Quicksort Correctness Proof

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

Outline

Topics and Learning Objectives

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

Exercise

- Quicksort loop invariant

Extra Resources

- https://me.dt.in.th/page/Quicksort/
- https://www.youtube.com/watch?v=ywWBy6J5gz8
- CLRS Chapter 7
- Algorithms Illuminated Chapter 5

What do we need to do?

Input: an array of n items in arbitrary order

Output: the same number in non-decreasing order

Assumptions: the items must be orderable (from an ordinal set)

Theorem: the Quicksort algorithm arranges all items in non-decreasing order.

1. **Lemma** involving `Partition`
2. **Lemma** involving `QuickSort`

31	47	11	91	67	23	89	51

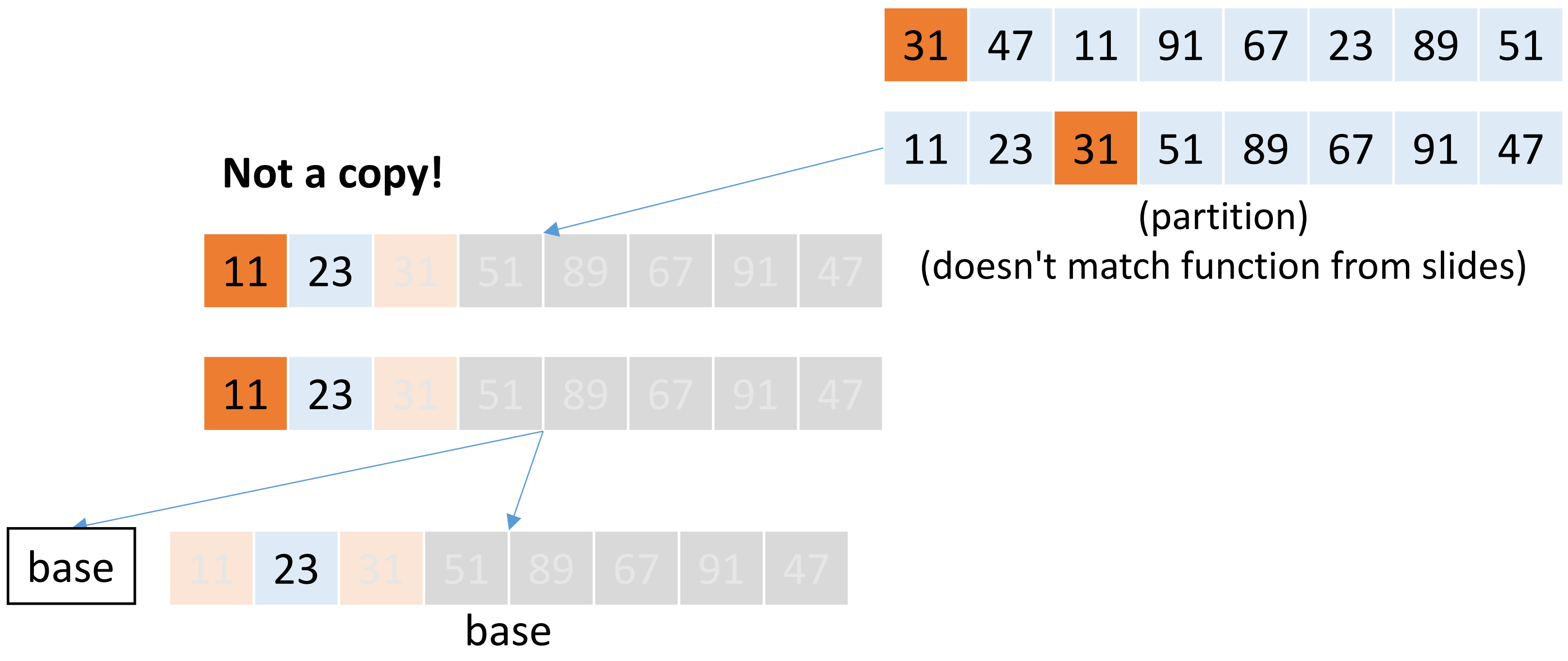
31 47 11 91 67 23 89 51

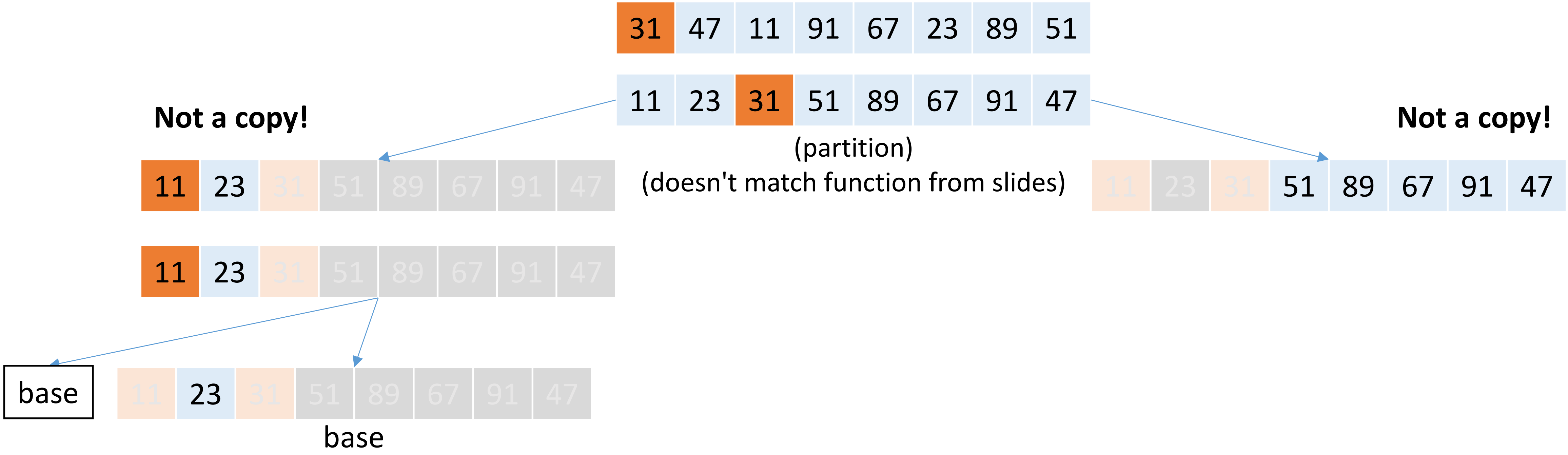
11 23 31 51 89 67 91 47

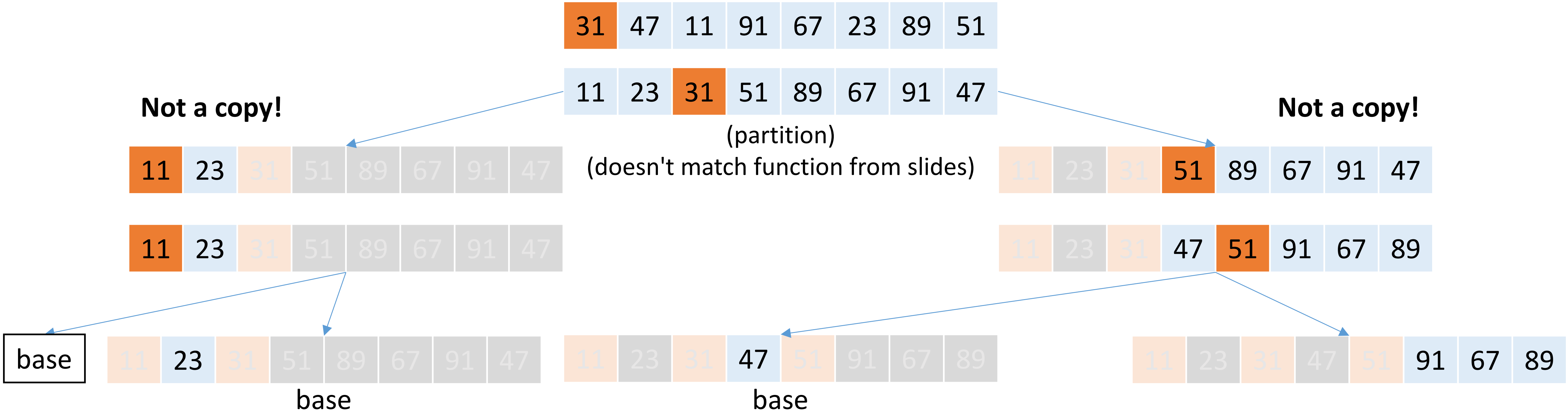
(partition)
(doesn't match function from slides)

Not a copy! (In-place)

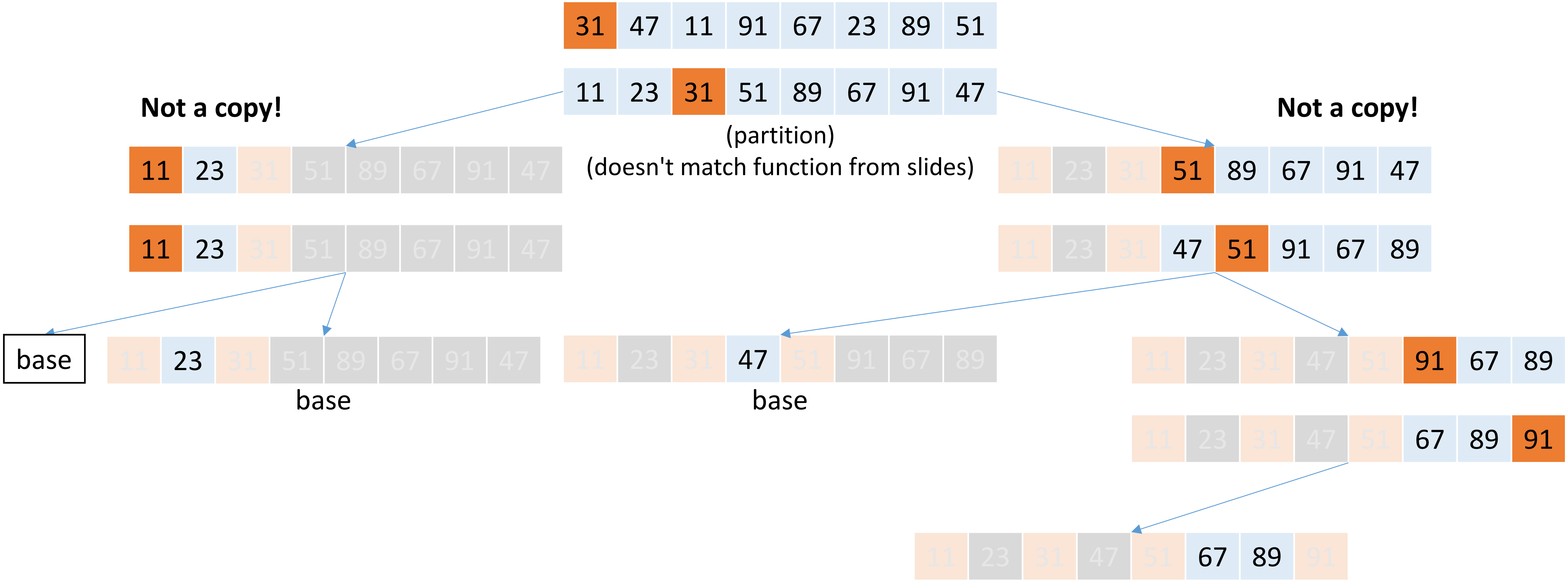
11 23 31 51 89 67 91 47

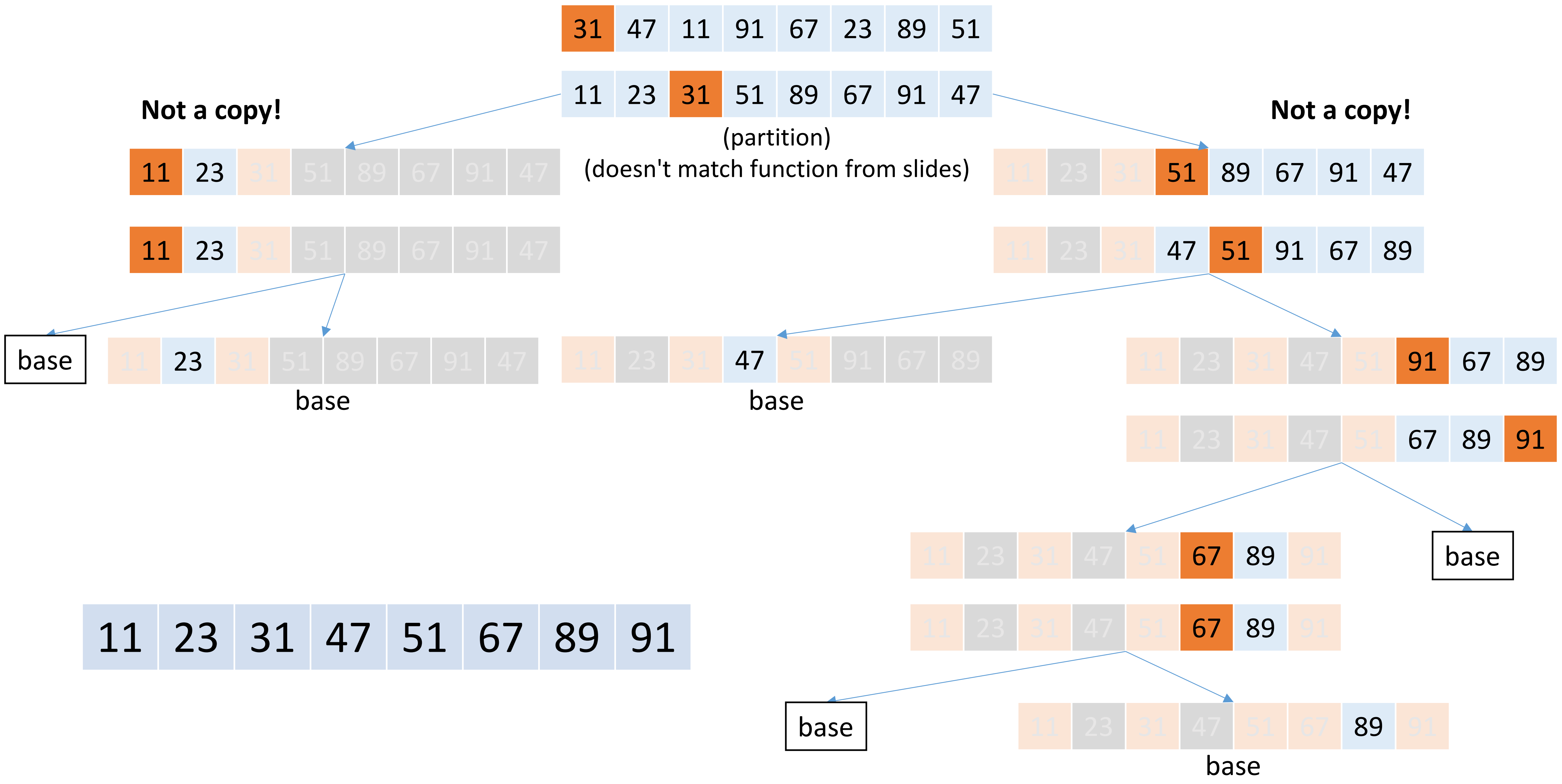

31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
(partition)
(doesn't match function from slides)
Not a copy!
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
base
11 23 31 51 89 67 91 47
base


31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
(partition)
(doesn't match function from slides)
Not a copy!
11 23 31 51 89 67 91 47
Not a copy!
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
base
11 23 31 51 89 67 91 47
base

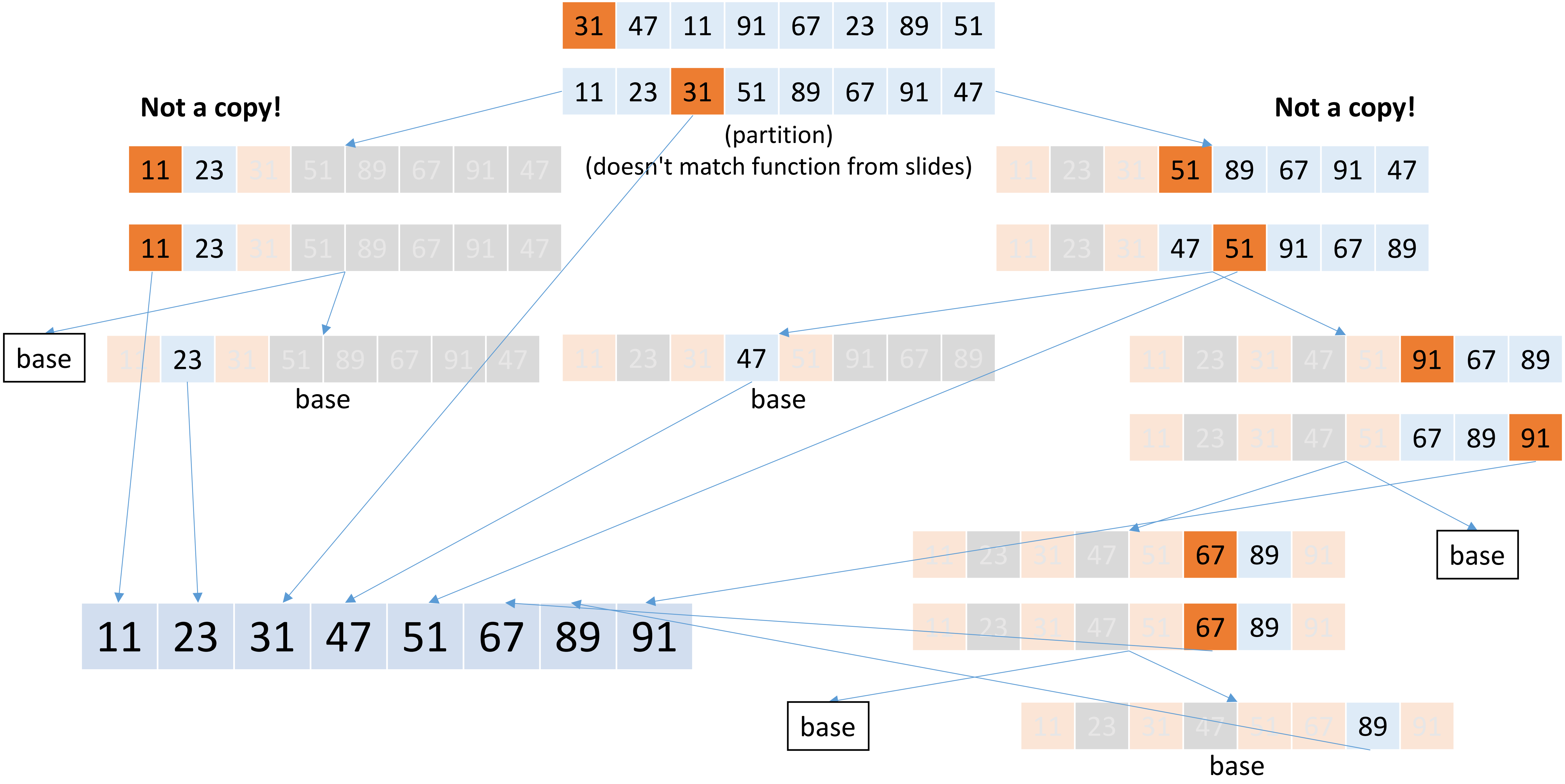


31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
Not a copy!
Not a copy!
(partition)
(doesn't match function from slides)
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
11 23 31 47 51 91 67 89
base
11 23 31 51 89 67 91 47
base
11 23 31 47 51 91 67 89
base
11 23 31 47 51 91 67 89


31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
(partition)
(doesn't match function from slides)
Not a copy!
11 23 31 51 89 67 91 47
Not a copy!
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
11 23 31 47 51 91 67 89
base
11 23 31 51 89 67 91 47
base
11 23 31 47 51 91 67 89
base
11 23 31 47 51 91 67 89
11 23 31 47 51 67 89 91
11 23 31 47 51 67 89 91



31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
(partition)
(doesn't match function from slides)
Not a copy!
11 23 31 51 89 67 91 47
Not a copy!
11 23 31 51 89 67 91 47
11 23 31 51 89 67 91 47
11 23 31 47 51 91 67 89
base
11 23 31 51 89 67 91 47
base
11 23 31 47 51 91 67 89
base
11 23 31 47 51 91 67 89
11 23 31 47 51 67 89 91
11 23 31 47 51 67 89 91
base
11 23 31 47 51 67 89 91
11 23 31 47 51 67 89 91
base
11 23 31 47 51 67 89 91
base



31 47 11 91 67 23 89 51
11 23 31 51 89 67 91 47
Not a copy!
Not a copy!
(partition)
(doesn't match function from slides)
11 23
51 89 67 91 47
11 23
47 51 91 67 89
base
23
base
47
base
91 67 89
67 89 91
base
67 89
67 89
base
89
base
11 23 31 47 51 67 89 91

Partition proof of correctness

Value	67	44	…	21	-87	…	5	101	-31	…	4
Index	0	1	…	left	left + 1	…	right - 1	right	right + 1	…	n - 1

```
FUNCTION Partition(array, left_index, right_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
```

Partition proof of correctness

Value	67	44	…	21	-87	…	5	101	-31	…	4
Index	0	1	…	left	left + 1	…	right - 1	right	right + 1	…	n - 1

```
FUNCTION Partition(array, left_index, right_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
```

How do we prove that `Partition` is correct?

Loop Invariant Proofs

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement should reference the purpose of the loop
   3. The statement should reference variables that change each iteration

Initialization

2. Show that the loop invariant is true before the loop starts

Maintenance

3. Show that the loop invariant holds when executing any iteration

4. Show that the loop invariant holds once the loop ends

Termination

Partition proof of correctn

Value	67	44	…	21	-87	…
Index	0	1	…	left	left + 1	…

```
FUNCTION Partition(array, left_index, right_in
    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
```

Exercise

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement should reference the purpose of the loop
   3. The statement should reference variables that change each iteration

Initialization

2. Show that the loop invariant is true before the loop starts

Maintenance

3. Show that the loop invariant holds when executing any iteration
4. Show that the loop invariant holds once the loop ends

Termination

Partition proof of correctness

Value	67	44	…	21	-87	…	5	101	-31	…	4
Index	0	1	…	left	left + 1	…	right - 1	right	right + 1	…	n - 1

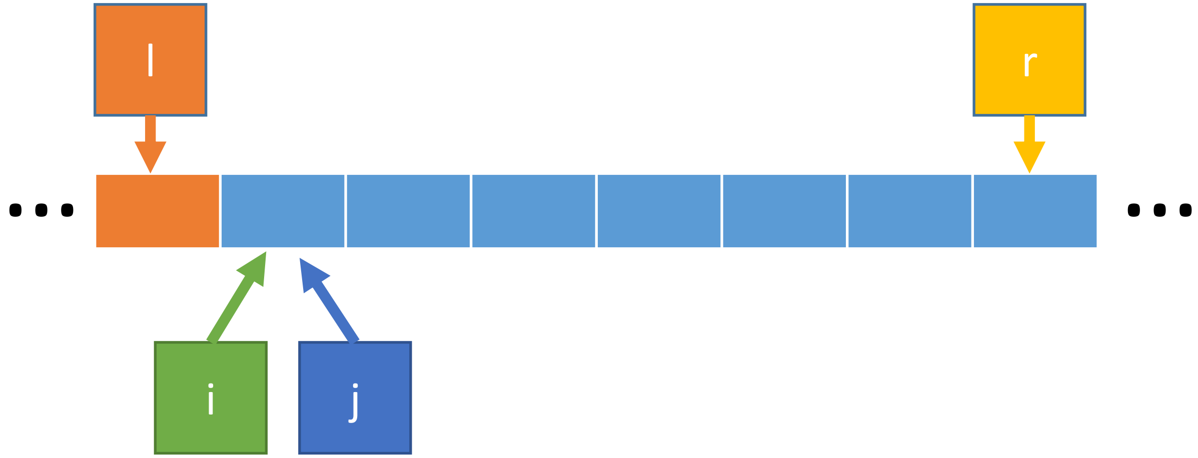
```
FUNCTION Partition(array, left_index, right_index)
    pivot_value = array[left_index]
    i = left_index + 1
    FOR j IN [left_index + 1 ..< right_index]
        IF array[j] < pivot_value
            swap(array, i, j)
            i = i + 1
    swap(array, left_index, i – 1)
    RETURN i – 1
```

How do we prove that `Partition` is correct?

Loop Invariant: At the start of the iteration with indices i and j:

1. All items in `array[l+1 ..= i-1]` are < `pivot_value`
2. All items in `array[i ..= j-1]` are ≥ `pivot_value`

Partition Proof

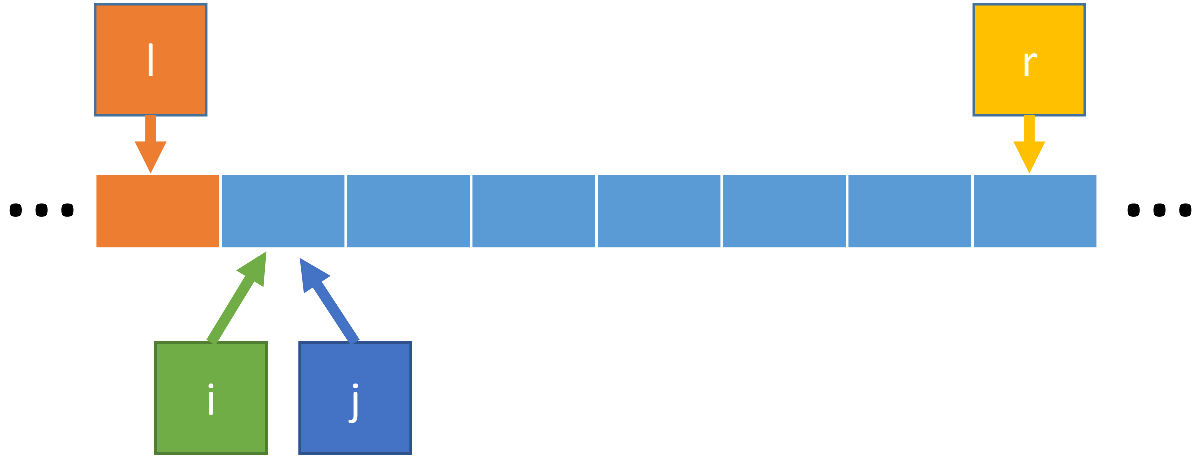


Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

```
FUNCTION Partition(a, l, r)
   pivot_value = a[l]
   i = l + 1
   FOR j IN [l + 1 ..< r]
      IF a[j] < pivot_value
         swap(a, i, j)
         i = i + 1
   swap(a, l, i - 1)
   RETURN i - 1
```

Partition Proof



Initialization: Show that the loop invariant is true before the loop starts

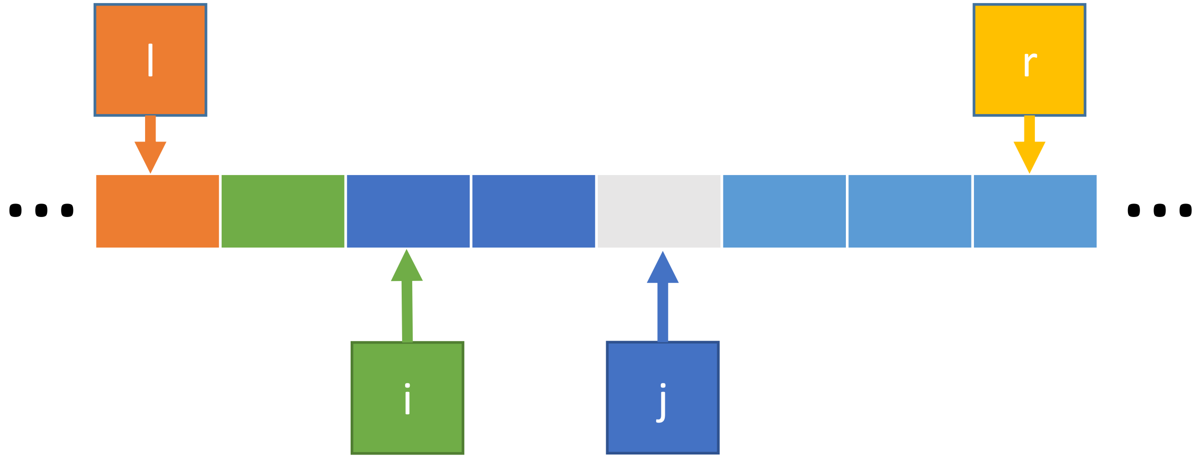
1. No numbers in a[l+1 ..= i-1]
2. No numbers in a[i ..= j-1]

Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i - 1)
    RETURN i - 1
```

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

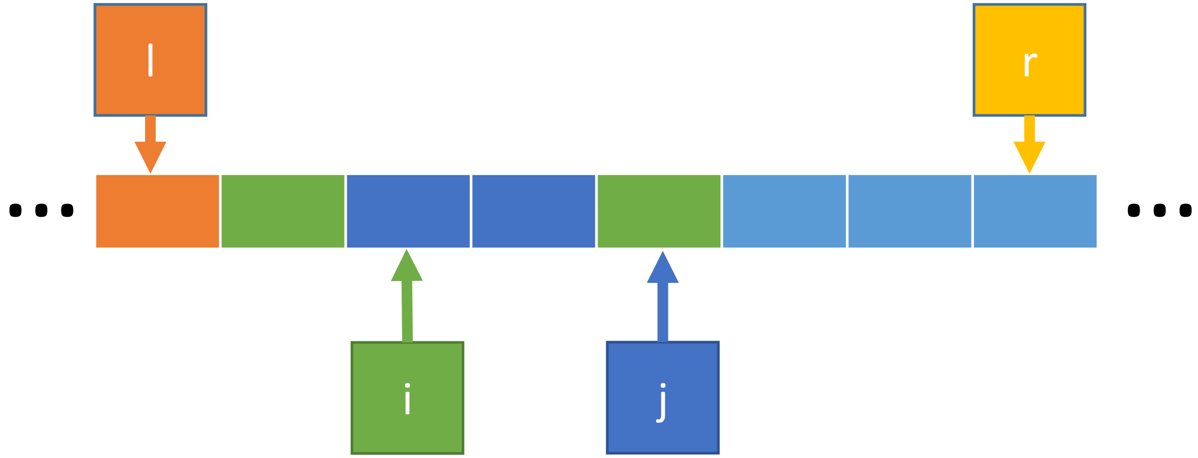
```
FUNCTION Partition(a, l, r)
  pivot_value = a[l]
  i = l + 1
  FOR j IN [l + 1 ..< r]
    IF a[j] < pivot_value
      swap(a, i, j)
      i = i + 1
  swap(a, l, i − 1)
  RETURN i − 1
```

true

Maintenance (**case 1**): Show that the loop invariant holds when executing any iteration

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] < pivot

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

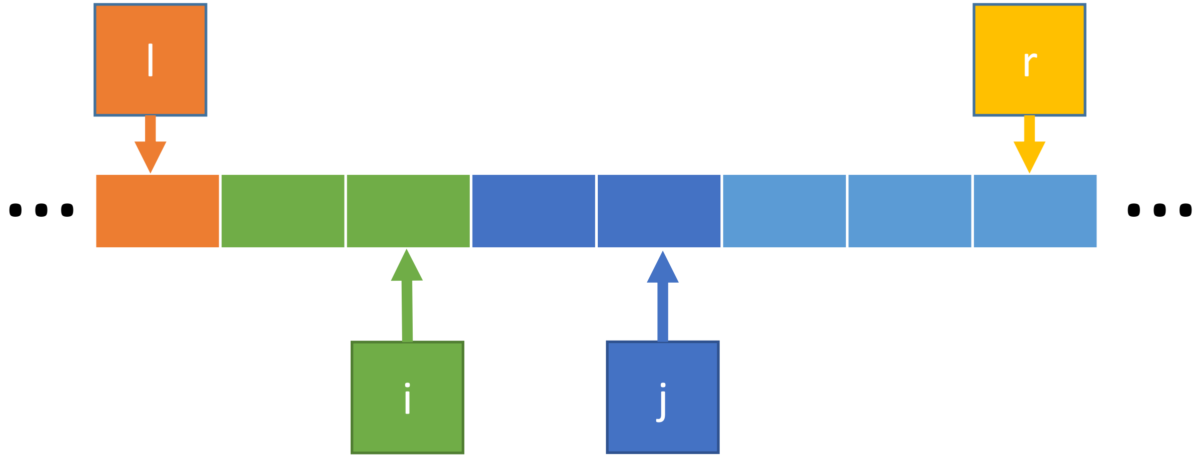
```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i - 1)
    RETURN i - 1
```

true

<u>Maintenance (**case 1**):</u>

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] < pivot
- Then a[j] and a[i] are swapped
- By (2), a[i] was > pivot so now a[i] < pivot and a[j] > pivot

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

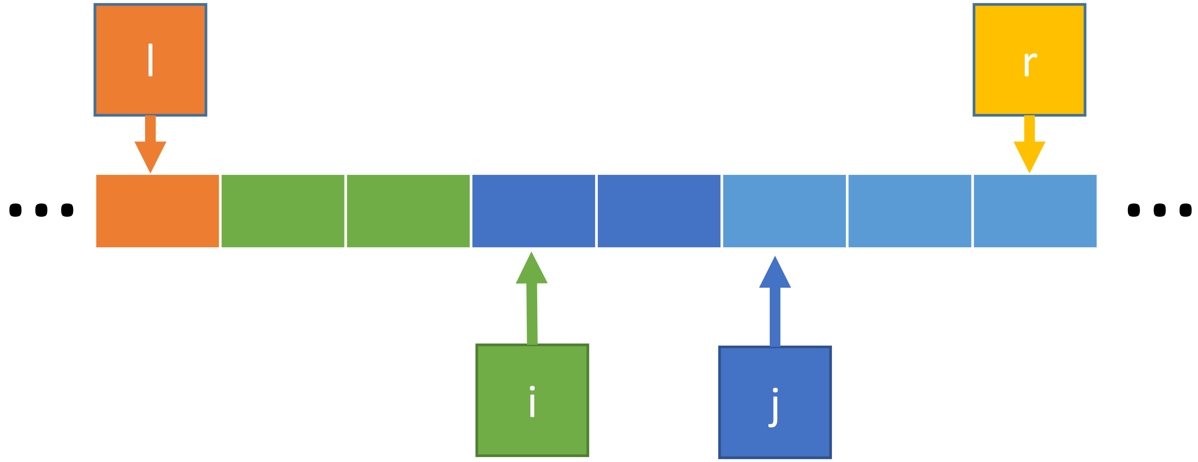
<u>Maintenance (**case 1**)</u>:

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] < pivot
- Then a[j] and a[i] are swapped
- By (2), a[i] was > pivot so now a[i] < pivot and a[j] > pivot

true →

```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i − 1)
    RETURN i − 1
```

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

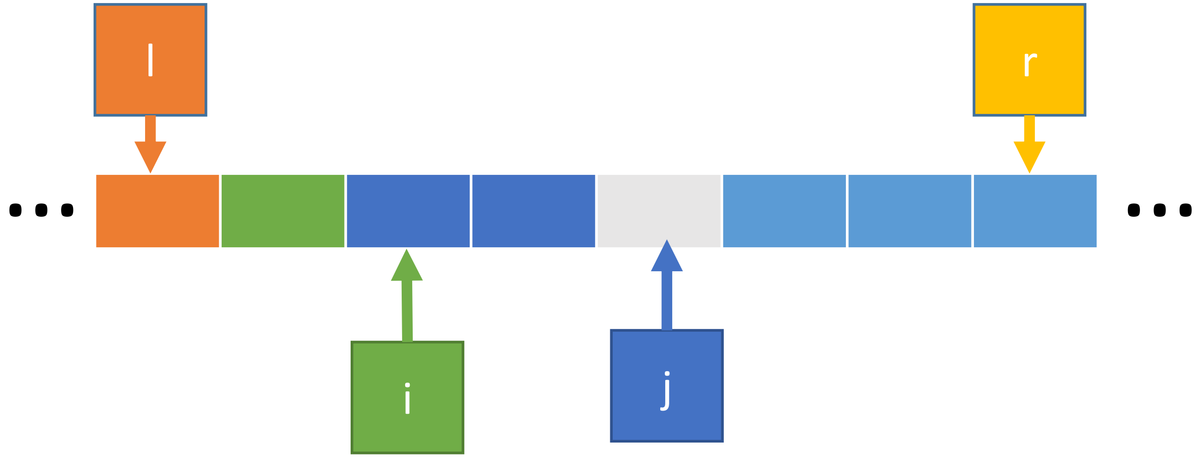
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

Maintenance (**case 1**):

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] < pivot → true
- Then a[j] and a[i] are swapped
- By (2), a[i] was > pivot so now a[i] < pivot and a[j] > pivot
- Incrementing i and j satisfies 1 and 2

```
FUNCTION Partition(a, l, r)
   pivot_value = a[l]
   i = l + 1
   FOR j IN [l + 1 ..< r]
      IF a[j] < pivot_value
         swap(a, i, j)
         i = i + 1
   swap(a, l, i − 1)
   RETURN i − 1
```

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

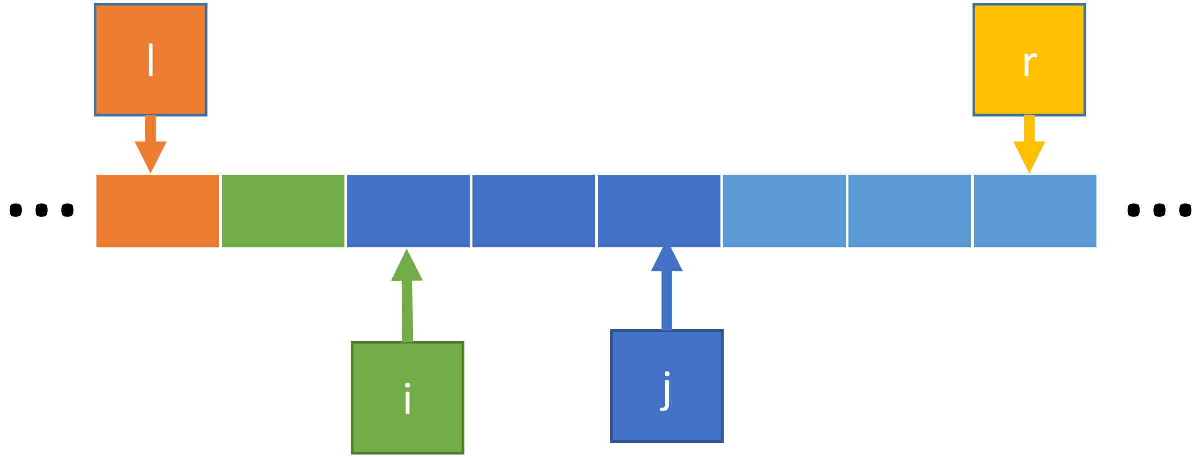
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

Maintenance (**case 2**):

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] ≥ pivot → false

```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i - 1)
    RETURN i - 1
```

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

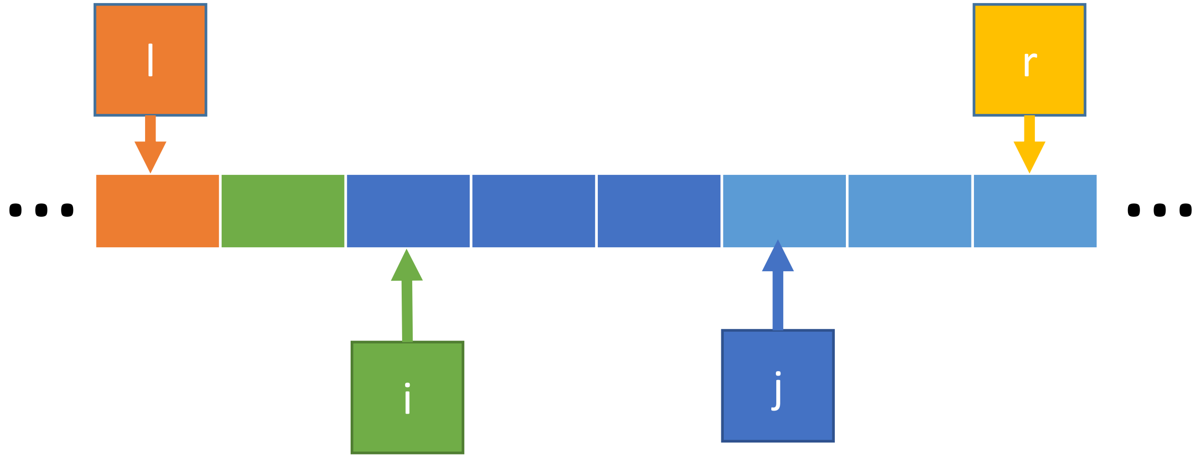
Maintenance (case 2):

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] ≥ pivot
- We do not change i so (1) holds

false

```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i - 1)
    RETURN i - 1
```

Partition Proof



Loop Invariant: At the start of the iteration with indices i and j:

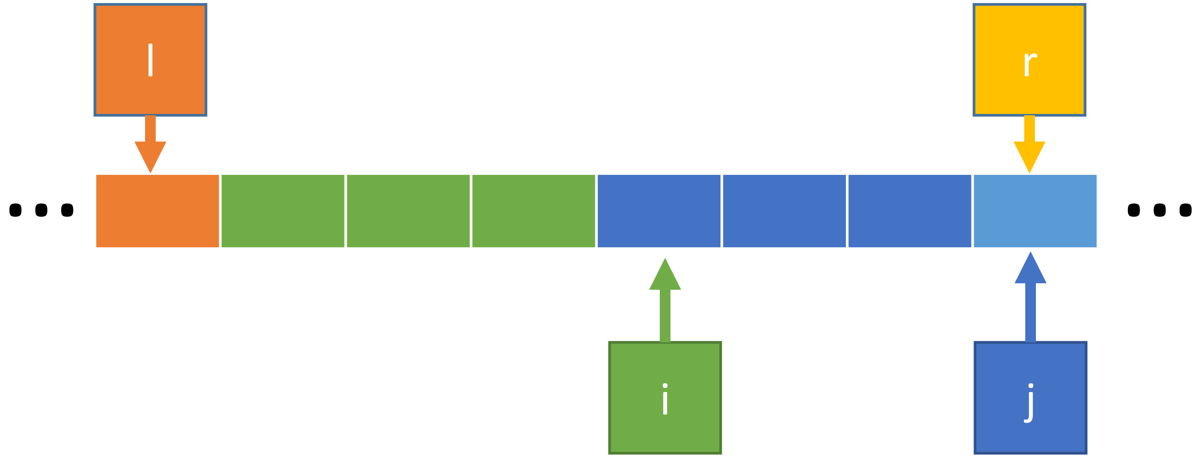
1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

```
FUNCTION Partition(a, l, r)
  pivot_value = a[l]
  i = l + 1
  FOR j IN [l + 1 ..< r]
    IF a[j] < pivot_value
      swap(a, i, j)
      i = i + 1
  swap(a, l, i - 1)
  RETURN i - 1
```

<u>Maintenance (**case 2**):</u>

- Suppose conditions 1 and 2 are met.
- Now, suppose a[j] ≥ pivot → false → IF a[j] < pivot_value
- We do not change i so (1) holds
- We increment j so (2) holds

Partition Proof



<u>Termination</u>: Show that the loop invariant holds once the loop ends

- Assume (1) and (2) are true
- Now j = r
- All items have been considered
- All items in a[l+1 ..= i-1] are < pivot
- All items in a[i ..= j-1] are ≥ pivot

Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

```
FUNCTION Partition(a, l, r)
    pivot_value = a[l]
    i = l + 1
    FOR j IN [l + 1 ..< r]
        IF a[j] < pivot_value
            swap(a, i, j)
            i = i + 1
    swap(a, l, i – 1)
    RETURN i – 1
```

Partition Proof

After the loop we perform the final swap

Loop Invariant: At the start of the iteration with indices i and j:

1. All items in a[l+1 ..= i-1] are < pivot
2. All items in a[i ..= j-1] are ≥ pivot

```
FUNCTION Partition(a, l, r)
   pivot_value = a[l]
   i = l + 1
   FOR j IN [l + 1 ..< r]
      IF a[j] < pivot_value
         swap(a, i, j)
         i = i + 1
   swap(a, l, i - 1)
   RETURN i - 1
```

What do we need to do?

Input: an array of n items in arbitrary order

Output: the same number in non-decreasing order

Assumptions: the items must be orderable (from an ordinal set)

Theorem: the Quicksort algorithm arranges all items in non-decreasing order.

1. **Lemma:** see proof by loop invariant of `Partition`
2. **Lemma** involving `QuickSort`

Theorem: the Quicksort algorithm arranges all items in non-decreasing order.

1. **Lemma 1**:

   Loop Invariant: At the start of the iteration with indices i and j:

   1. All items in `array[l+1 ..= i-1]` are < `pivot_value`
   2. All items in `array[i ..= j-1]` are ≥ `pivot_value`

   (See corresponding proof by loop invariant)

1. **Lemma 2** involving `QuickSort`

Proof by Induction in General

Some property P that we want to prove

- A base case: some statement regarding P(1)
- An inductive hypothesis: assume we know that P(n) is true
- An inductive step: if P(n) is correct then so is P(n+1) because…

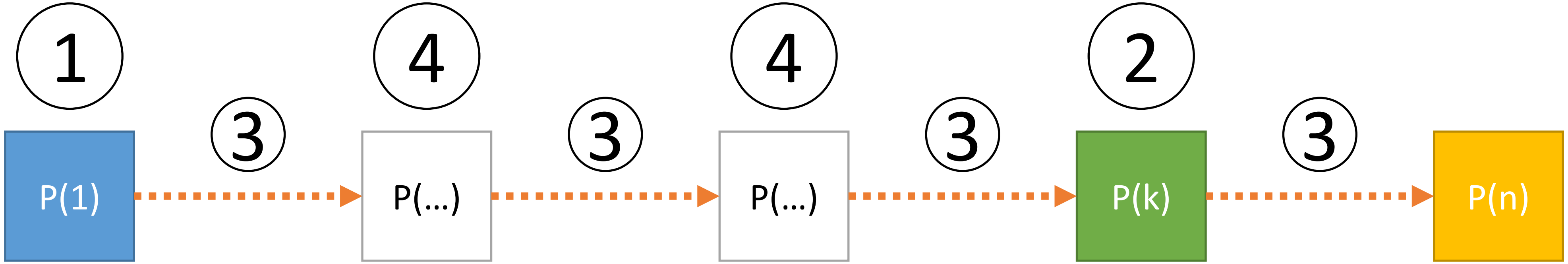
For quicksort we are going to use a slightly different form

- If P(k) where k < n is correct, then P(n) is also correct
- An inductive hypothesis: assume we know that P(k) is true
- An inductive step: if P(k) is correct then so is P(n) because…

Proof by Induction Cheat-sheet

Proof by induction that P(n) holds for all n

1. P(1) holds because <something about the code/problem>
2. Let's assume that P(k) (where k < n) holds.
3. P(n) holds because of P(k) and <something about the code>
4. Thus, by induction, P(n) holds for all n



We can infer all intermediate jumps due to steps 1 and 3.

Quicksort Proof

Proof by induction that P(n) holds for all n

- P(1) holds because …
- Let's assume that P(k) (where k < n) holds.
- P(n) holds because of P(k) and …
- Thus, by induction, P(n) holds for all n

P(n) =

Quicksort Proof

Proof by induction that P(n) holds for all n

- P(1) holds because …
- Let's assume that P(k) (where k < n) holds.
- P(n) holds because of P(k) and …
- Thus, by induction, P(n) holds for all n

P(n) = arranges all items in non-decreasing order.

- P(1)

Quicksort Proof

Proof by induction that P(n) holds for all n

- P(1) holds because …
- Let's assume that P(k) (where $k < n$) holds.
- P(n) holds because of P(k) and …
- Thus, by induction, P(n) holds for all n

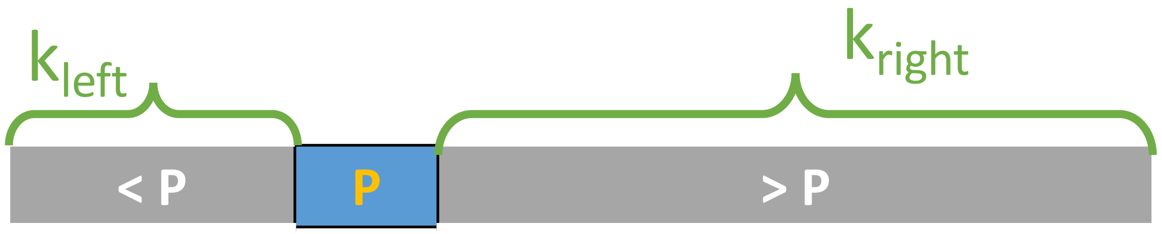
P(n) = arranges all items in non-decreasing order.

- P(1) is an array of one element, and any such array is always sorted.
- <u>Assume (hypothesis)</u>
- P(n) holds because:

Proof by induction that P(n) holds for all n

- P(1) holds because …
- Let's assume that P(k) (where k < n) holds.
- P(n) holds because of P(k) and …
- Thus, by induction, P(n) holds for all n

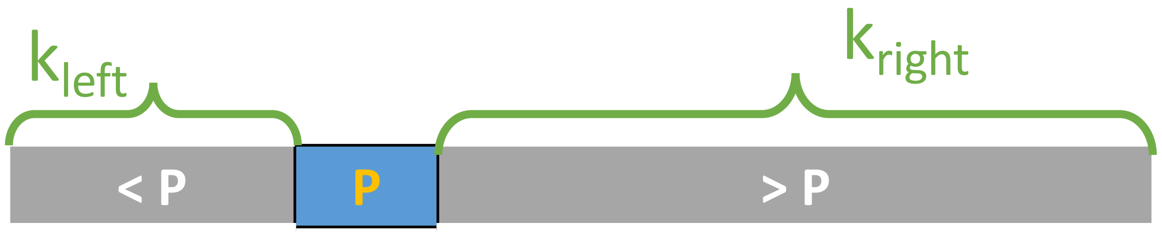
Quicksort Proof



P(n) = arranges all items in non-decreasing order.

- P(1) is an array of one element, and any such array is always sorted.
- <u>Assume (hypothesis)</u> that P(k) is correct for k < n
- P(n) holds because:
  - Let k_{left}, k_{right} = the lengths of the left and right subarrays
  - k_{left}, k_{right} < n (strictly less than n)
  - By our **inductive hypothesis**, the left and right subarrays are correctly sorted
  - The **partition loop-invariant** guarantees that the pivot is in the correct spot

Quicksort Proof



Proof by induction that P(n) holds for all n

- P(1) holds because …
- Let's assume that P(k) (where k < n) holds.
- P(n) holds because of P(k) and …
- Thus, by induction, P(n) holds for all n

P(n) = arranges all items in non-decreasing order.

- P(1) is an array of one element, and any such array is always sorted. — Base case

- Assume (hypothesis) that P(k) is correct for k < n — Inductive Hypothesis

- P(n) holds because: — Inductive Step
  - Let k_{left}, k_{right} = the lengths of the left and right subarrays
  - k_{left}, k_{right} < n (strictly less than n)
  - By our **inductive hypothesis**, the left and right subarrays are correctly sorted
  - The **partition loop-invariant** guarantees that the pivot is in the correct spot

Theorem: the Quicksort algorithm arranges all items in non-decreasing order.

1. **Lemma 1**:

   Loop Invariant**:** At the start of the iteration with indices i and j:

   1. All items in `array[l+1 ..= i-1]` are < `pivot_value`
   2. All items in `array[i ..= j-1]` are ≥ `pivot_value`

   (See corresponding proof by loop invariant)

1. **Lemma 2**:

   P(n) = Quicksort arranges all items in non-decreasing order.

   (See corresponding proof by induction)