Given an array that you what to sort . . .

Recursively **divide** the array into sub-arrays half the size until you have arrays of size 1. Note: an array of size 1 is sorted.

Then **conquer** by merging the (technically sorted) single-element arrays into progressively larger sorted sub-arrays as the recursion "unwinds".

What?

Divide and Conquer :: Merge Sort

Given an array that you what to sort . . .

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
2	**8**	**7**	**1**	**3**	**6**	**5**	**4**

Divide and Conquer :: Merge Sort

Recursively **divide** the array into sub-arrays half the size

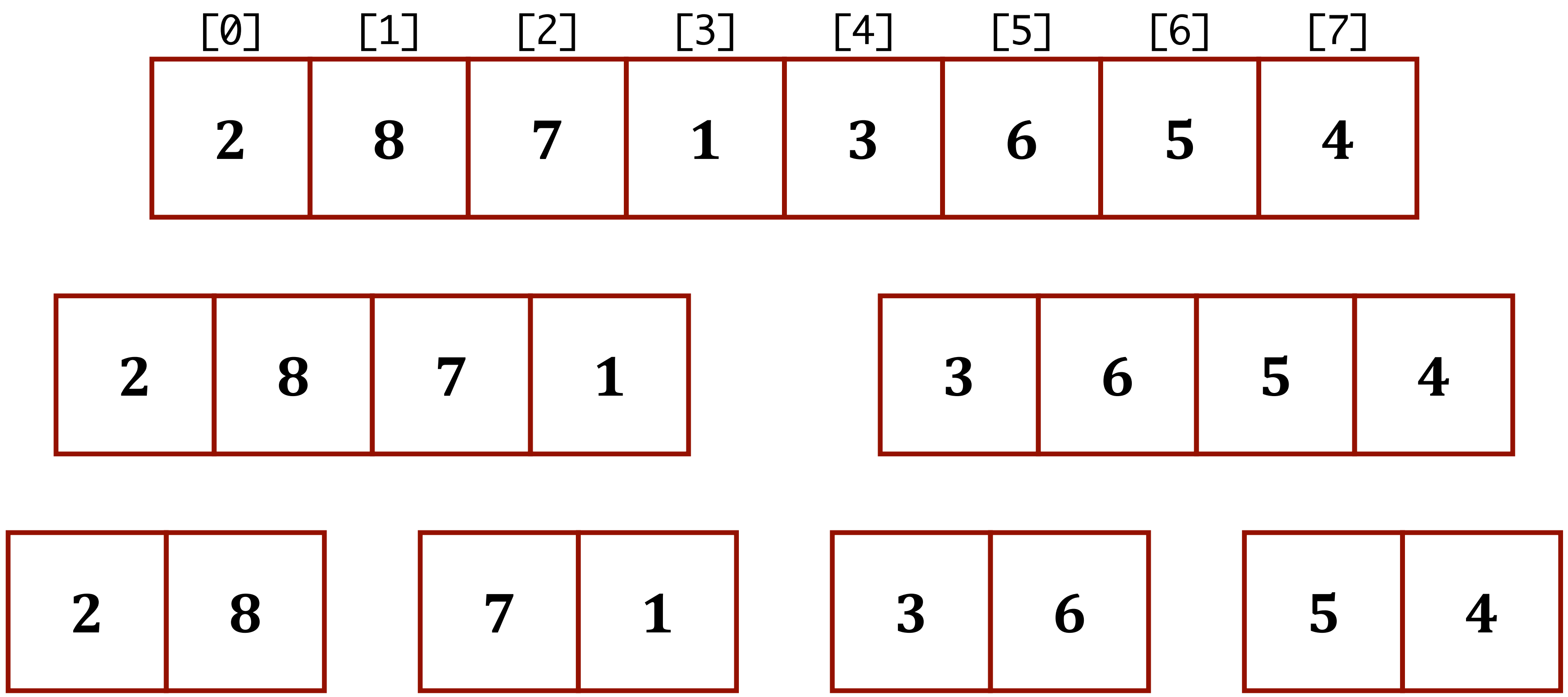
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
2	**8**	**7**	**1**	**3**	**6**	**5**	**4**

2	**8**	**7**	**1**

3	**6**	**5**	**4**

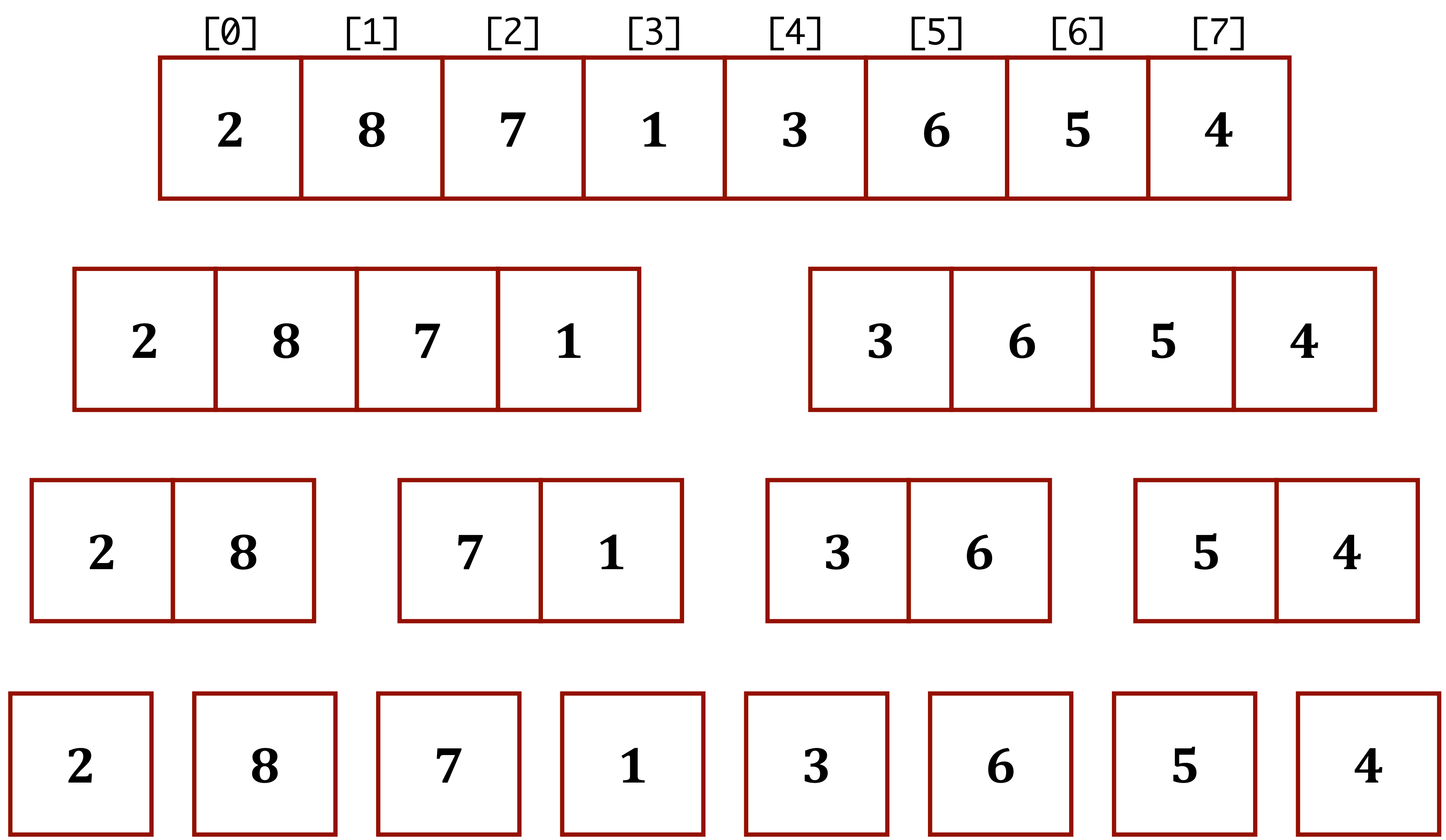
Divide and Conquer :: Merge Sort

Recursively **divide** the array into sub-arrays half the size . .



Divide and Conquer :: Merge Sort

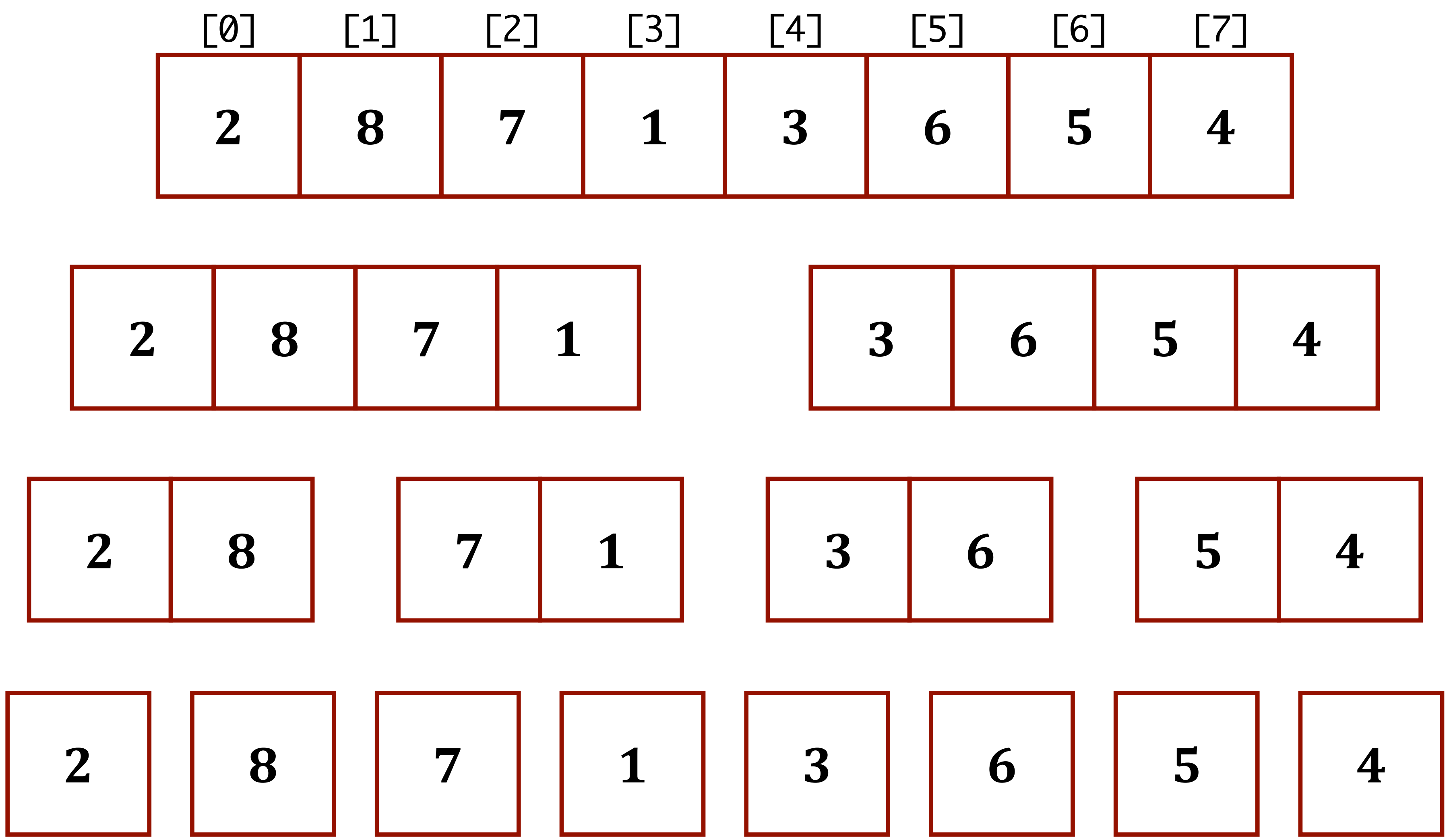
Recursively **divide** the array into sub-arrays half the size . . .



. . . until you have arrays of size 1. (Arrays of size 1 are sorted.)

Divide and Conquer :: Merge Sort

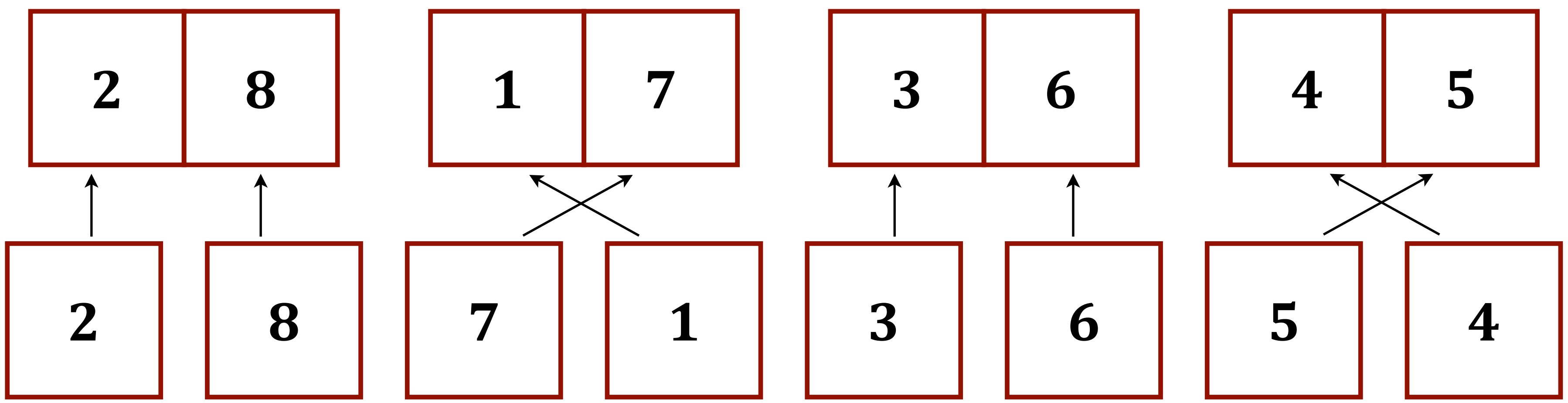
Recursively **divide** the array into sub-arrays half the size . . .



. . . until you have arrays of size 1. (Arrays of size 1 are sorted.)

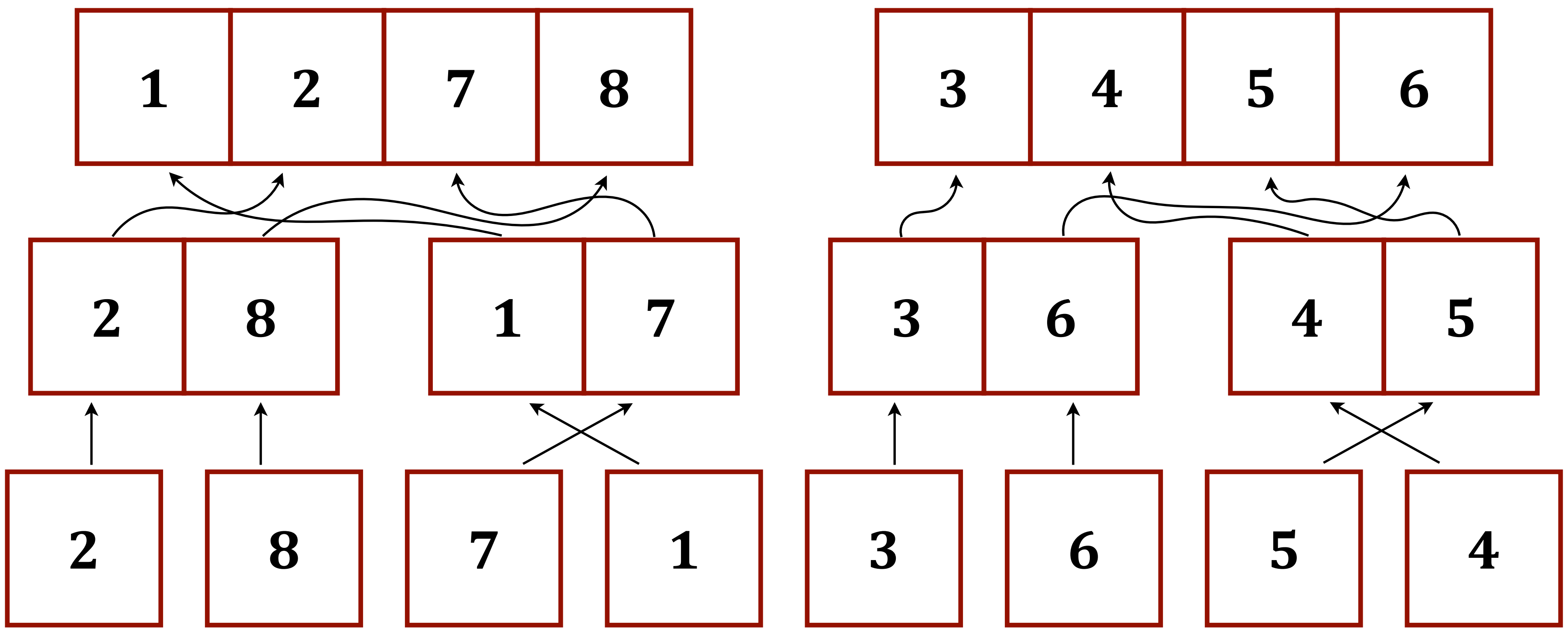
How many times did we “**divide**” (in terms of the *n* items to sort)?

Divide and Conquer :: Merge Sort



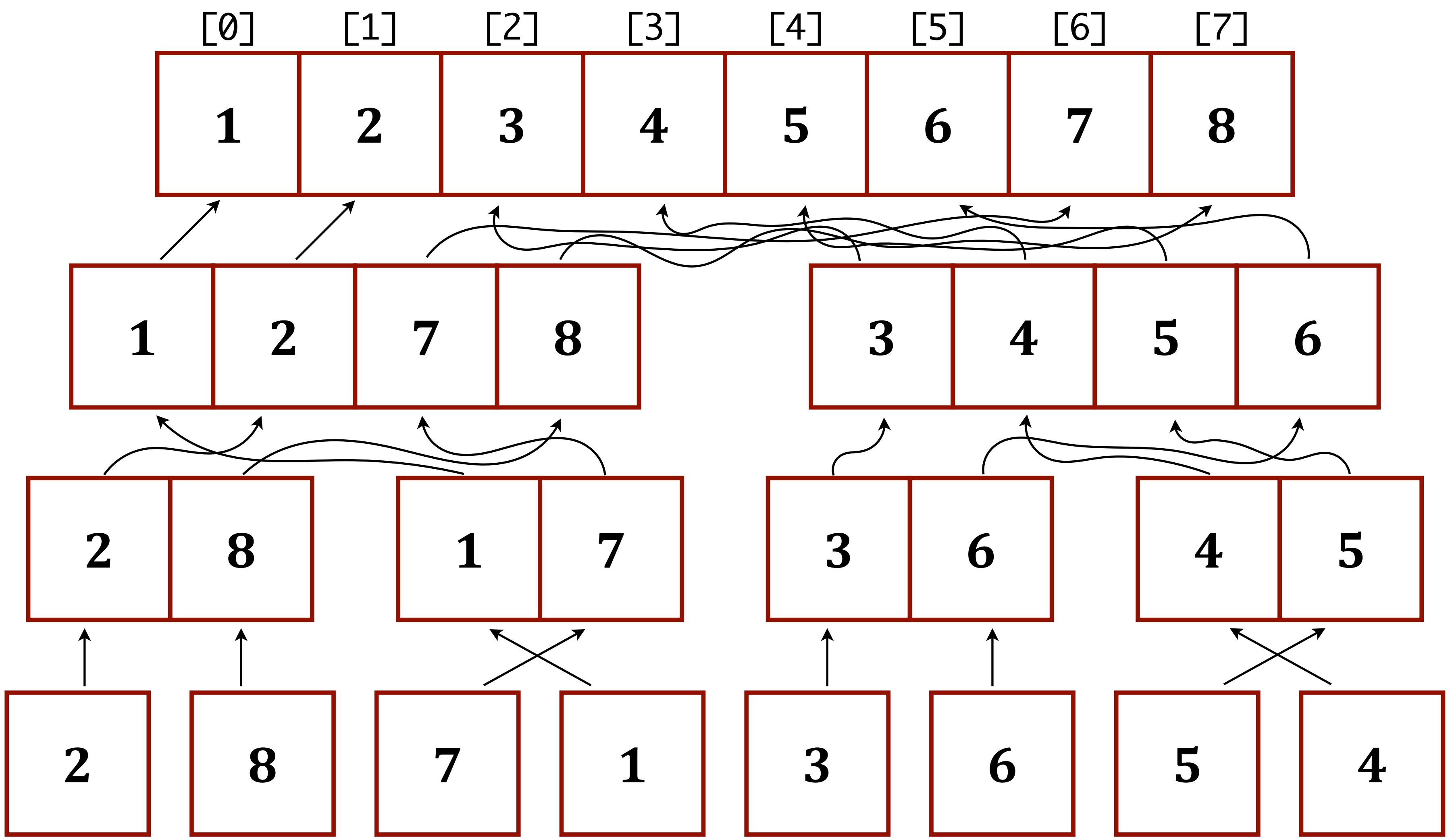
Conquer by merging the sub-arrays into progressively larger **sorted** arrays .

Divide and Conquer :: Merge Sort



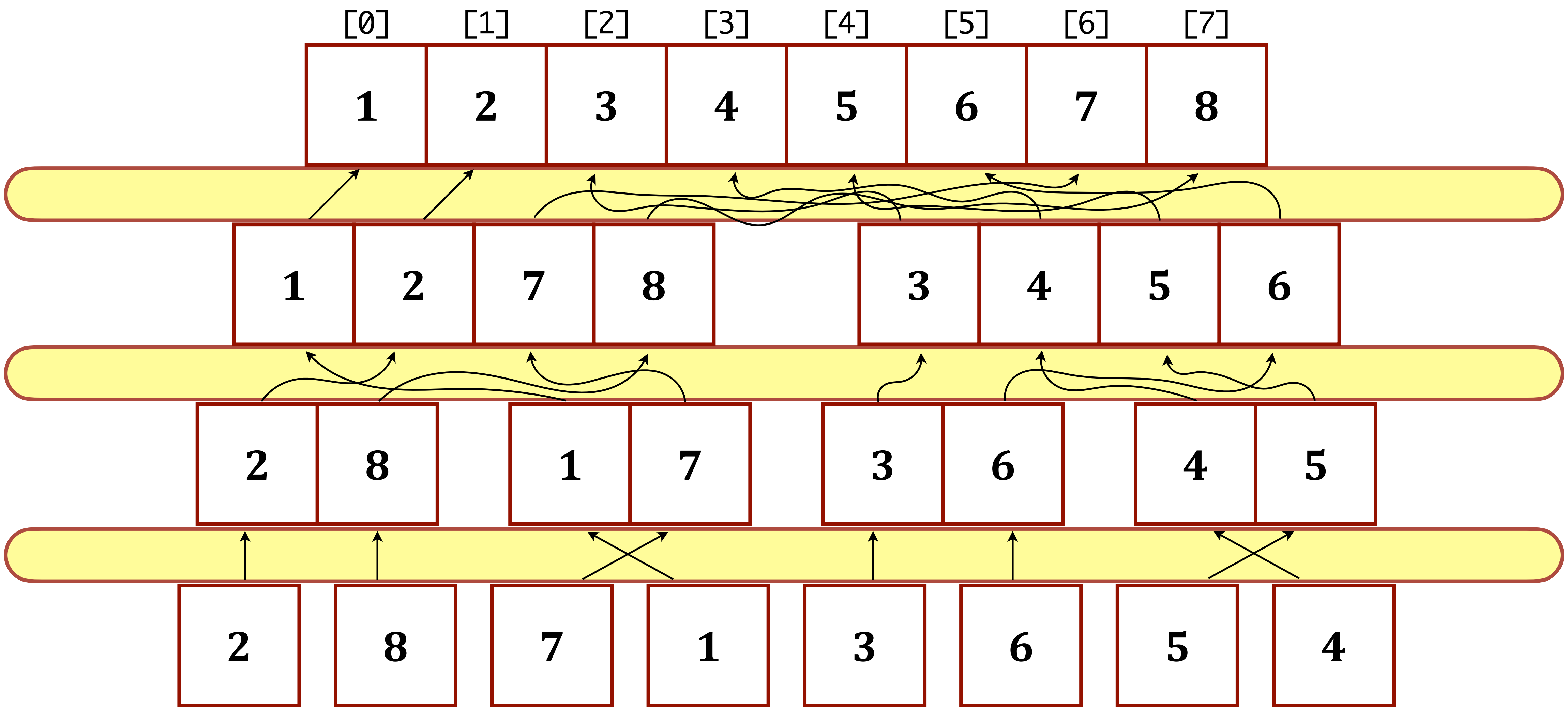
Conquer by merging the sub-arrays into progressively larger **sorted** arrays . .

Divide and Conquer :: Merge Sort



Conquer by merging the sub-arrays into progressively larger **sorted** arrays . . . until the entire thing is sorted.

Divide and Conquer :: Merge Sort



The sorting work is done in the **merge** steps.

How long does each **merge** step this take in terms of n?

Divide and Conquer :: Merge Sort

```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])        ⟨⟨Recurse!⟩⟩
      MERGESORT(A[m+1..n])      ⟨⟨Recurse!⟩⟩
      MERGE(A[1..n], m)
```

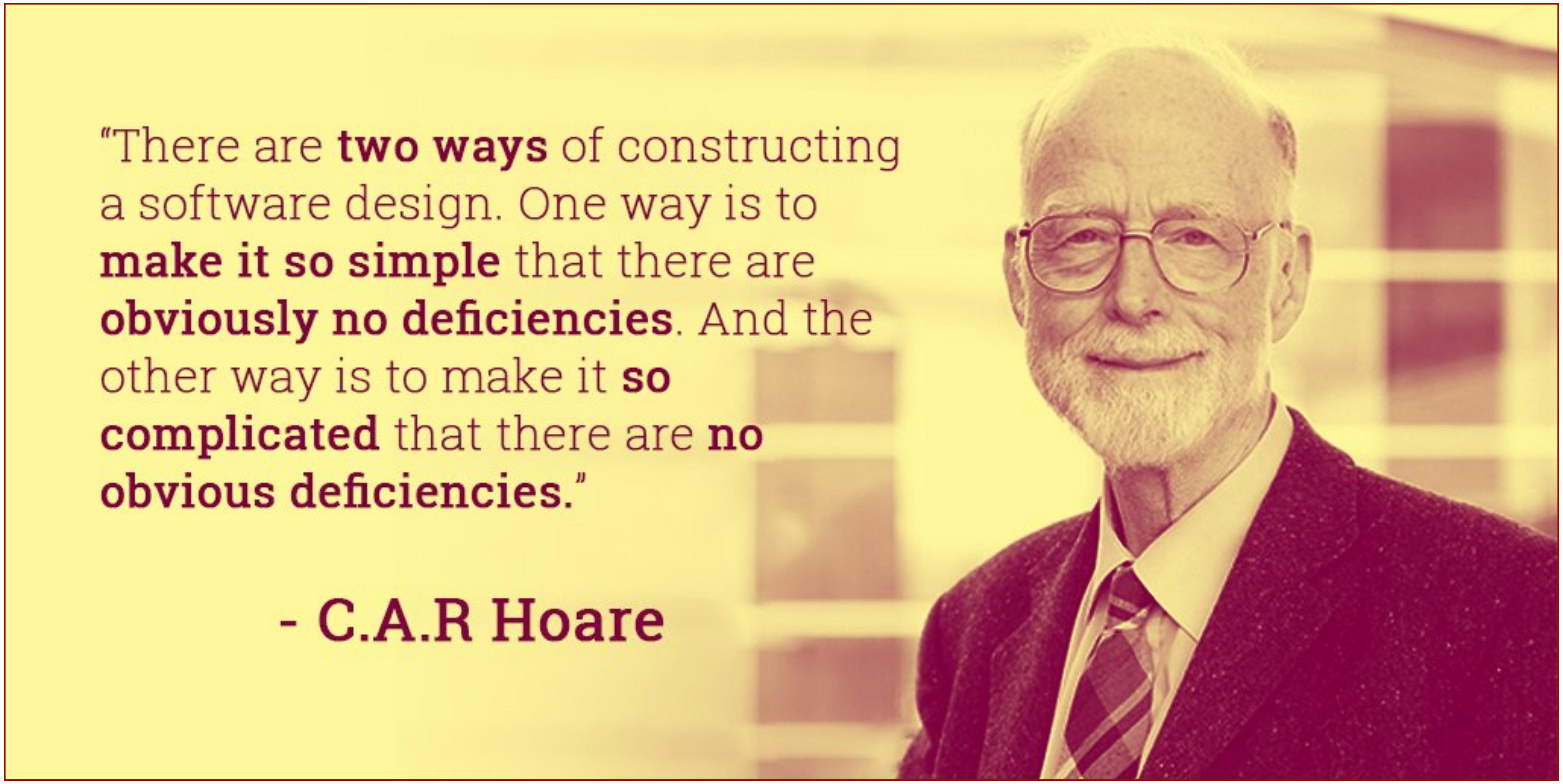
```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i+1
      else if i > m
          B[k] ← A[j]; j ← j+1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i+1
      else
          B[k] ← A[j]; j ← j+1

  for k ← 1 to n
      A[k] ← B[k]
```

Figure 1.6. Mergesort

from the Jeff Erikson Algorithms book, linked on our web site

How long does each **merge** step this take in terms of *n*?

Divide and Conquer :: Quick Sort



Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .

Recursively **divide** the array into halves — **conquering** by partitioning those halves around a “pivot” value — until the smallest sub-arrays are sorted.

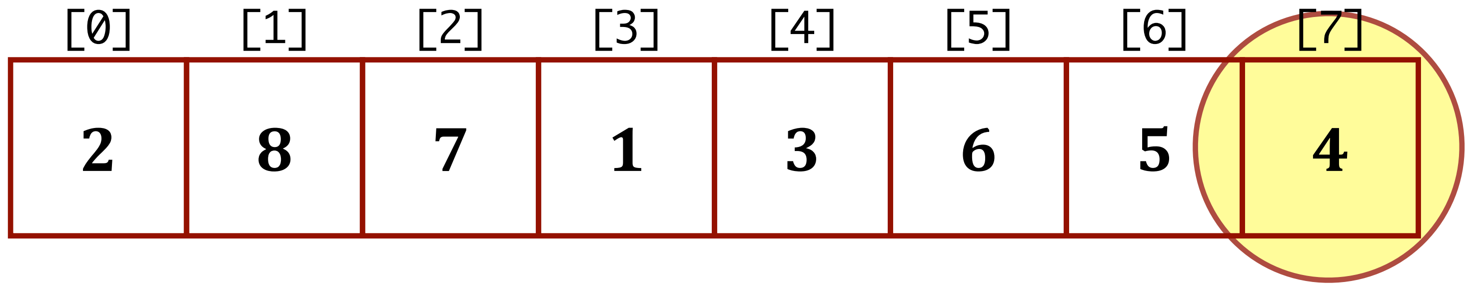
Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
2	**8**	**7**	**1**	**3**	**6**	**5**	**4**

Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .



Randomly select an index to provide the pivot value . . .

Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .

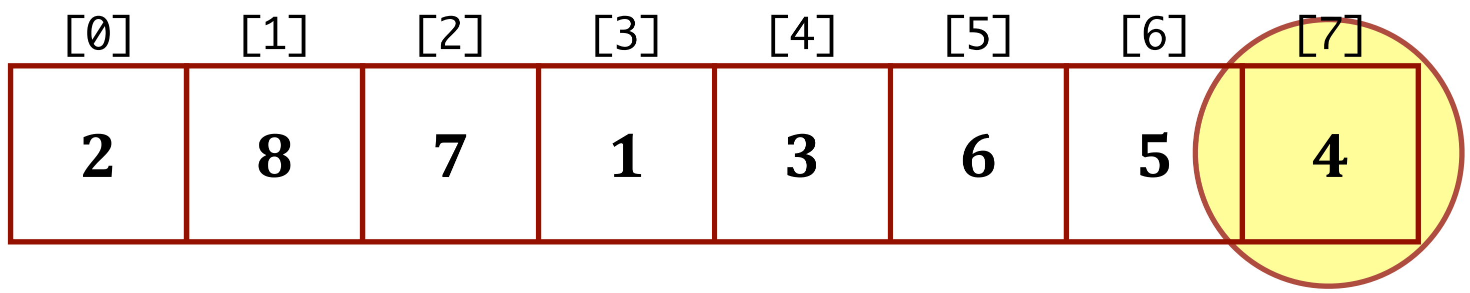
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
2	**8**	**7**	**1**	**3**	**6**	**5**	**4**

Randomly select an index to provide the pivot value . . . and **divide** the array into halves — **conquering** by partitioning those halves around a "pivot" value.

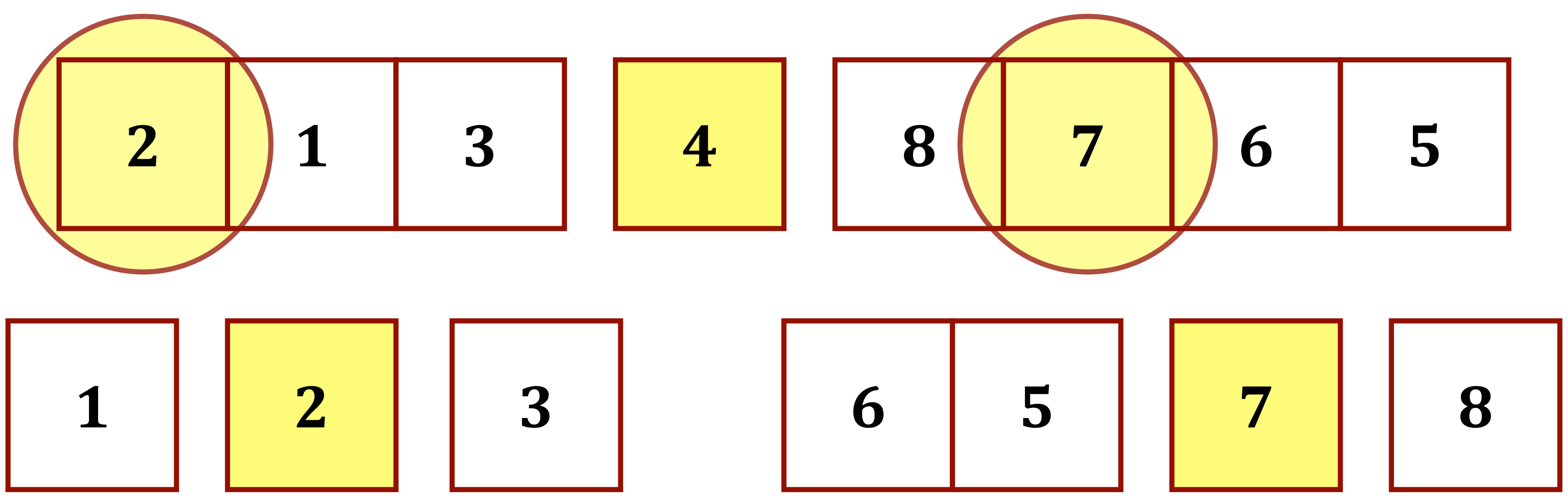
2	**1**	**3**		**4**		**8**	**7**	**6**	**5**

Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .

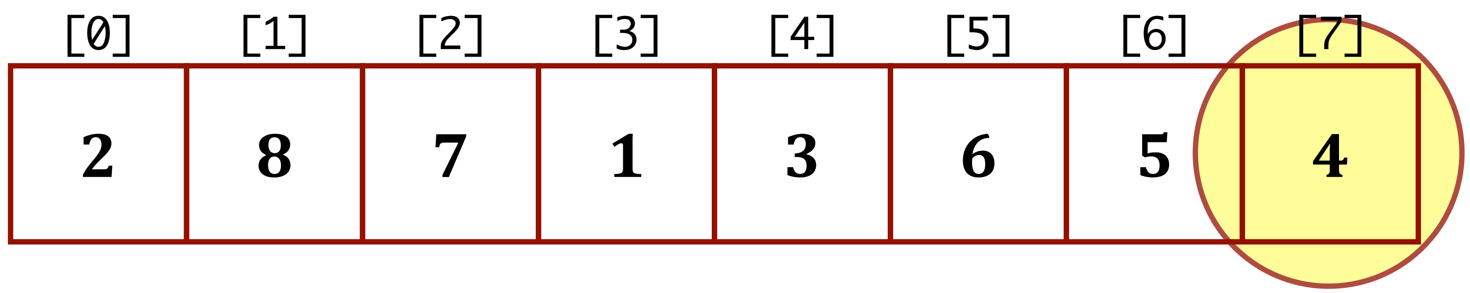


Randomly select an index to provide the pivot value . . . and **divide** the array into halves — **conquering** by partitioning those halves around a “pivot” value.

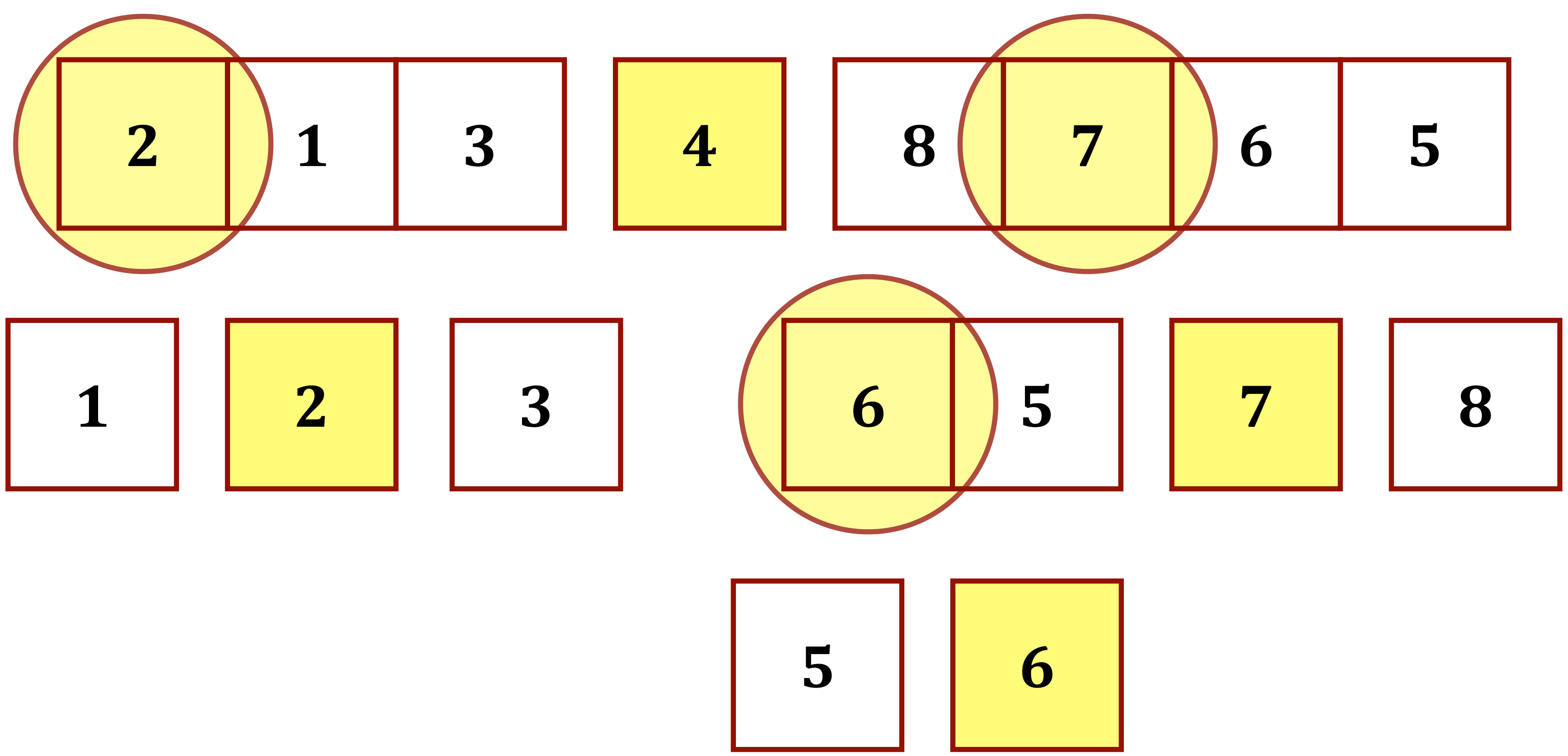


Divide and Conquer :: Quick Sort

Given an array that you what to sort . . .



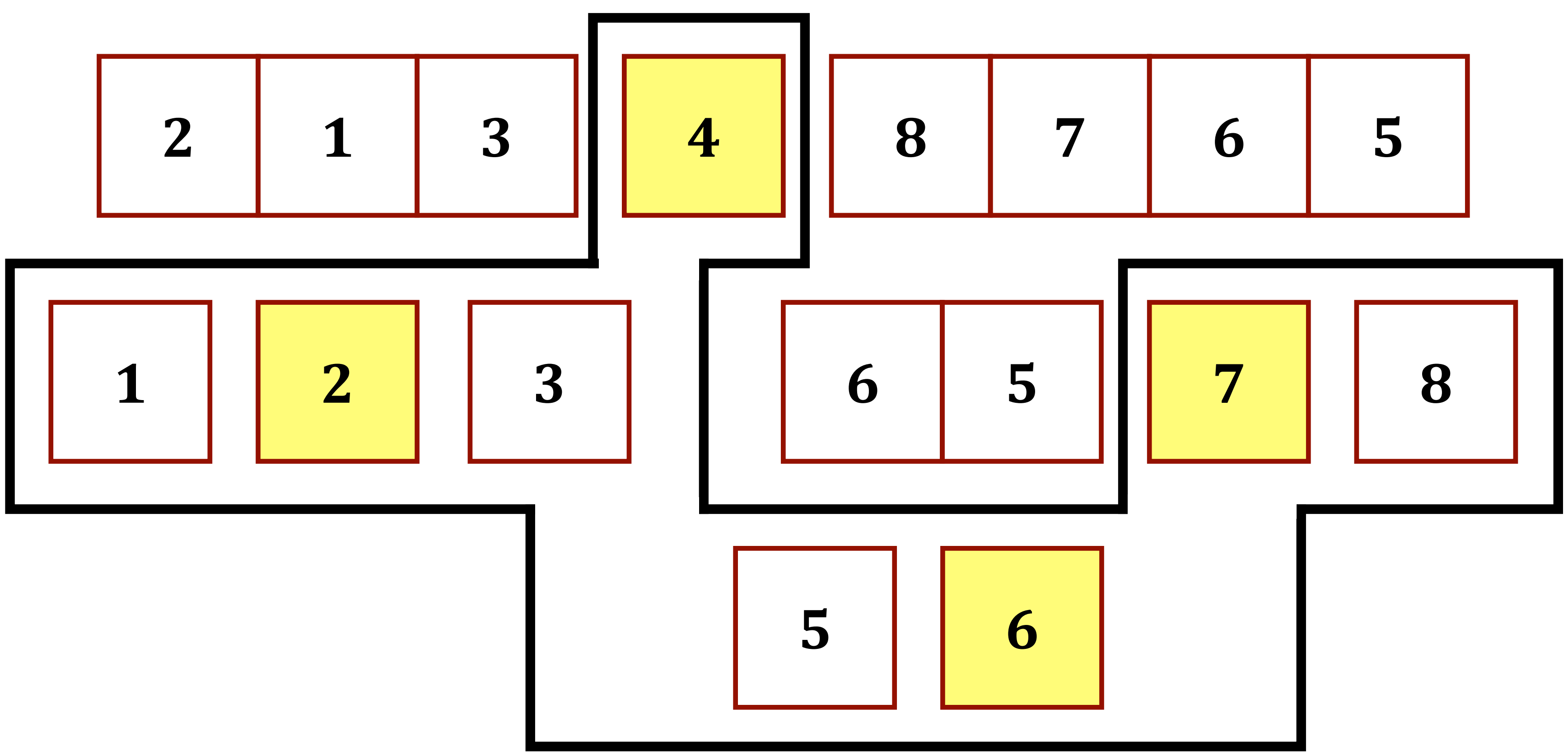
Randomly select an index to provide the pivot value . . . and **divide** the array into halves — **conquering** by partitioning those halves around a "pivot" value.



Divide and Conquer :: Quick Sort

We are done when all the sub-arrays are of size 1.

The sorting work is done in the **partition** steps.

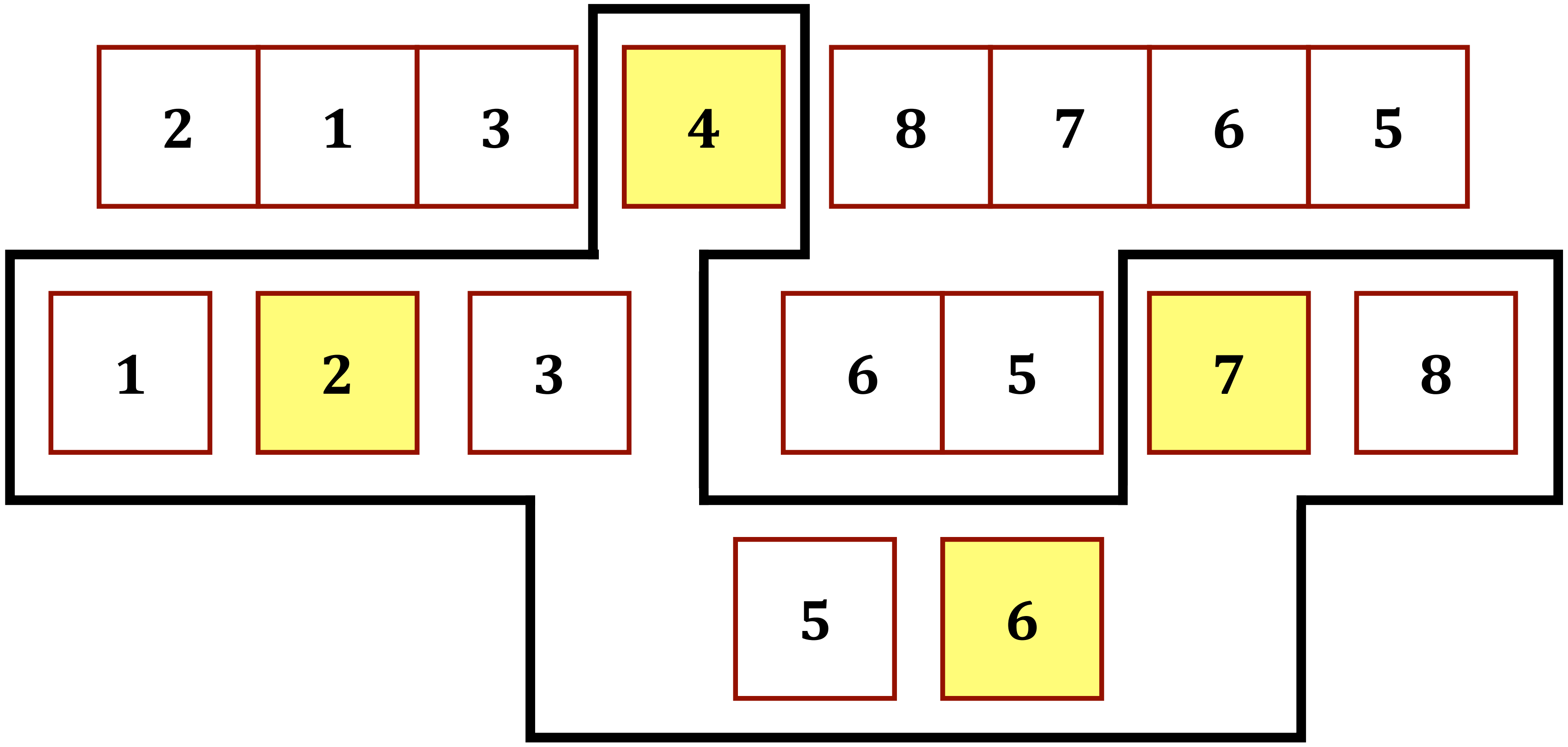


Divide and Conquer :: Quick Sort

We are done when all the sub-arrays are of size 1.

The sorting work is done in the **partition** steps.

How long does each partition step take, and how many times do we do it?



Divide and Conquer :: Quick Sort

```
QUICKSORT(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← PARTITION(A, p)
      QUICKSORT(A[1..r − 1])     ⟨⟨Recurse!⟩⟩
      QUICKSORT(A[r + 1..n])     ⟨⟨Recurse!⟩⟩
```

```
PARTITION(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                  ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n − 1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ + 1]
  return ℓ + 1
```

Figure 1.8. Quicksort

from the Jeff Erikson Algorithms book, linked on our web site

How long does each **partition** step this take in terms of n?

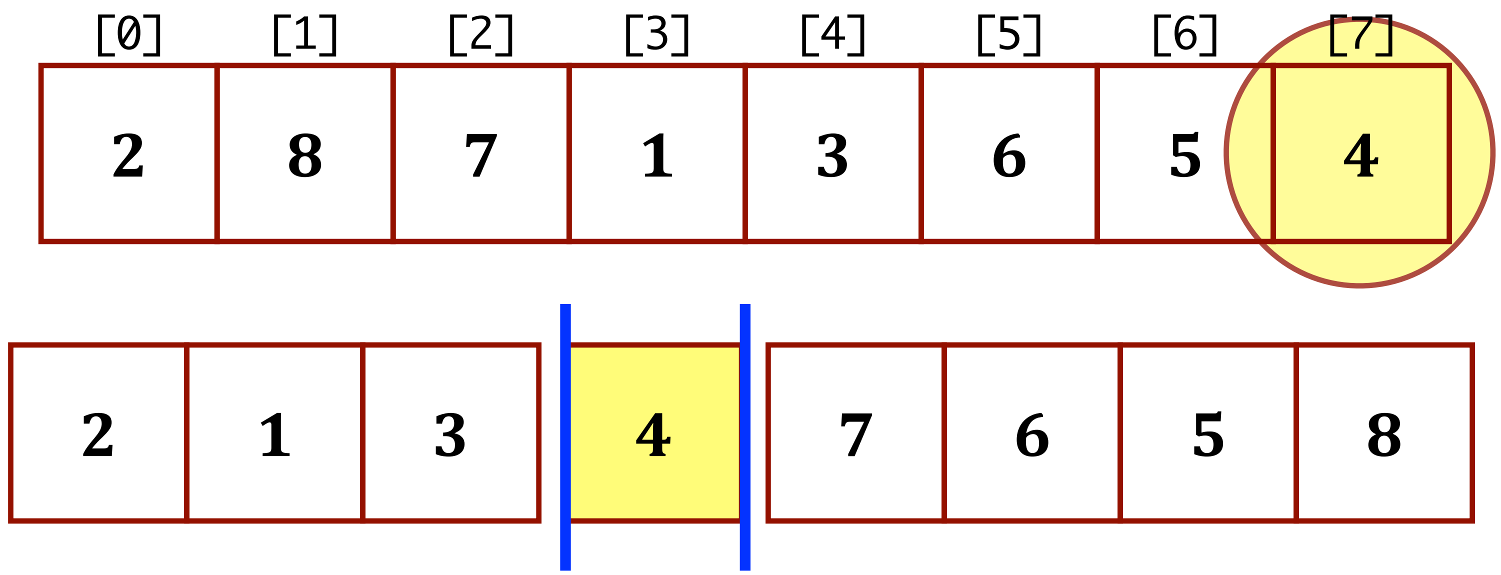
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
2	**8**	**7**	**1**	**3**	**6**	**5**	**4**

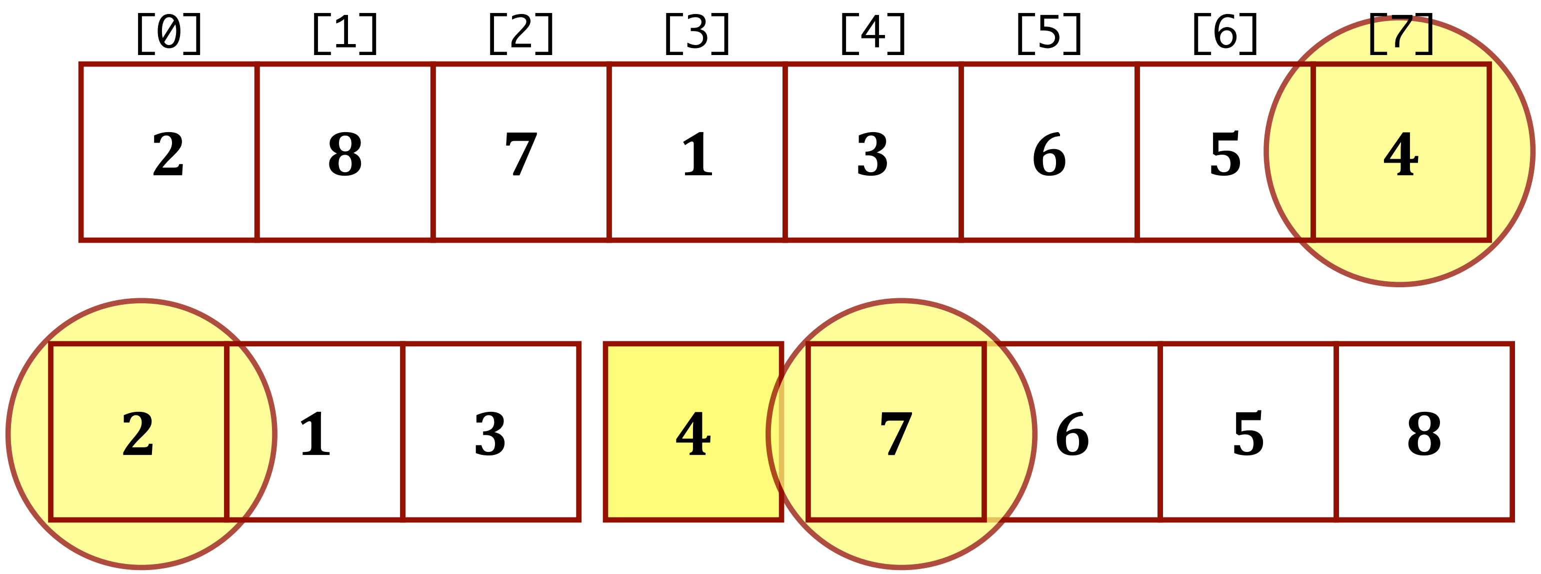
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



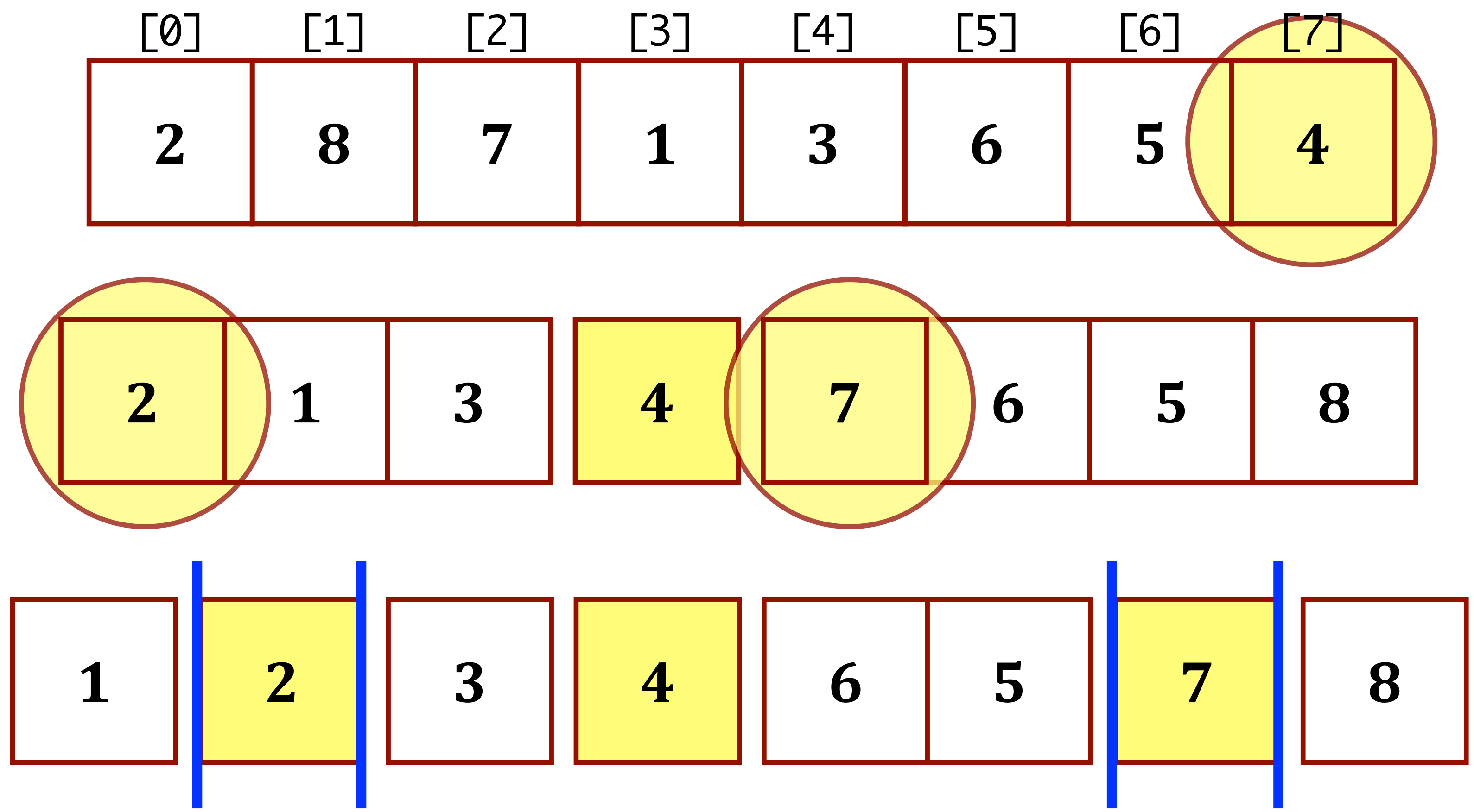
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



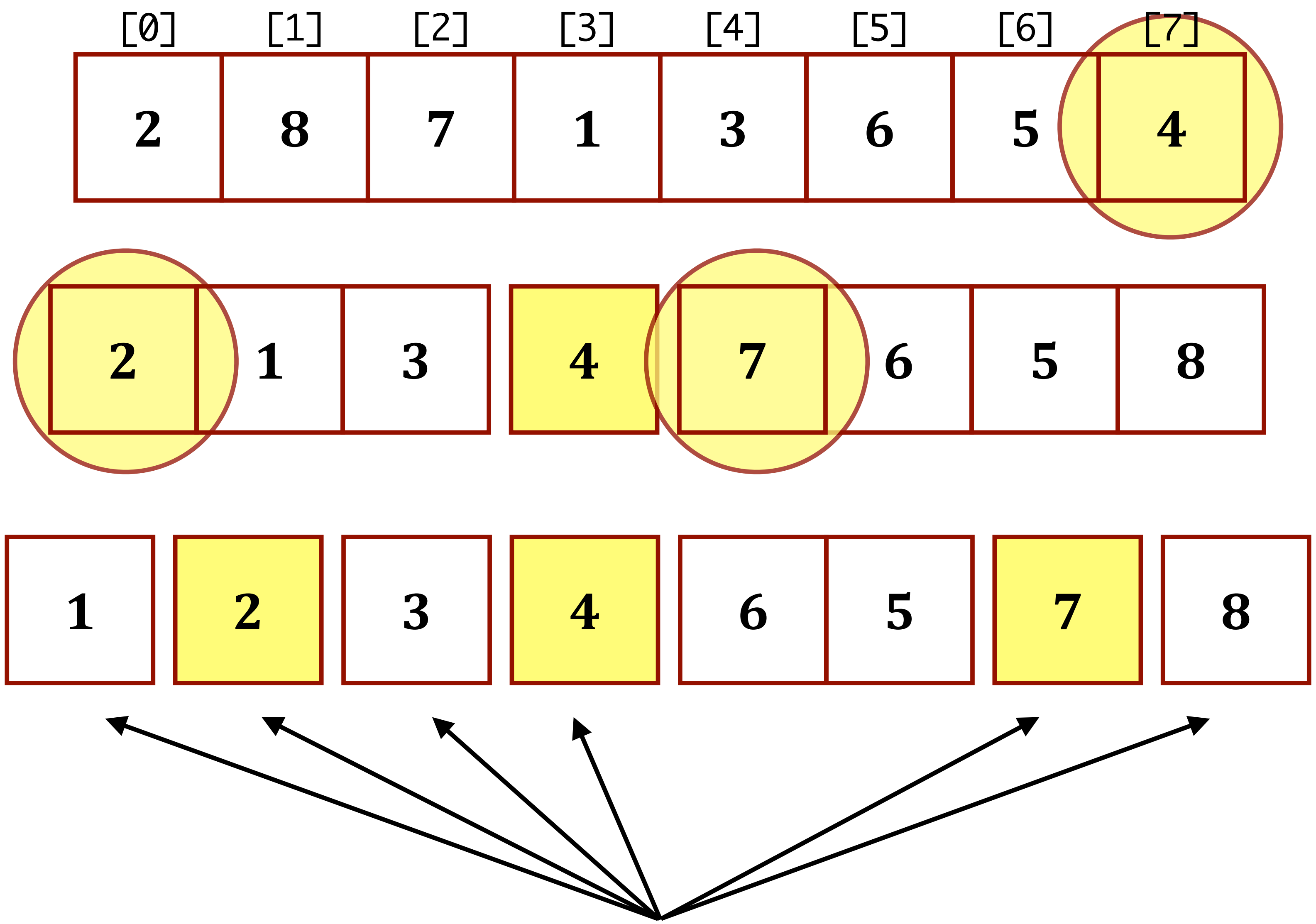
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



Divide and Conquer :: Quick Sort

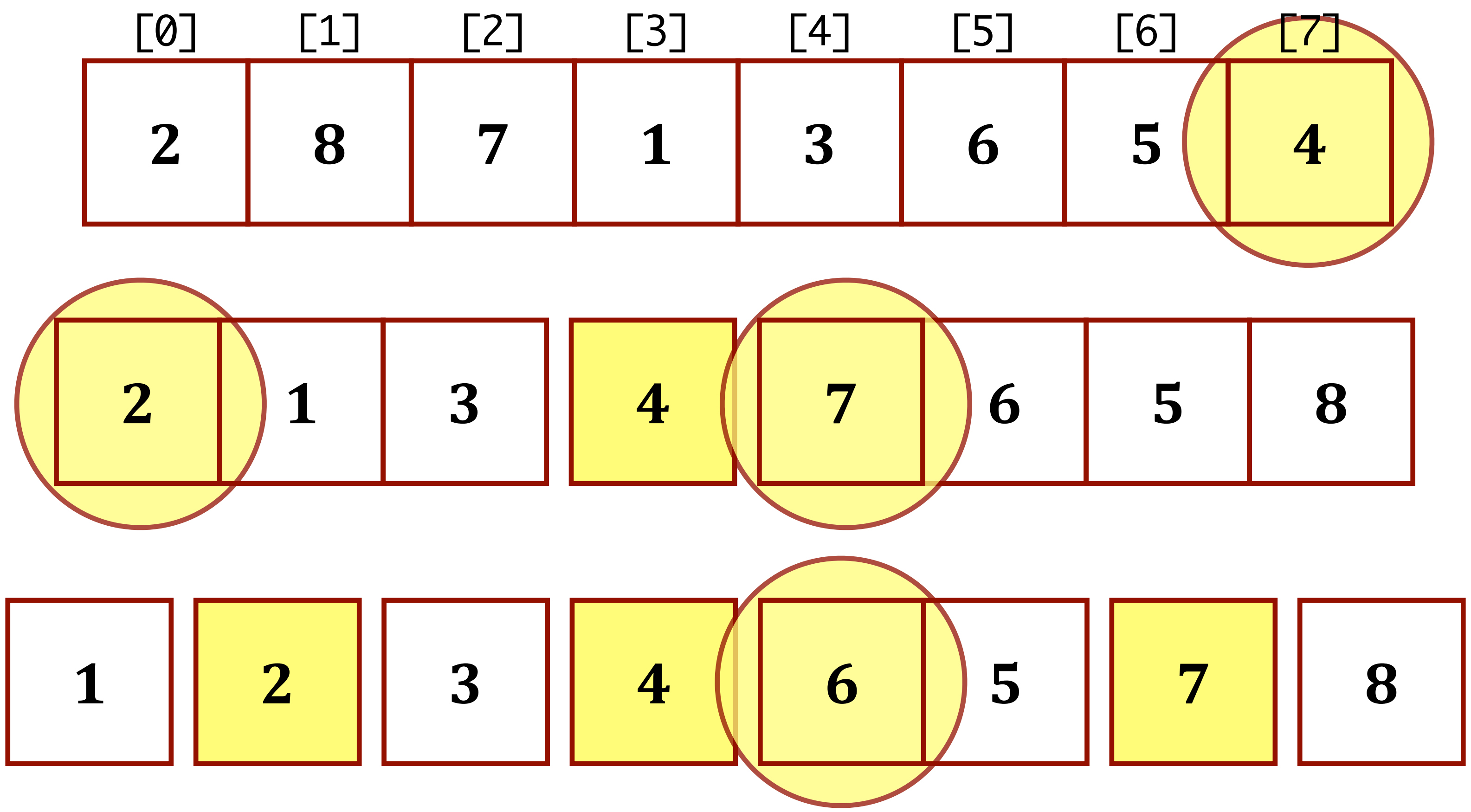
Let's look at Quicksort again, this time focused on what the array looks like at each step.



Sub-arrays of size 1. Mostly done.

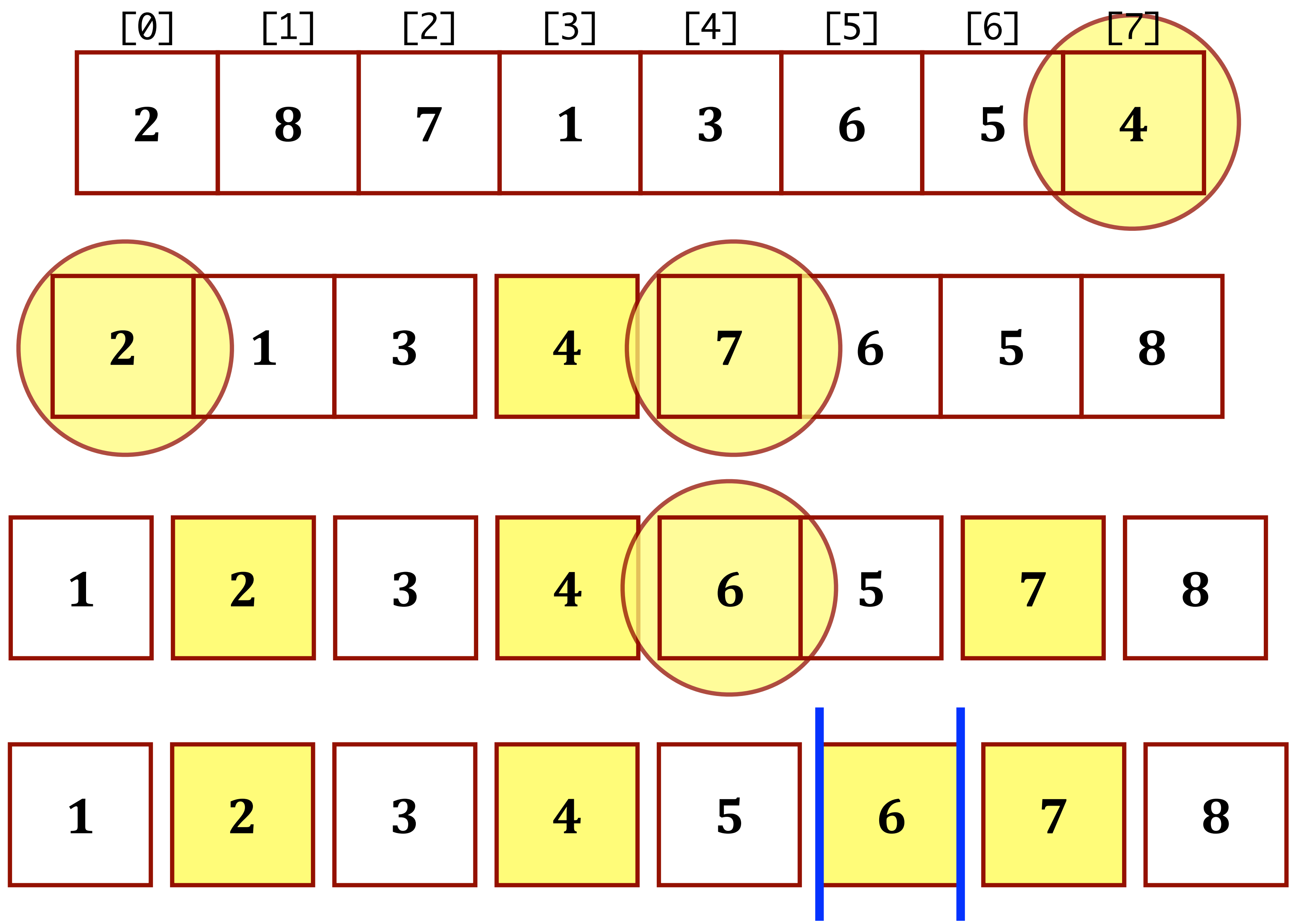
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



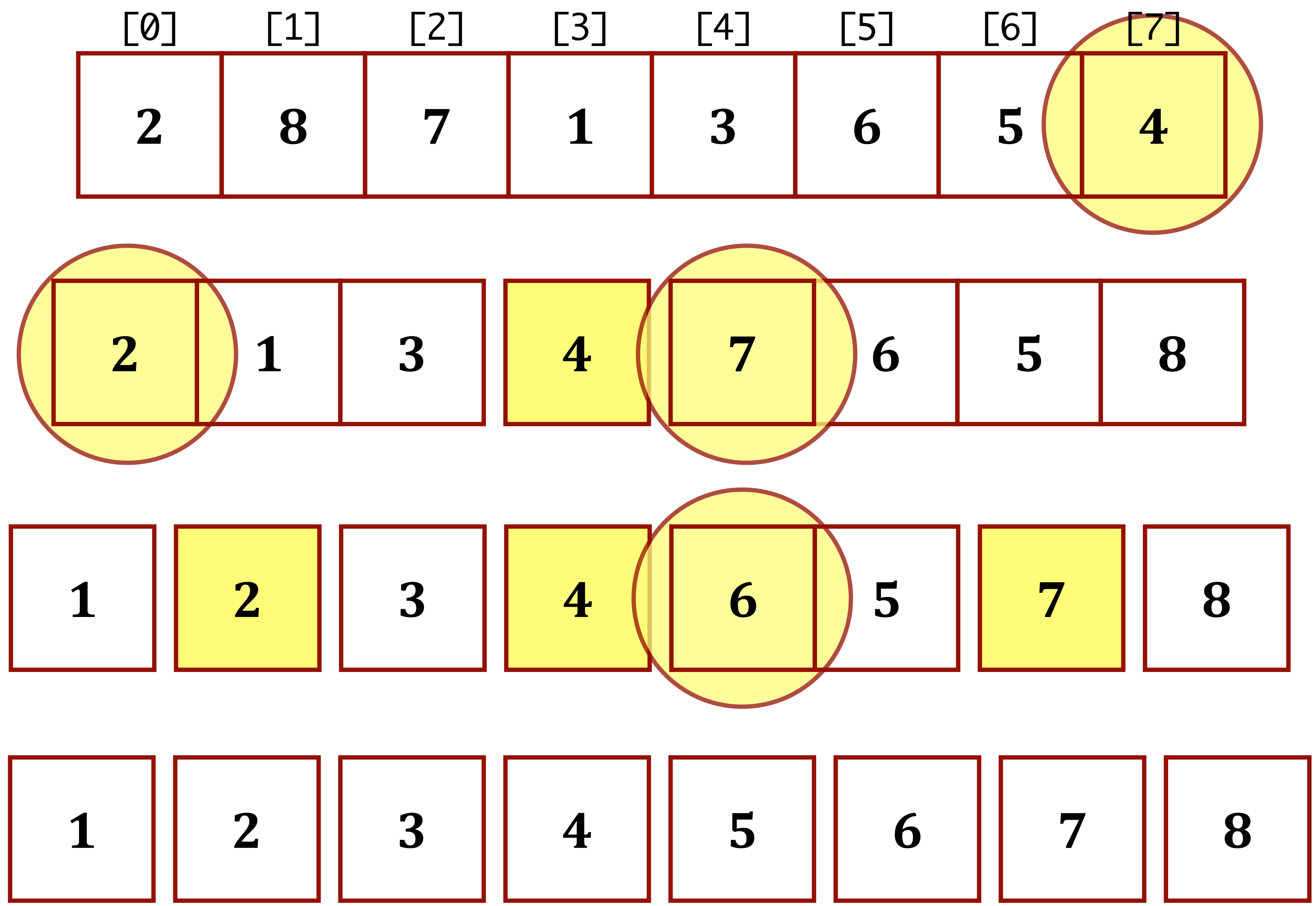
Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



Divide and Conquer :: Quick Sort

Let's look at Quicksort again, this time focused on what the array looks like at each step.



Sorted!

Divide and Conquer :: Quick Sort

Let's look at Quicksort one more time, this time with dancers.

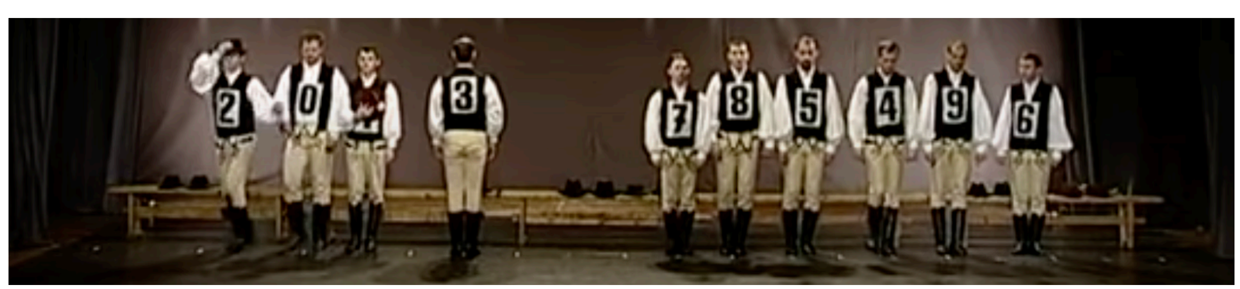
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	**0**	**1**	**8**	**7**	**2**	**5**	**4**	**9**	**6**

Divide and Conquer :: Quick Sort

Let's look at Quicksort one more time, this time with dancers.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	**0**	**1**	**8**	**7**	**2**	**5**	**4**	**9**	**6**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
2	**0**	**1**	**3**	**7**	**8**	**5**	**4**	**9**	**6**

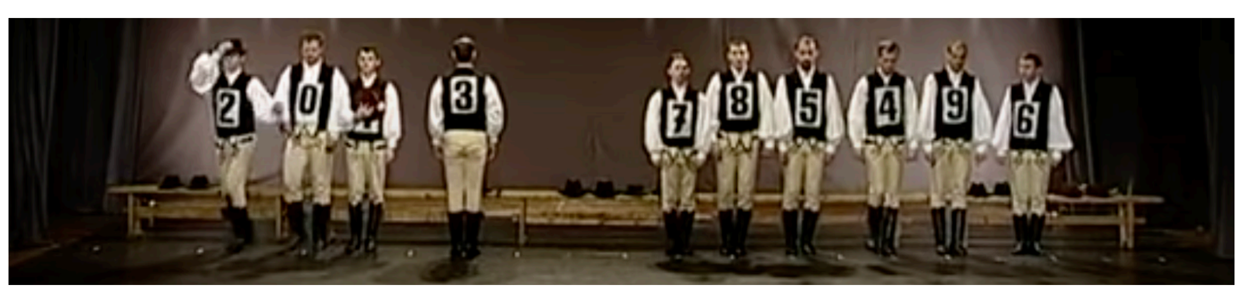


Divide and Conquer :: Quick Sort

Let's look at Quicksort one more time, this time with dancers.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	**0**	**1**	**8**	**7**	**2**	**5**	**4**	**9**	**6**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
2	**0**	**1**	**3**	**7**	**8**	**5**	**4**	**9**	**6**



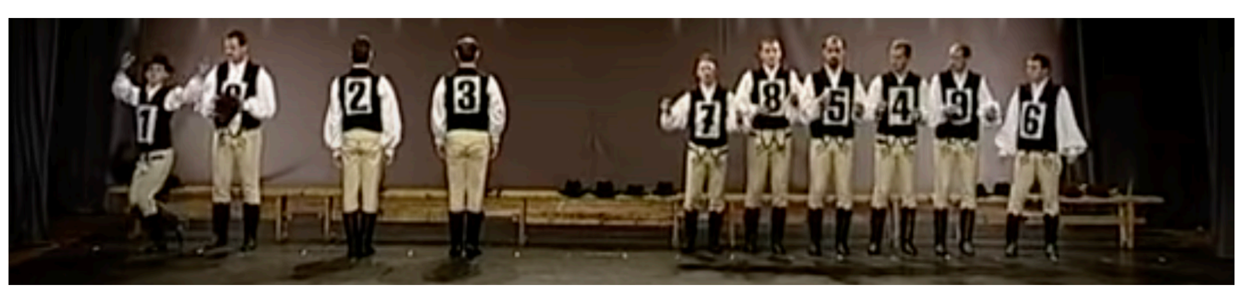
Divide and Conquer :: Quick Sort

Let's look at Quicksort one more time, this time with dancers.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	**0**	**1**	**8**	**7**	**2**	**5**	**4**	**9**	**6**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
2	**0**	**1**	**3**	**7**	**8**	**5**	**4**	**9**	**6**

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	**0**	**2**	**3**	**7**	**8**	**5**	**4**	**9**	**6**



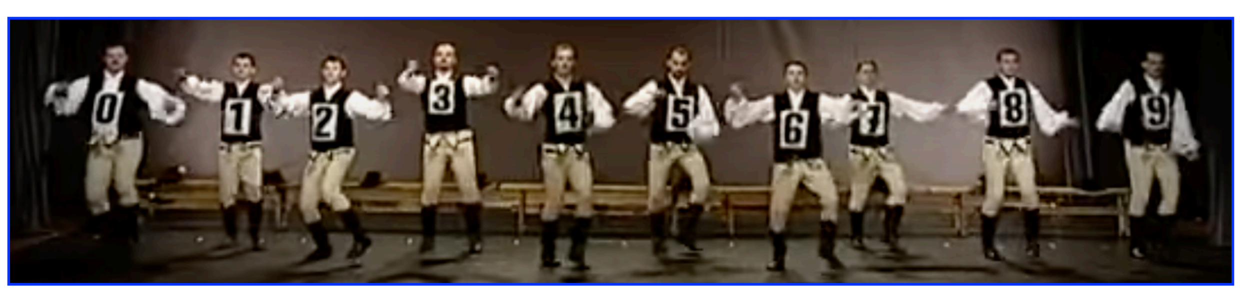
Divide and Conquer :: Quick Sort

Let's look at Quicksort one more time, this time with dancers.

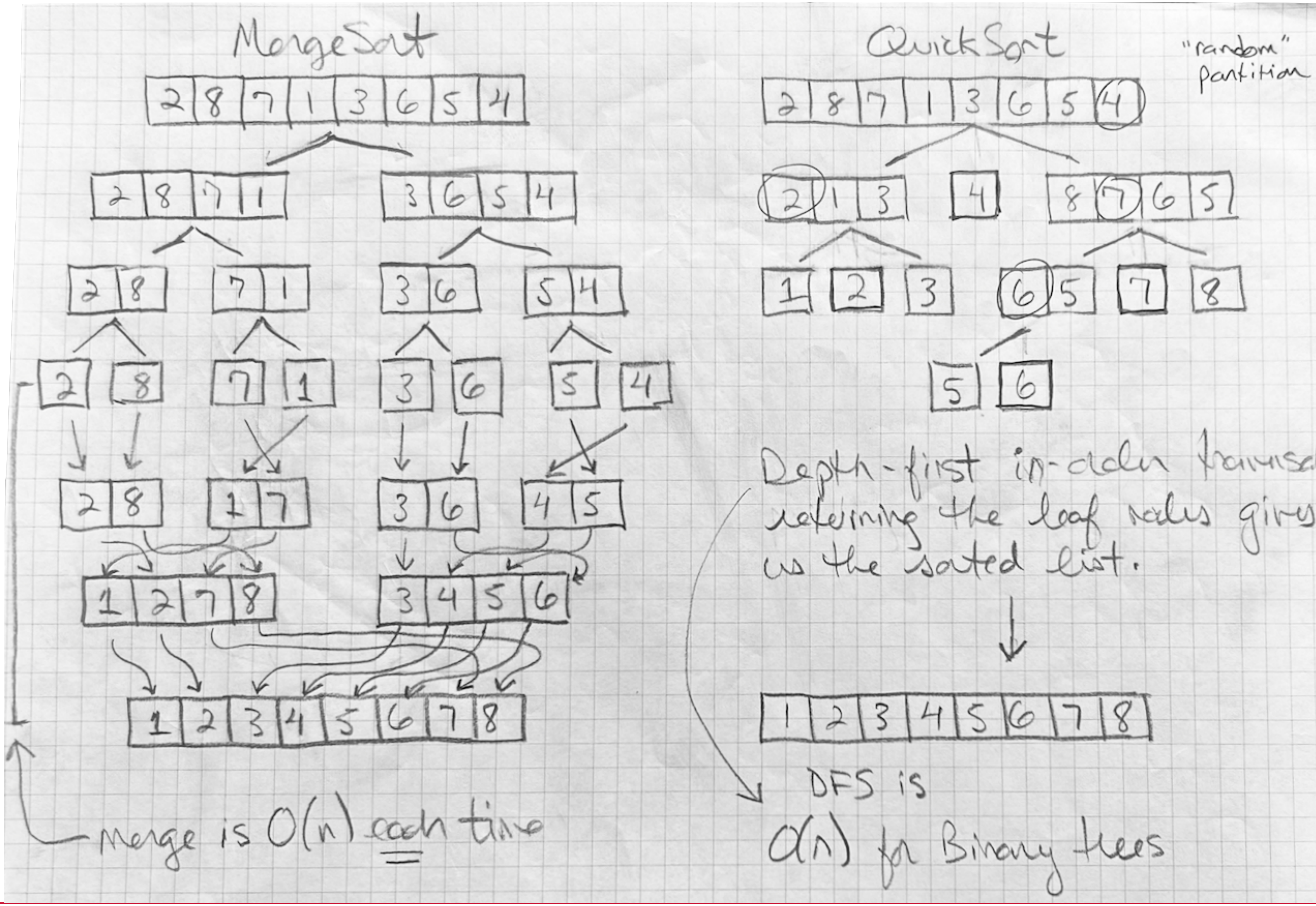
Right side sorted

⋮

All sorted



Divide and Conquer :: Merge Sort and Quick Sort



Divide and Conquer :: Merge Sort and Quick Sort

So… what is the complexity of Merge Sort and QuickSort?

Divide and Conquer :: Merge Sort and Quick Sort

So… what is the complexity of Merge Sort and QuickSort?

Both Merge Sort and QuickSort tend to be $O(n \times \log_2 n)$.

Why?

Divide and Conquer :: Merge Sort and Quick Sort

So… what is the complexity of Merge Sort and QuickSort?

Both Merge Sort and QuickSort tend to be $O(n \times \log_2 n)$.

Why?