Quicksort Implementation

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Learn how quicksort works
- Learn how to partition an array

Exercise

- Partitioning

Extra Resources

- https://me.dt.in.th/page/Quicksort/
- https://www.youtube.com/watch?v=ywWBy6J5gz8
- CLRS Chapter 7

Quicksort

- A practical and simple algorithm
- The running time = O(n lg n)
- Superior to other O(n lg n) in some respects
- The <u>hidden</u> constants are small (hidden by Big-O)
- Our first stochastic algorithm

Quicksort

Input : an array of n elements in any order

Output : a reordering of the input array such that the elements are in non-decreasing order

Key idea of Quicksort: **partition** the array around a pivot element

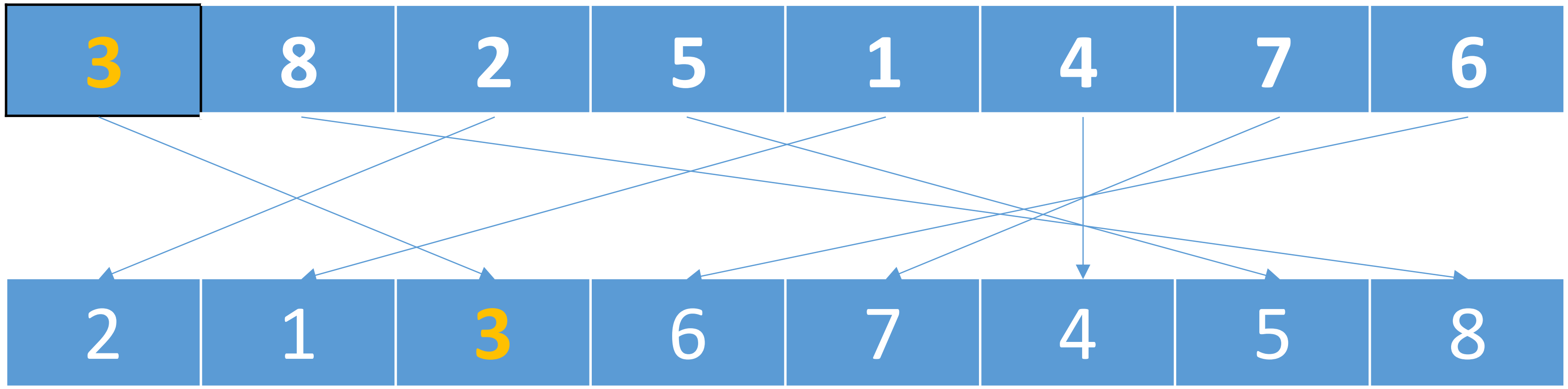
Key concept of Quicksort

- Pick an element and call it the pivot
- Partition (rearrange) the elements so that:
  - Everything to the left of the pivot is less than the pivot
  - Everything to the right of the pivot is greater than the pivot
  - Let's ignore ties for now

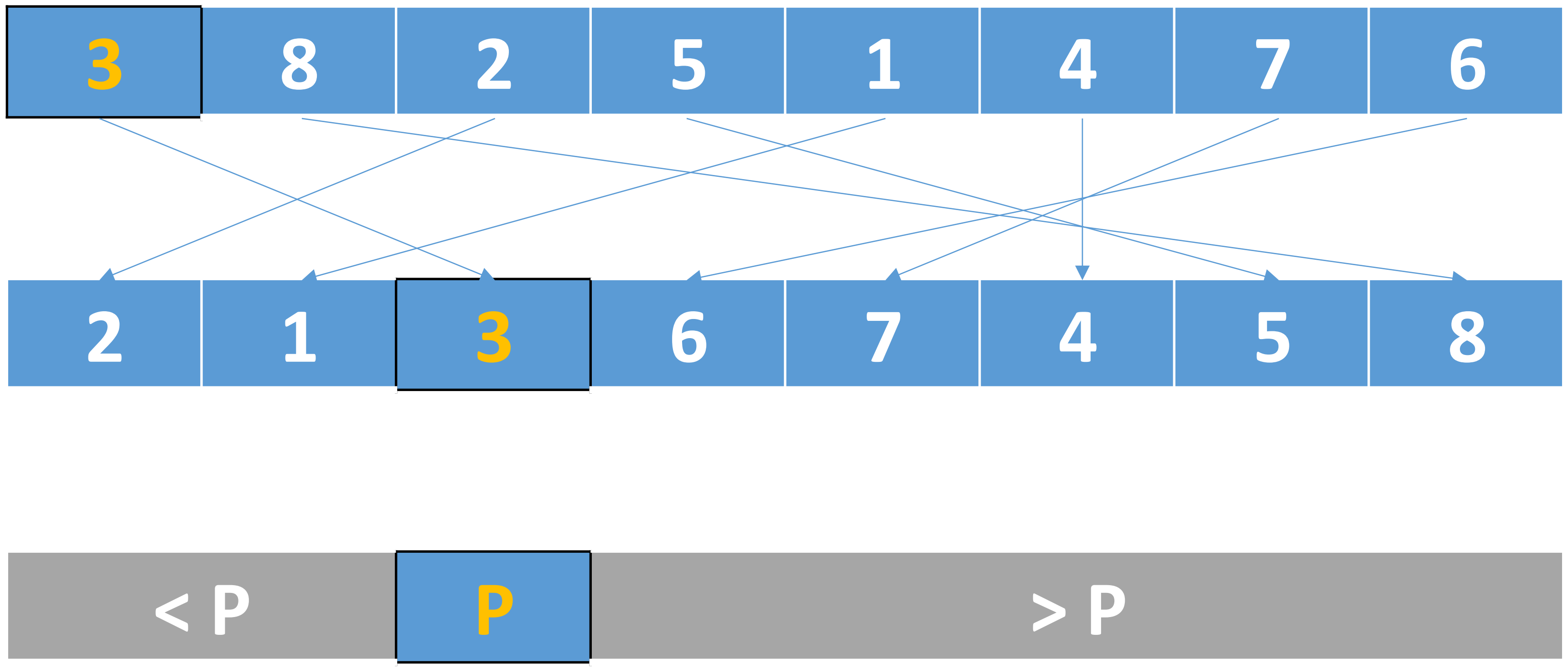
- This is a partial sorting into "buckets"
- What can you tell me about the pivot?
- **Pivot is now in the correct spot (we've made progress!)**

What would be the running time of calling partition on every element?

Partitioning



Partitioning



Pivot around “hello”

[“hello”, “are”, “you”, “how”, “today”, “doing”, “class”]

Quicksort (NOT IN-PLACE PARTITIONING)

```
FUNCTION BadQuicksort(array)
  IF array.length ≤ 1
    RETURN array

  pivot_index = ChoosePivot(array.length)
  left_array, right_array = Partition(array, pivot_index)

  left_sorted = BadQuicksort(left_array)
  right_sorted = BadQuicksort(right_sorted)

  RETURN left_sorted ++ array[pivot_index] ++ right_sorted
```

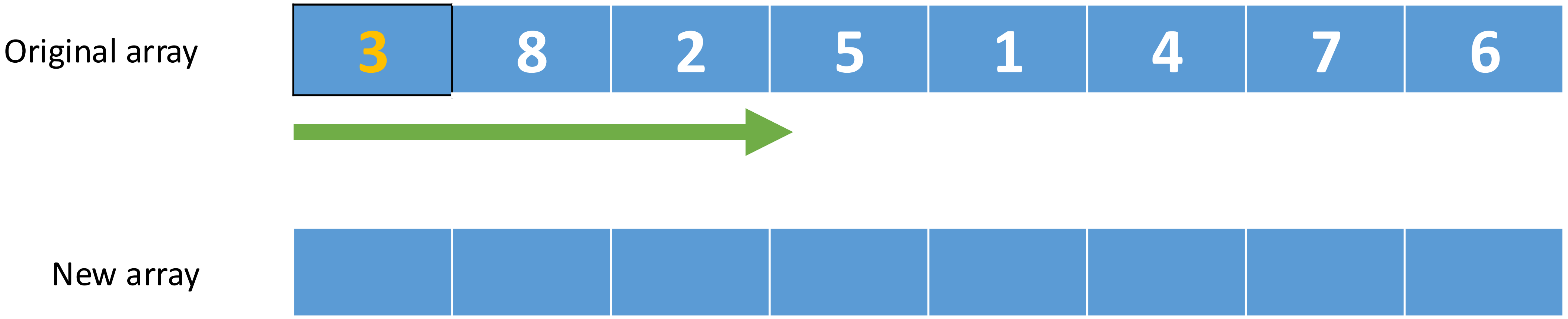
What is the recurrence equation for Quicksort?

Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array

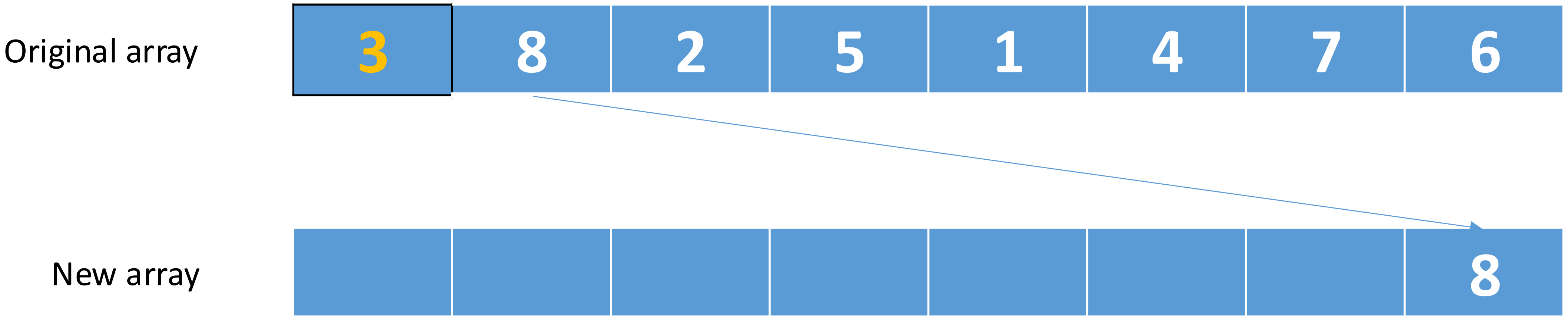
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



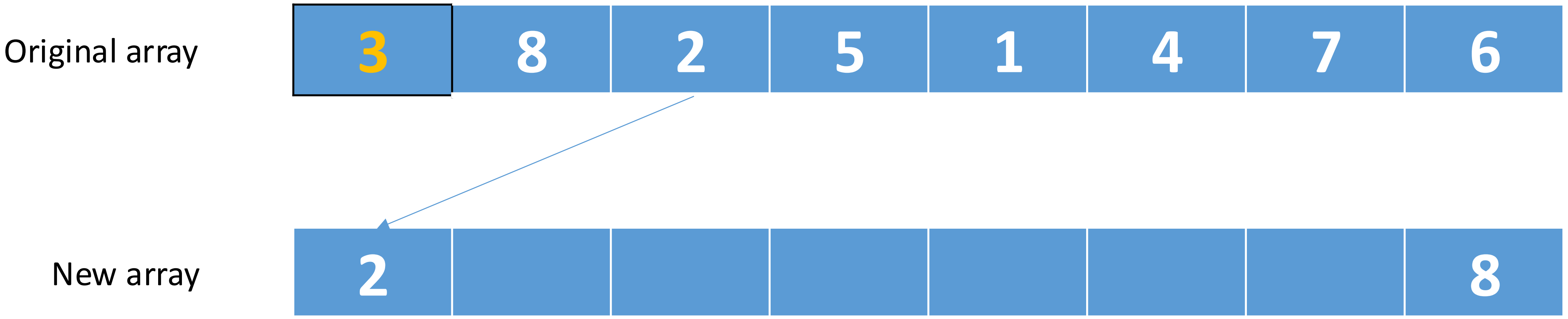
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



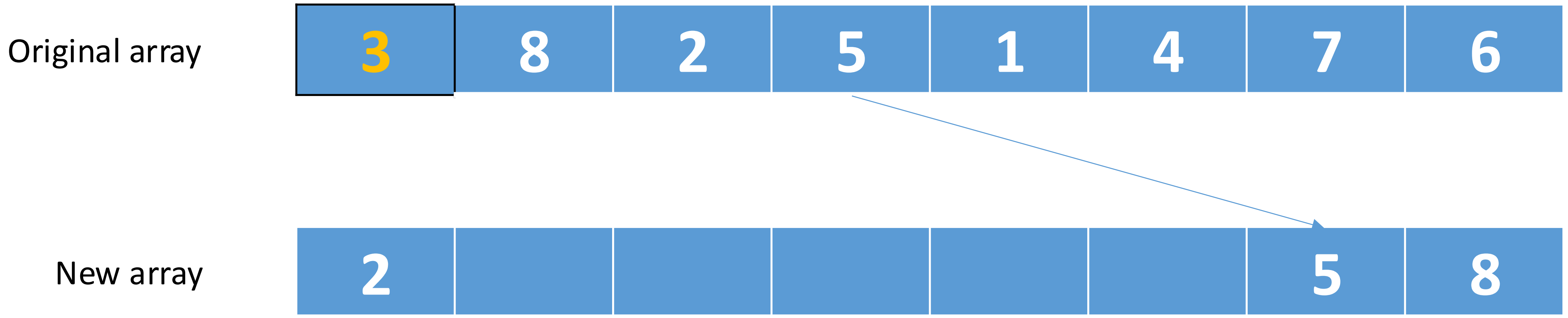
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



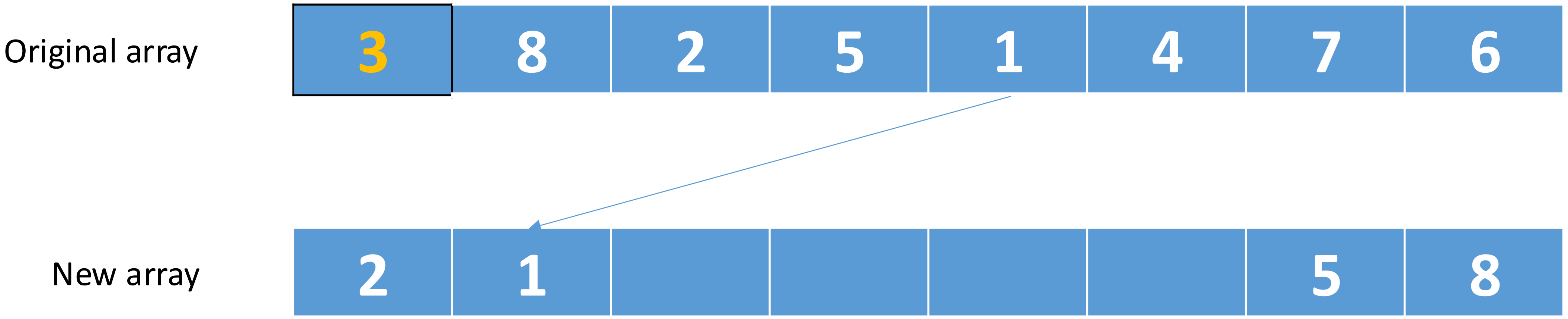
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



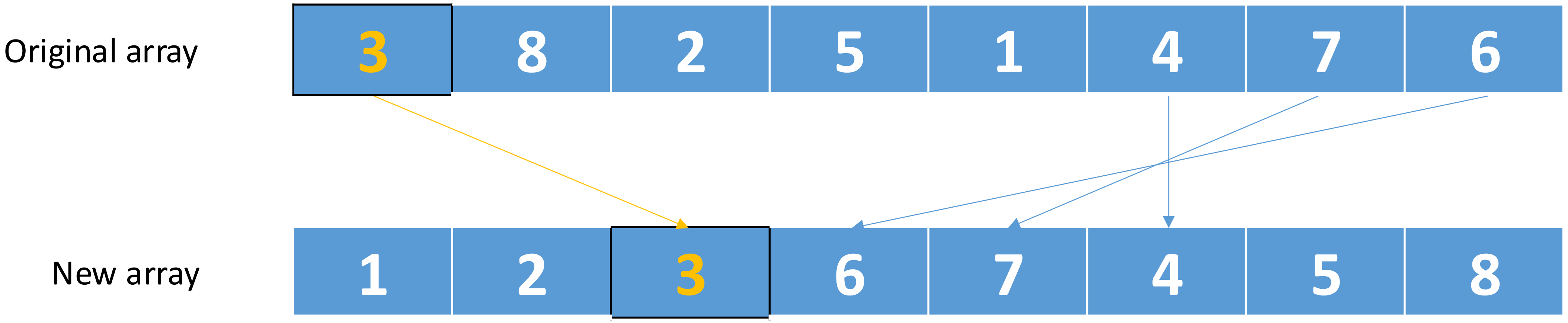
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



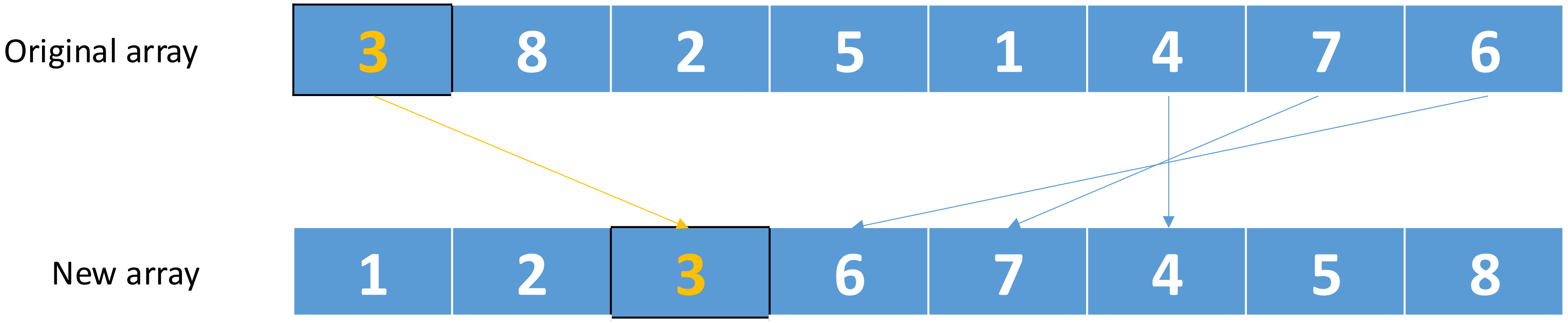
Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



Partitioning the Easy Way

- How would you partition? (how did we perform a merge?)
- Copy all elements to a new array



- This would be like merge sort.
- Lots of memory allocations (one for each node in the recursion tree).

Partitioning the Easy Way

- Nothing inherently wrong with this approach **in theory**
- But can we do the same thing without the extra memory?

- Note: implementing **merge sort** “in-place” is possible
- You can do so with an iterative (stack based) approach

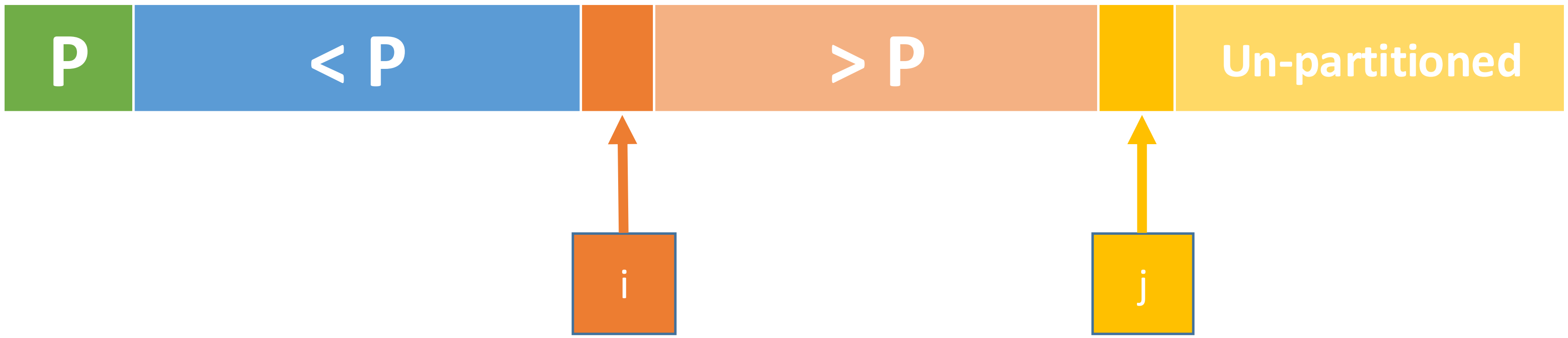
Partitioning In-Place

- For now, assume that the pivot is in the first spot of a subarray
- (we can swap the pivot with the first spot if needed)

- Idea: gradually build up a subarray that is correctly partitioned by scanning through the array

P	**< P**	**> P**	**Un-partitioned**

Partitioning In-Place



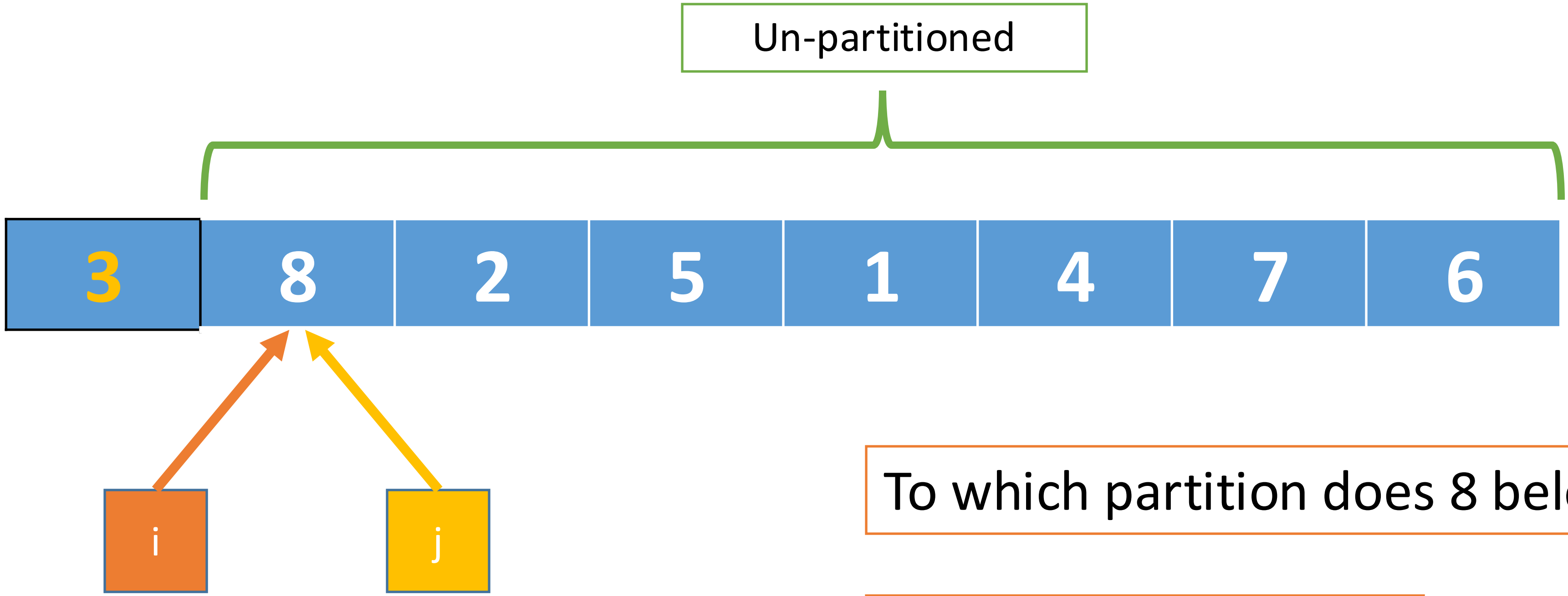
Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition

3	8	2	5	1	4	7	6

Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition



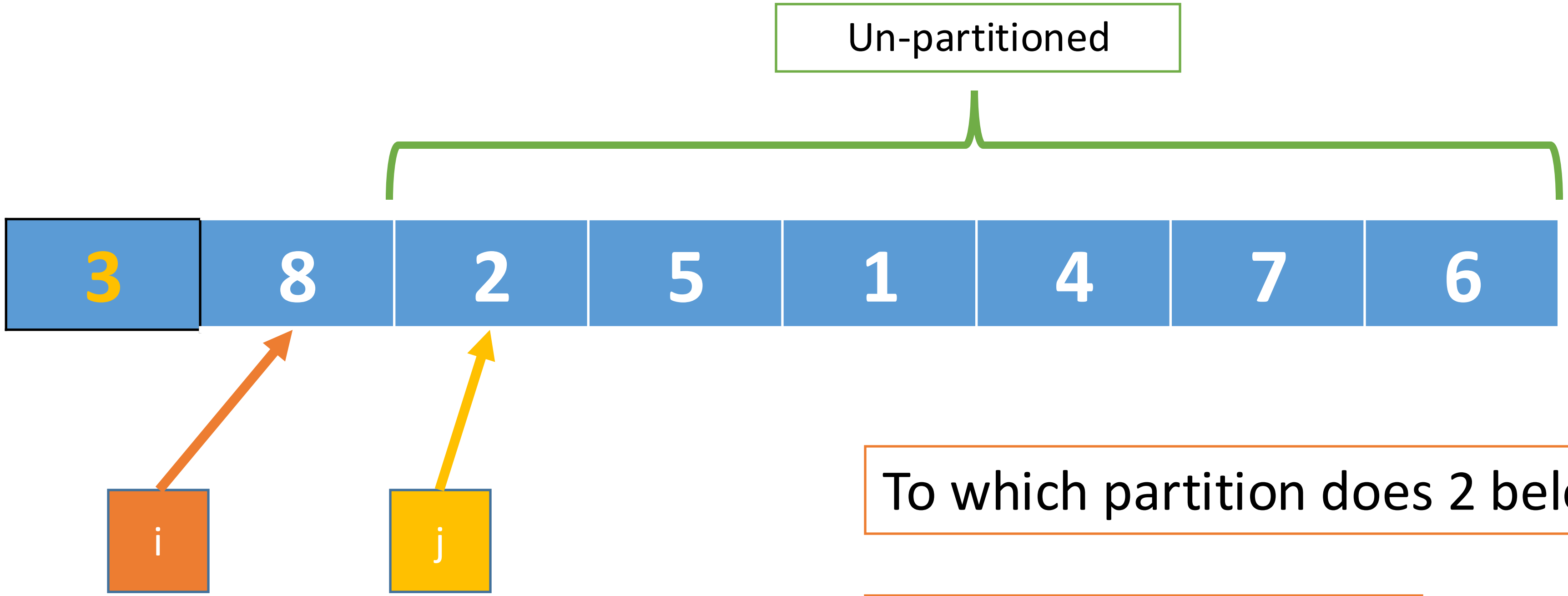
To which partition does 8 belong?

How do I put it there?

How should we initialize i and j?

Index one to the right of the “smaller-than“ partition

Index one to the right of the “larger-than” partition

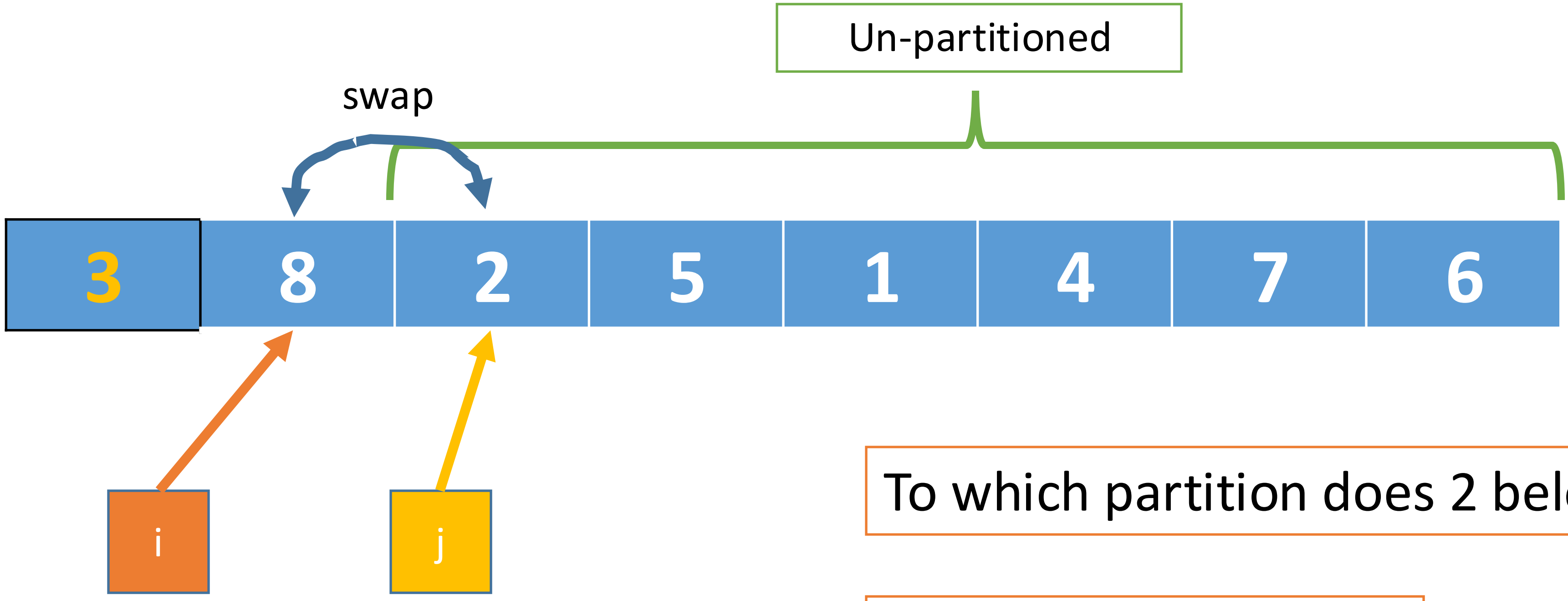


To which partition does 2 belong?

How do I put it there?

Index one to the right of the “smaller-than“ partition

Index one to the right of the “larger-than” partition

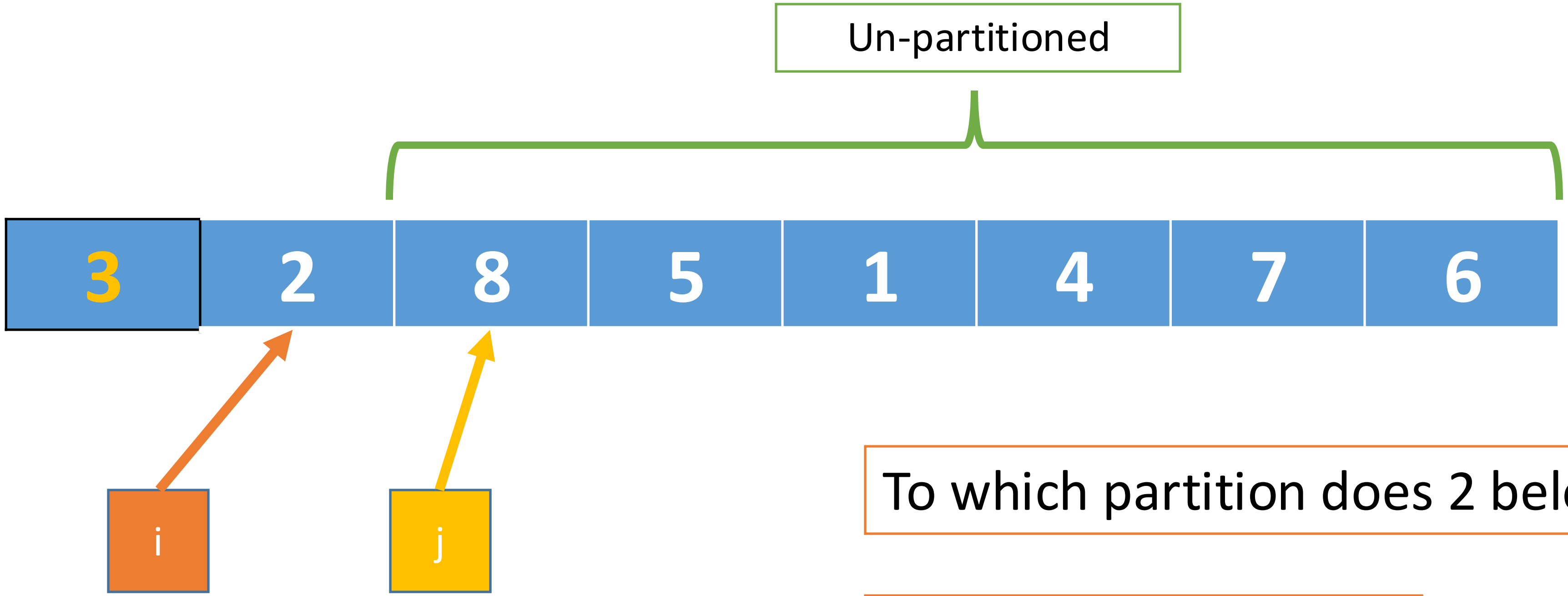


To which partition does 2 belong?

How do I put it there?

Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition



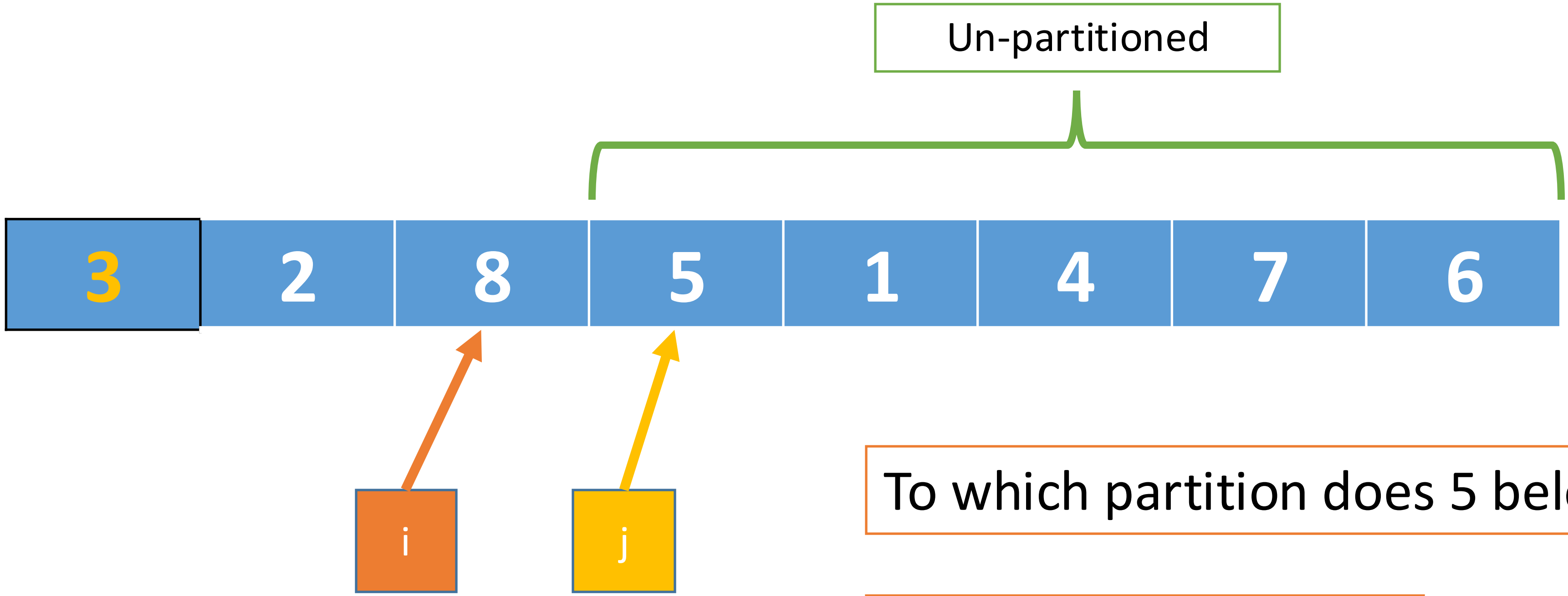
To which partition does 2 belong?

How do I put it there?

Now what?

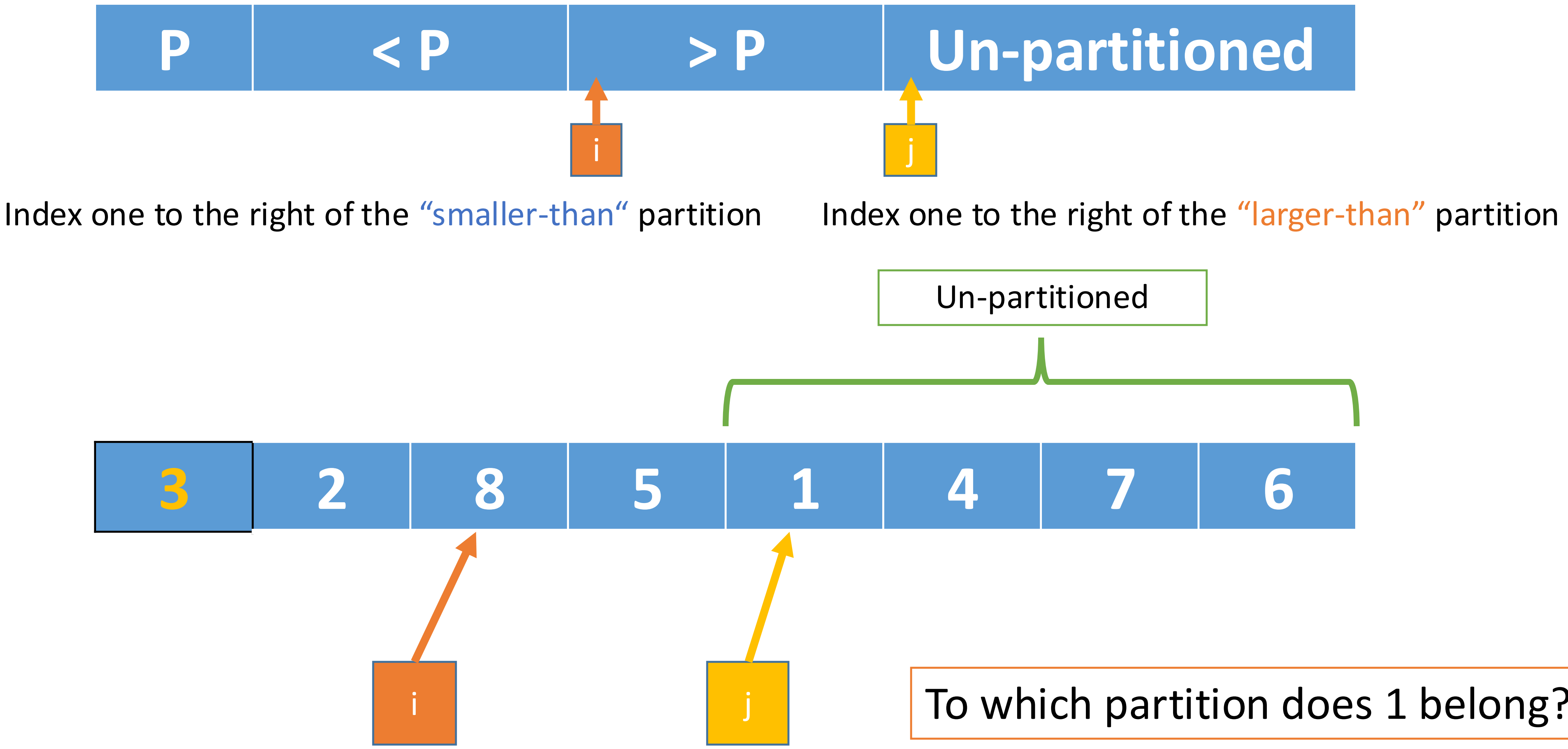
Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition



To which partition does 5 belong?

How do I put it there?

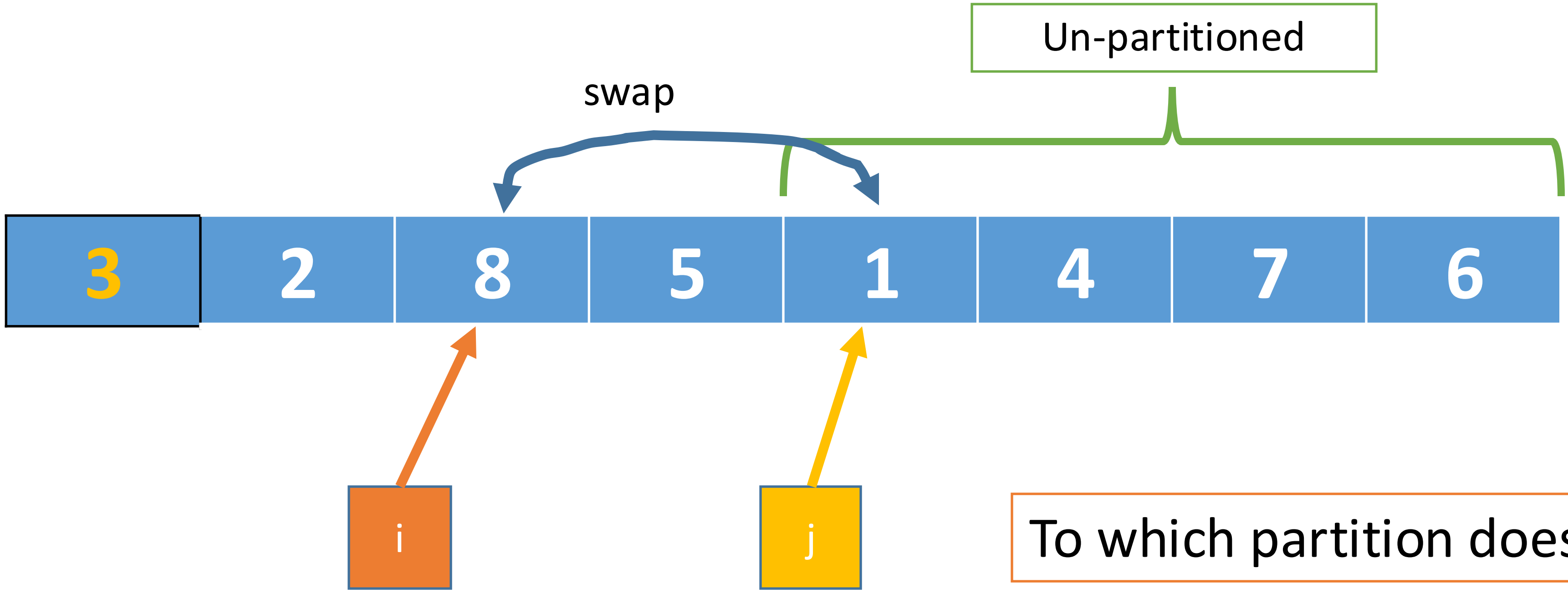


To which partition does 1 belong?

How do I put it there?

Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition

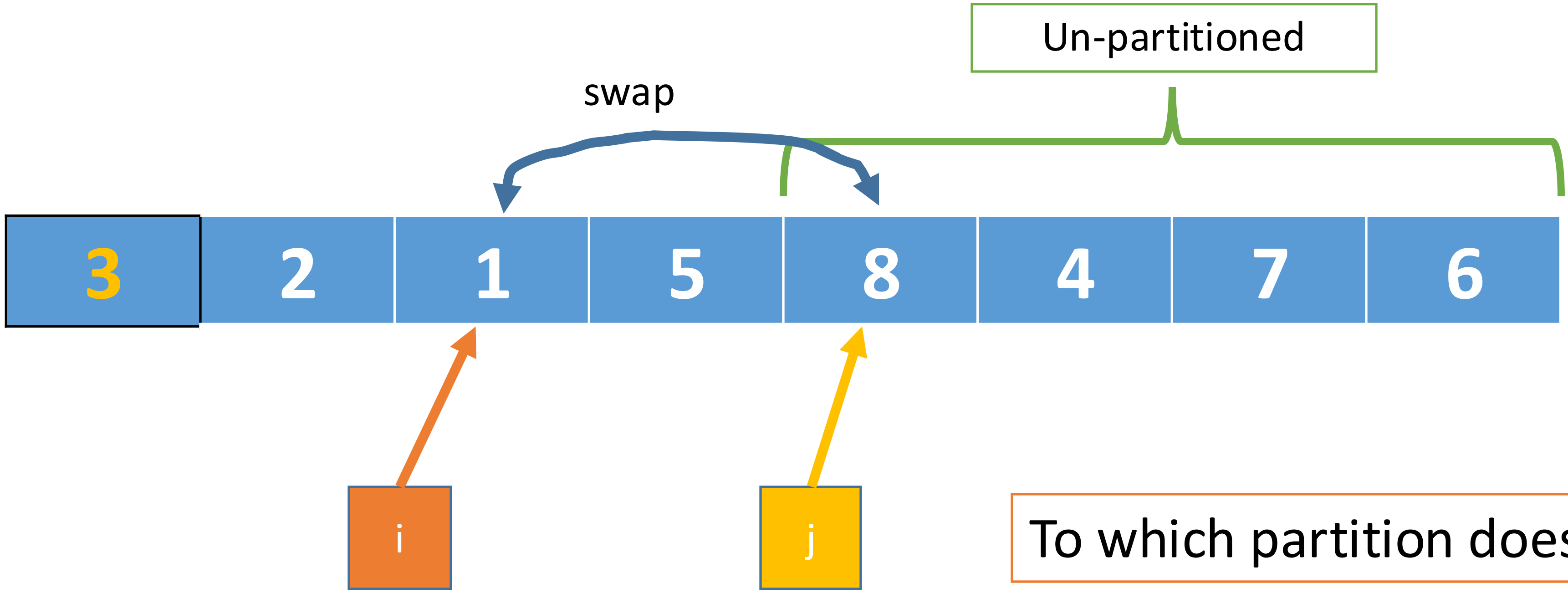


To which partition does 1 belong?

How do I put it there?

Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition



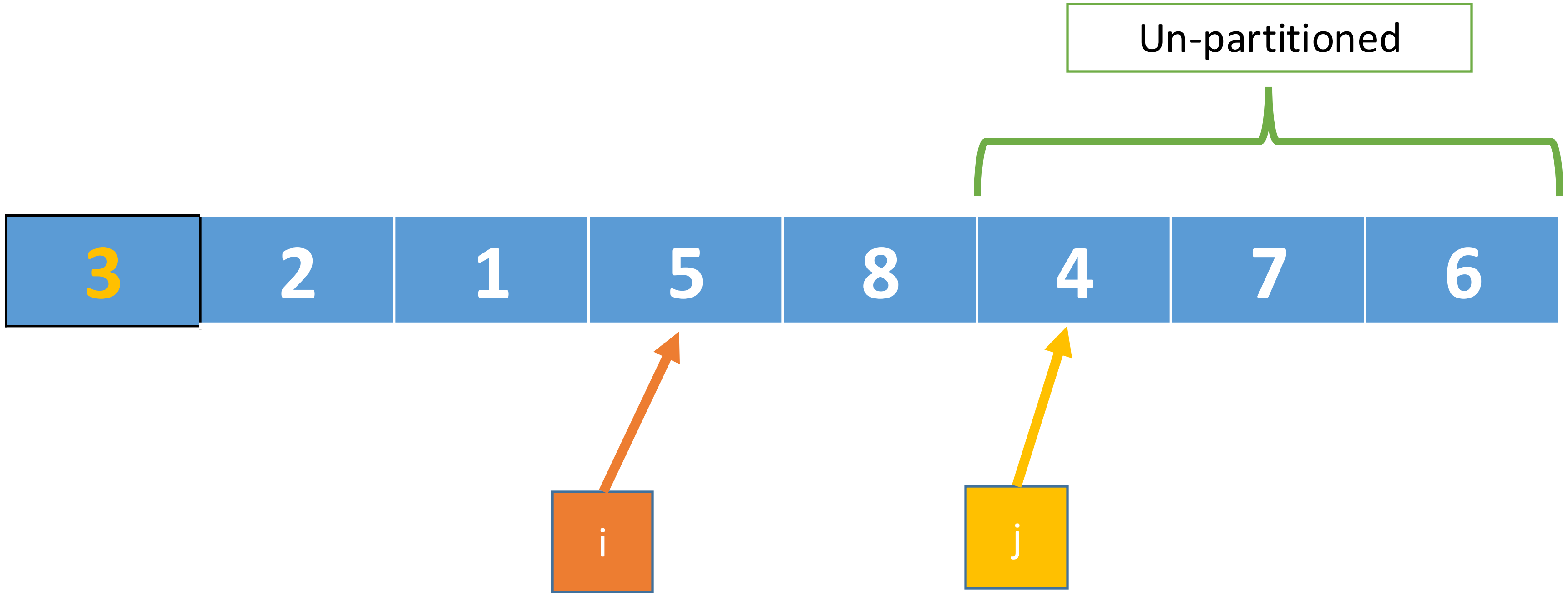
To which partition does 1 belong?

How do I put it there?

Now what?

Index one to the right of the “smaller-than“ partition

Index one to the right of the “larger-than” partition



P	< P	> P	Un-partitioned

i

j

Index one to the right of the “smaller-than“ partition

Index one to the right of the “larger-than” partition

3	2	1	5	8	4	7	6

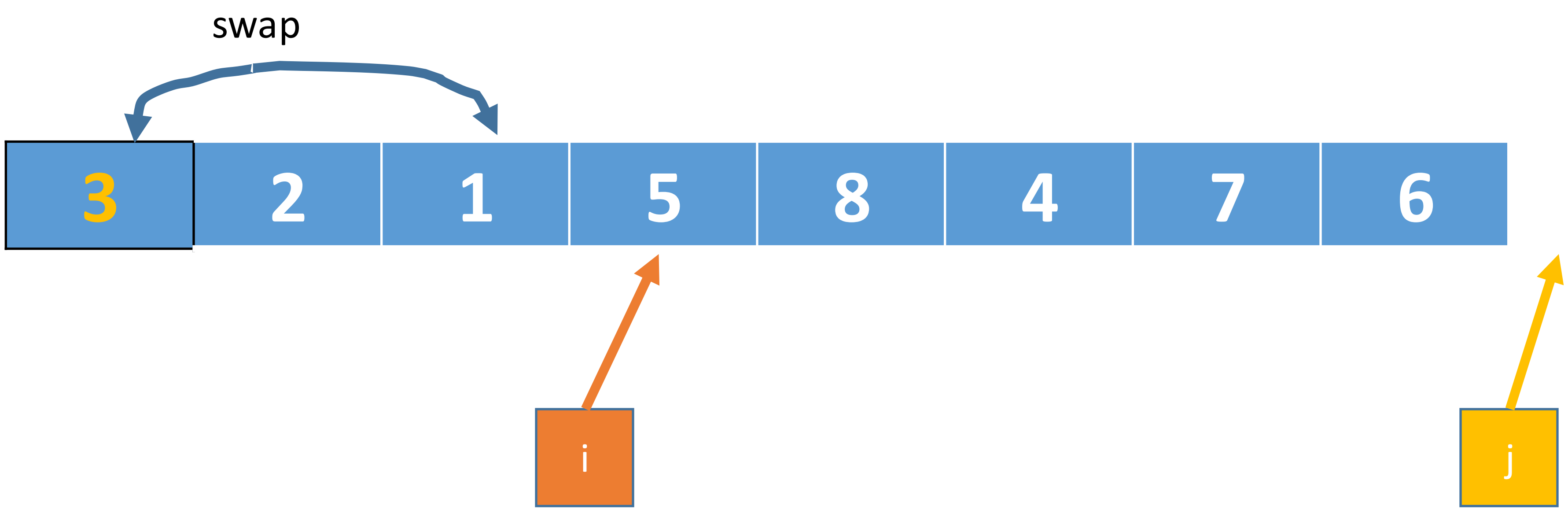
i

j

Now what?

Index one to the right of the "smaller-than" partition

Index one to the right of the "larger-than" partition



1	2	3	5	8	4	7	6

What is the asymptotic running time?

```
FUNCTION Partition(array, left_index, right_index)
  # Partition the subarray array[left_index ..< right_index]
  # around the value at left_index

  pivot_value = array[left_index]

  i = left_index + 1
  FOR j IN [left_index + 1 ..< right_index]
    IF array[j] < pivot_value
      swap(array, i, j)
      i = i + 1

  swap(array, left_index, i - 1)
  RETURN i - 1
```

1. O(n), where n is right_index - left_index
2. In-place
   no extra memory

```
FUNCTION QuickSort(array, left_index, right_index)
    IF (left
        RETURN

    MovePivotToLeft(left_index, right_index)
    pivot_index = Partition(array, left_index, right_index)

    QuickSort(array, left_index,
    QuickSort(array, pivot_index
```

Our Partition function expects the pivot element to be at left_index

How would you call QuickSort?

```
FUNCTION QuickSort(array, left_index, right_index)
    IF left_index ≥ right_index
        RETURN

    MovePivotToLeft(left_index, right_index)
    pivot_index = Partition(array, left_index, right_index)

    QuickSort(array, left_index, pivot_index)
    QuickSort(array, pivot_index + 1, right_index)
```

Our Partition function expects the pivot element to be at left_index