Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 24
Recursive sorting II

FIT 1008&2085
Introduction to Computer Science

MONASH University
Information Technology

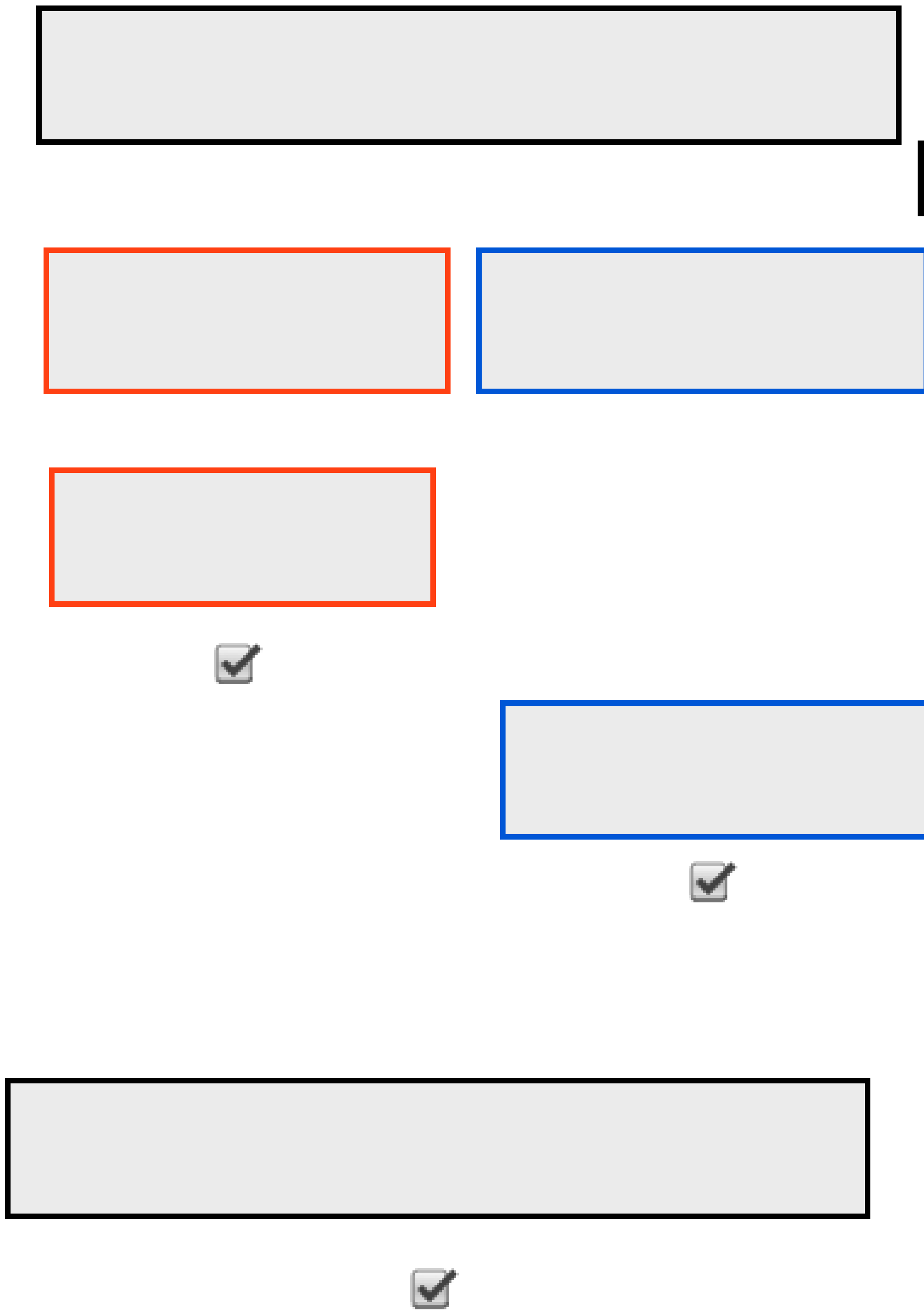
Overview

- To review what a “**divide and conquer**” algorithm is

- To review in more depth two different “divide and conquer” sorting algorithms:
  - **Merge Sort**
  - **Quick Sort**

- To be able to **implement** them and compare their efficiency for different classes of inputs

Divide and Conquer: **Sorting**



Divide the array into **2 parts**

Sort the first part

Sort the second part

Combine

Divide and Conquer: **Sorting**

General Idea

```python
def sort(array):
    if len(array) > 1:
        split(array, first_part, second_part)
        sort(first_part)
        sort(second_part)
        combine(first_part, second_part)
```

- **Merge Sort** has a simple split and a elaborate combine
- **Quick Sort** has a elaborate split and a simple combine

Quicksort

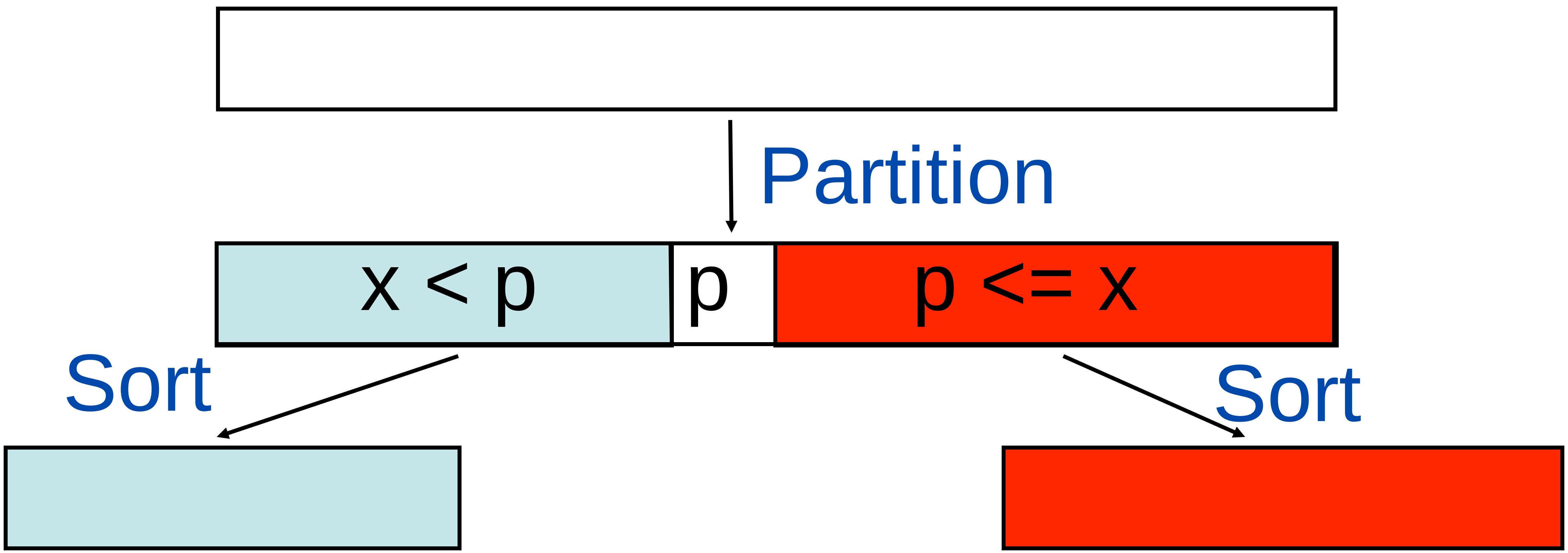
Top-10 algorithms 20th century (SIAM)

Quick Sort

- **Partition** the list
- Sort the first part (recursively)
- Sort the second part (recursively)
- (Combine: there is nothing to do!)

Partition

- Choose an item in the list, called it the **pivot**.
- The **first part** consists of all those **items** which are **less than the pivot**.
- The **second part** consists of all those **items larger than or equal to the pivot (except the pivot)**.



- **Partition:** Elaborate, based on a pivot p.

- **Combination:** Simple append, pivot in the middle.

Ideally, what should the pivot be?

A) The smallest element of the list.

B) The largest element of the list.

C) The middle element of the input list.

D) The middle element of the output list.

E) Something else.

F) It actually makes no difference.

Given a list of size N, how efficiently can the median be computed?

A) $O(1)$

B) $O(\log N)$

C) $O(N)$

D) $O(N \log N)$

E) $O(N^2)$

Example Partition

array:

5	89	35	14	24	15	37	13	20	7	70

start:0 **end:10**

Example Partition

`array:`

5	89	35	14	24	15	37	13	20	7	70

Randomly choose a pivot, which happens to be in the middle

Example Partition

`array:`

5	89	35	14	24	15	37	13	20	7	70

partition:

5	89	35	14	24	15	37	13	20	7	70

result

7	14	5	13	15	35	37	89	20	24	70

pivot position: 4

note that the pivot defines the boundaries

sort first half (using QS), sort second half (using QS)

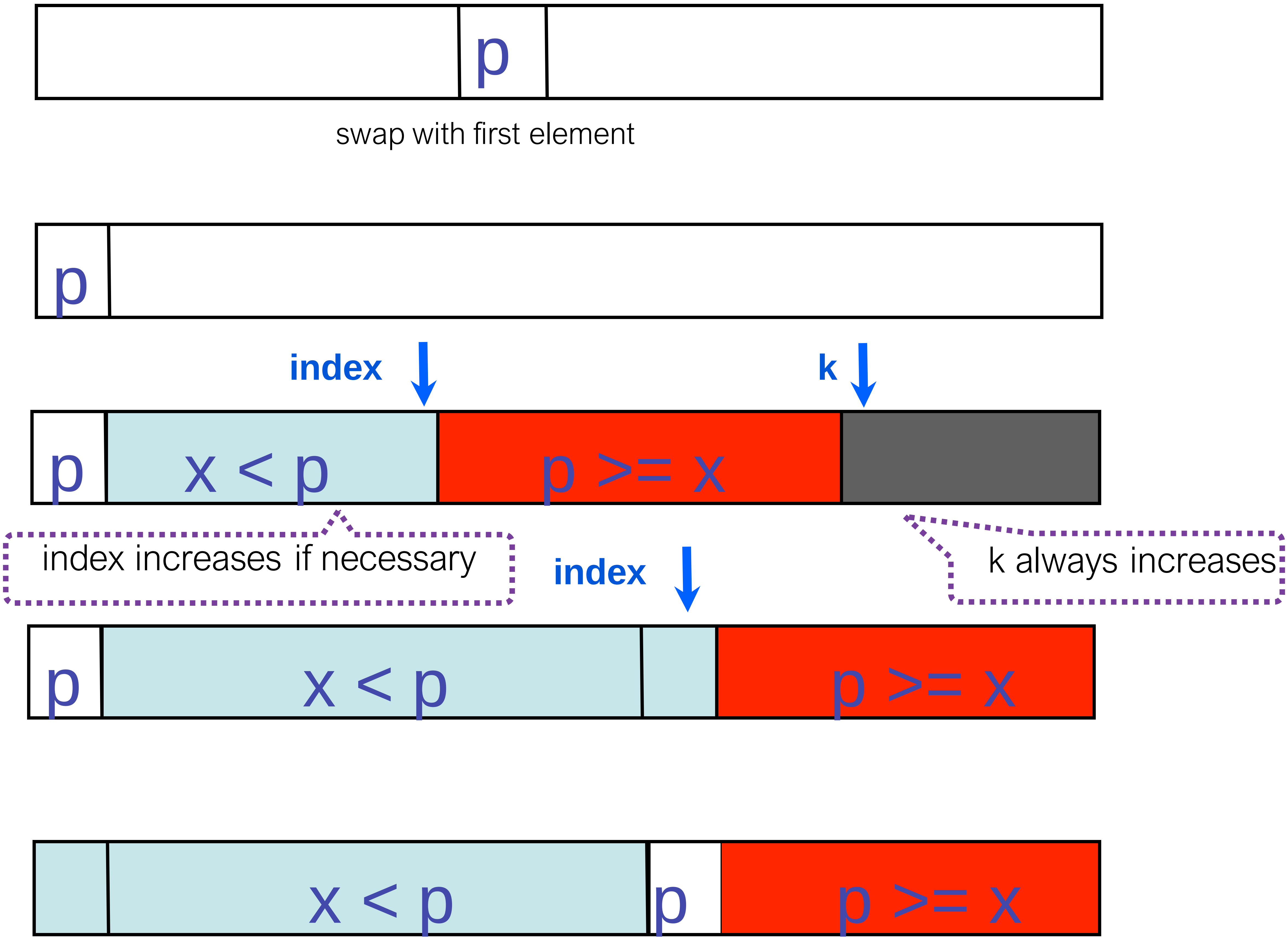
Quicksort

```
def quick_sort(array):
```

Quicksort

```python
def quick_sort(array):
    start = 0
    end = len(array)-1
    quick_sort_aux(array, start, end)

def quick_sort_aux(array, start, end):
    if start < end:
        boundary = partition(array, start, end)
        quick_sort_aux(array, start, boundary-1)
        quick_sort_aux(array, boundary+1, end)
```


p
swap with first element
p
index
k
p
x < p
p >= x
index increases if necessary
k always increases
index
p
x < p
p >= x
x < p
p
p >= x

Example Partition

randomly pick element in position 5

array:

5	89	35	14	24	15	37	13	20	7	70

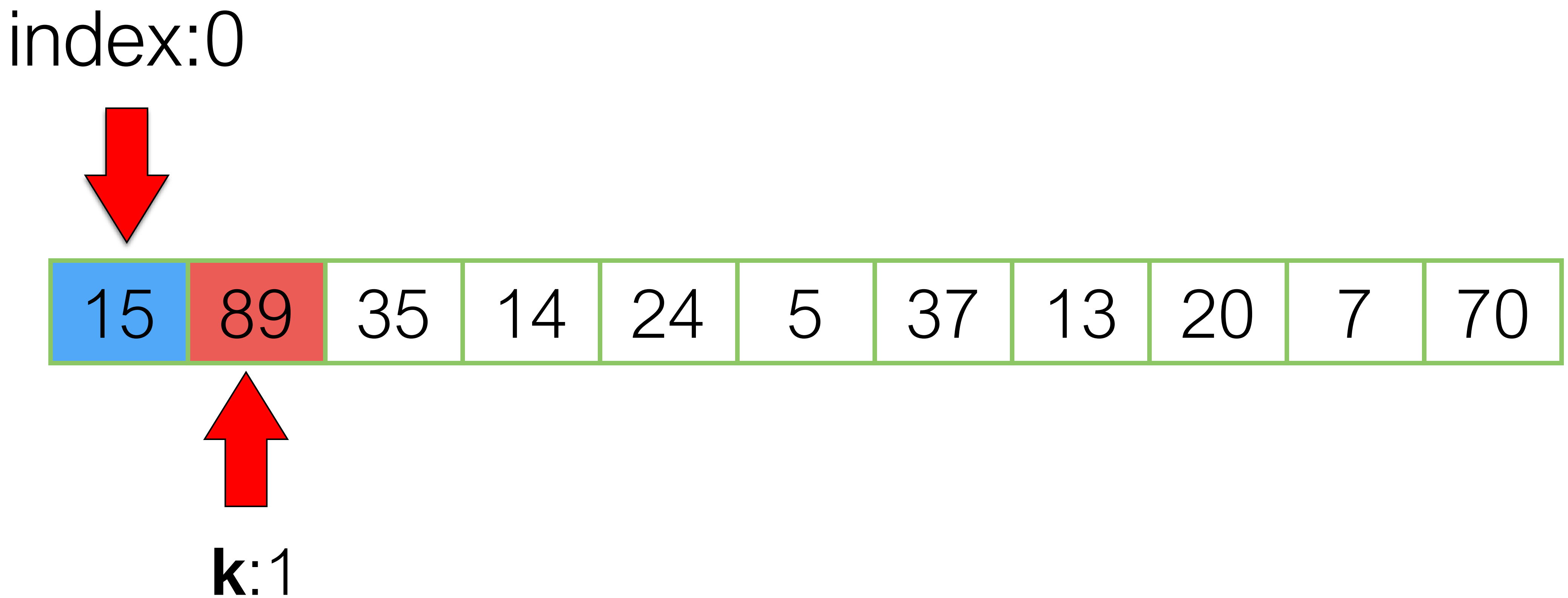
15	89	35	14	24	5	37	13	20	7	70

Example Partition

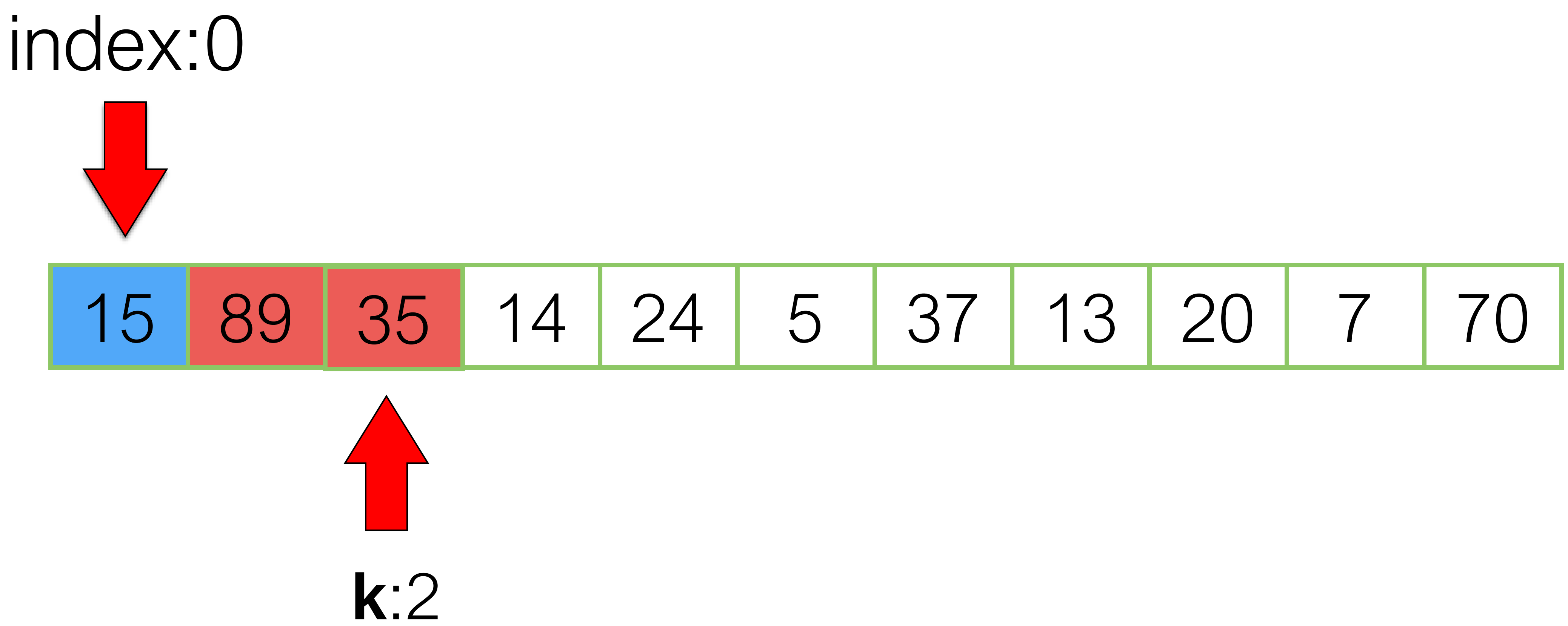
15	89	35	14	24	5	37	13	20	7	70

k:1

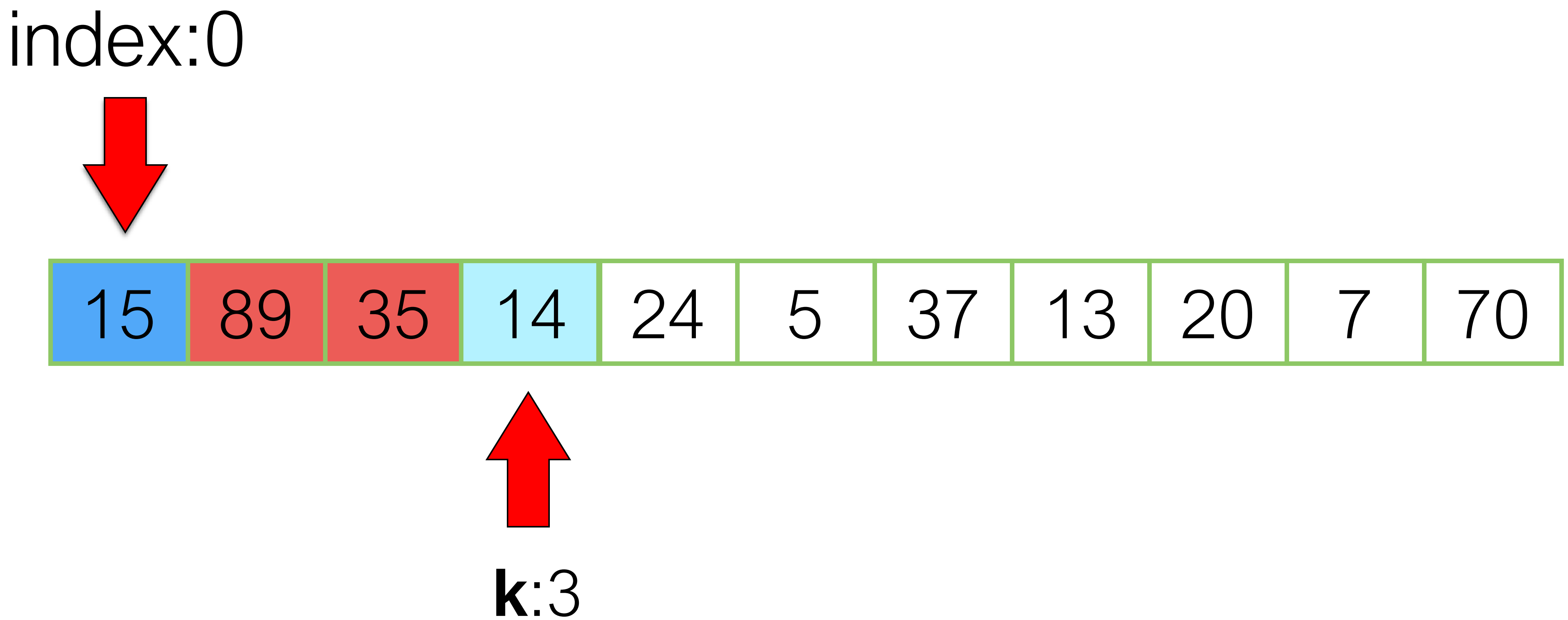
Example Partition



Example Partition



Example Partition



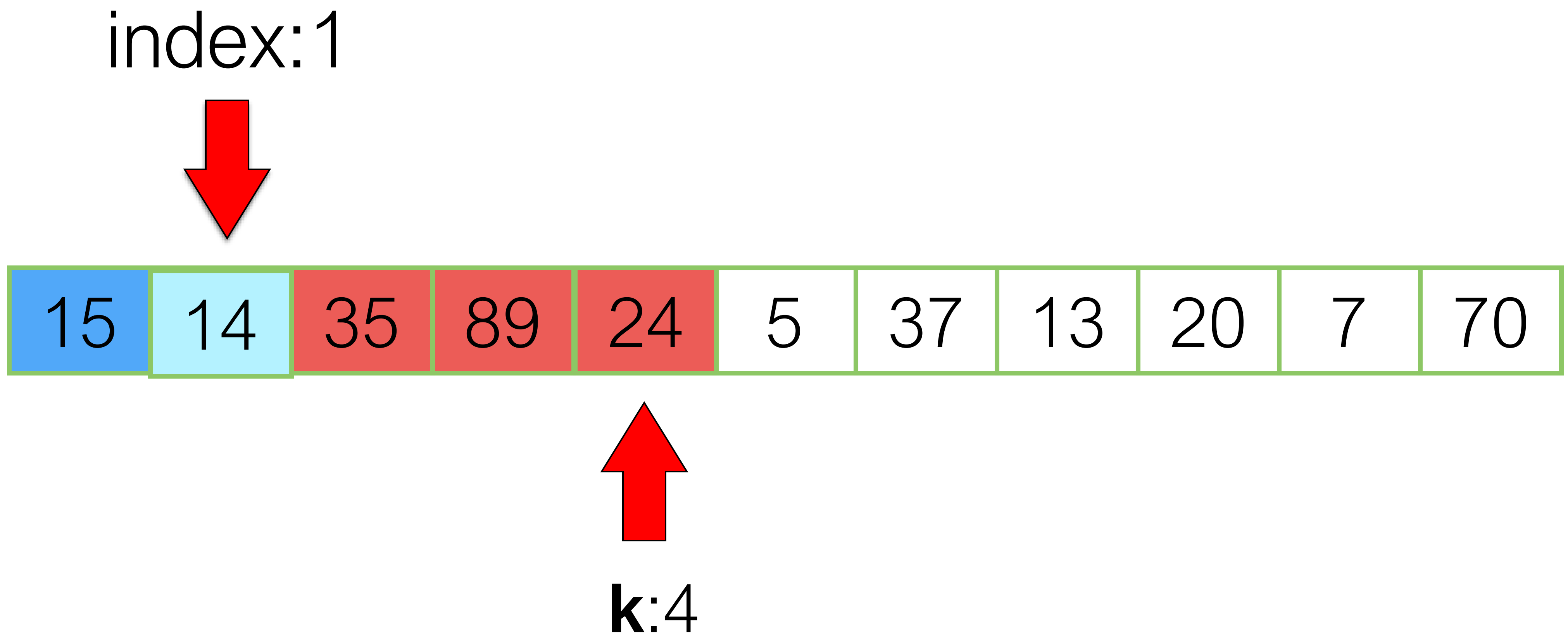
Example Partition

index:1

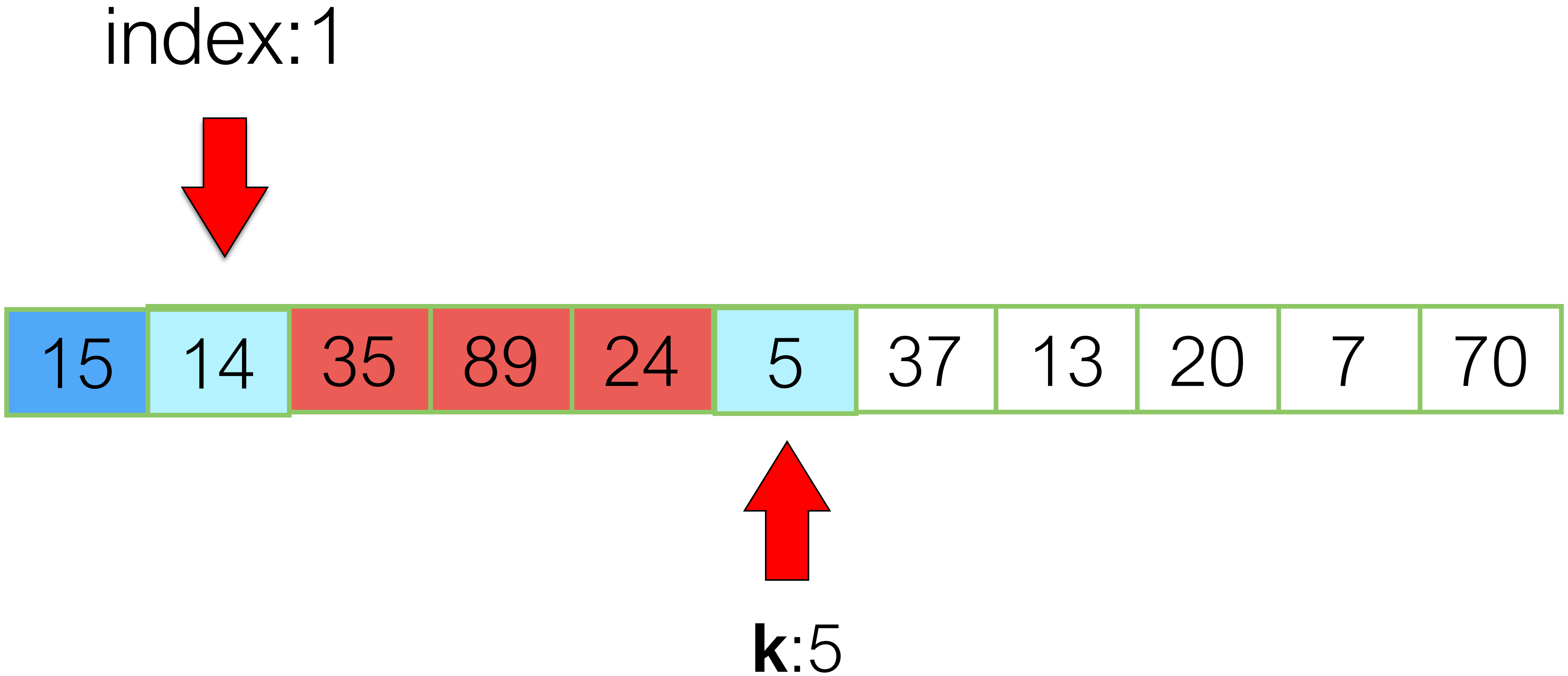
15	14	35	89	24	5	37	13	20	7	70

k:3

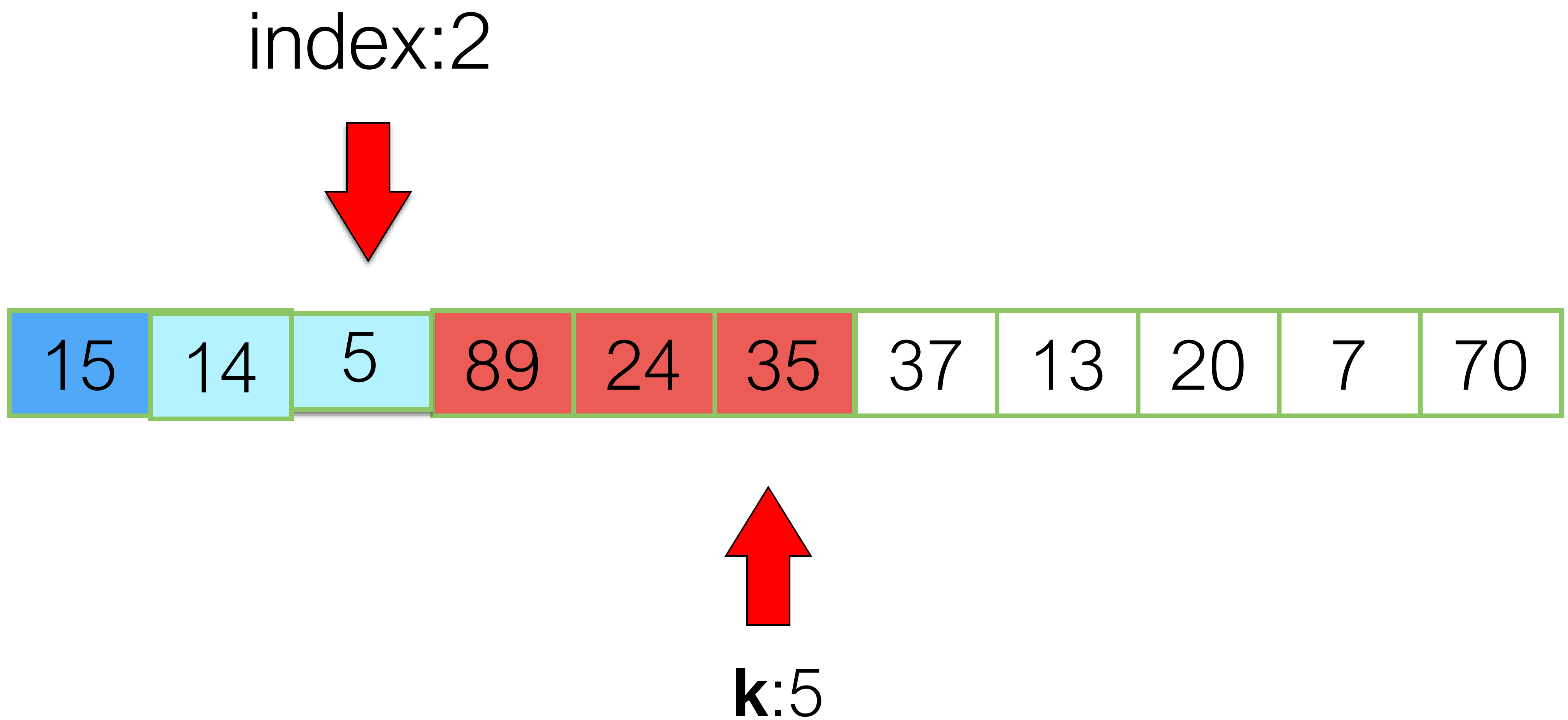
Example Partition



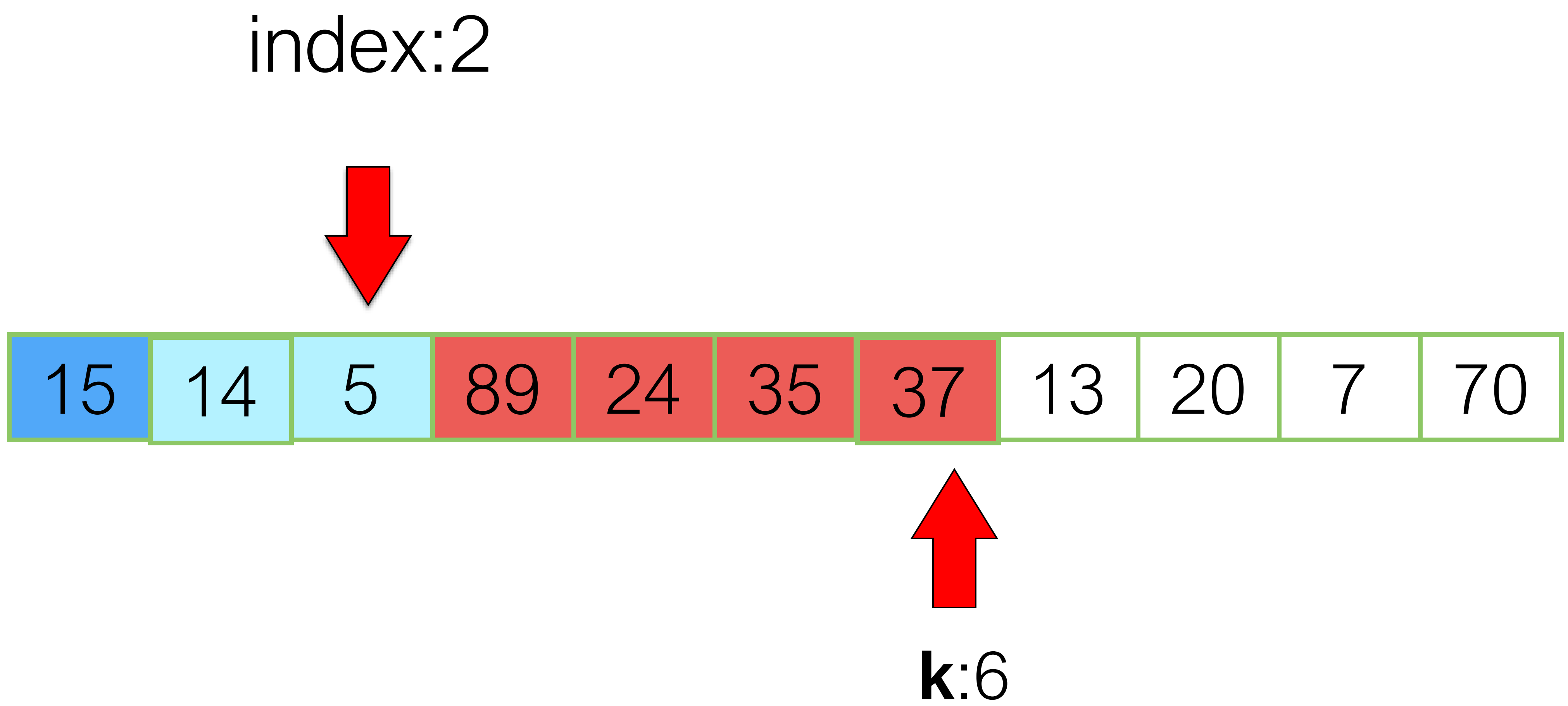
Example Partition



Example Partition

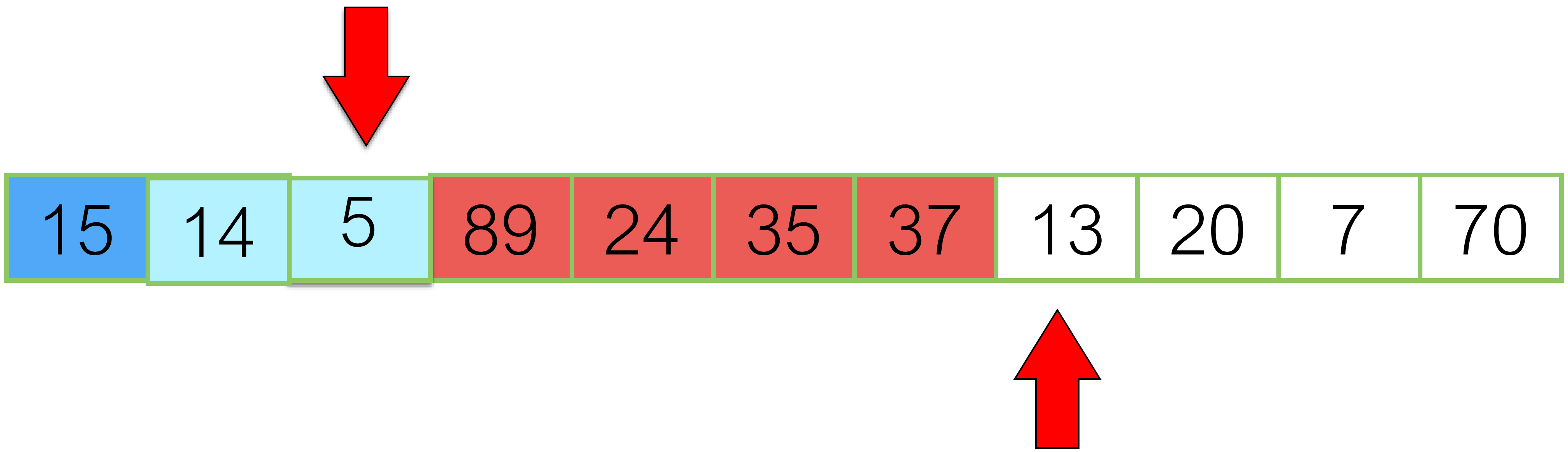


Example Partition



Example Partition

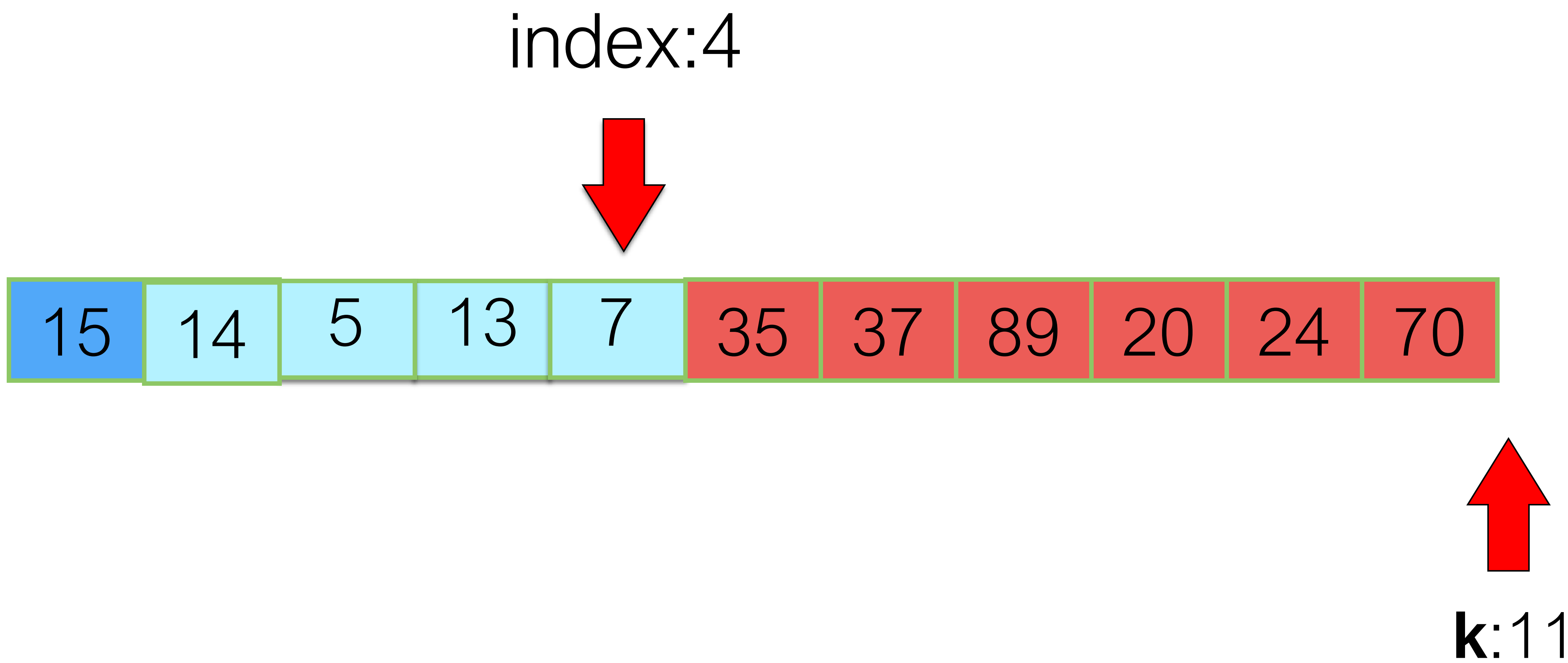
index:2



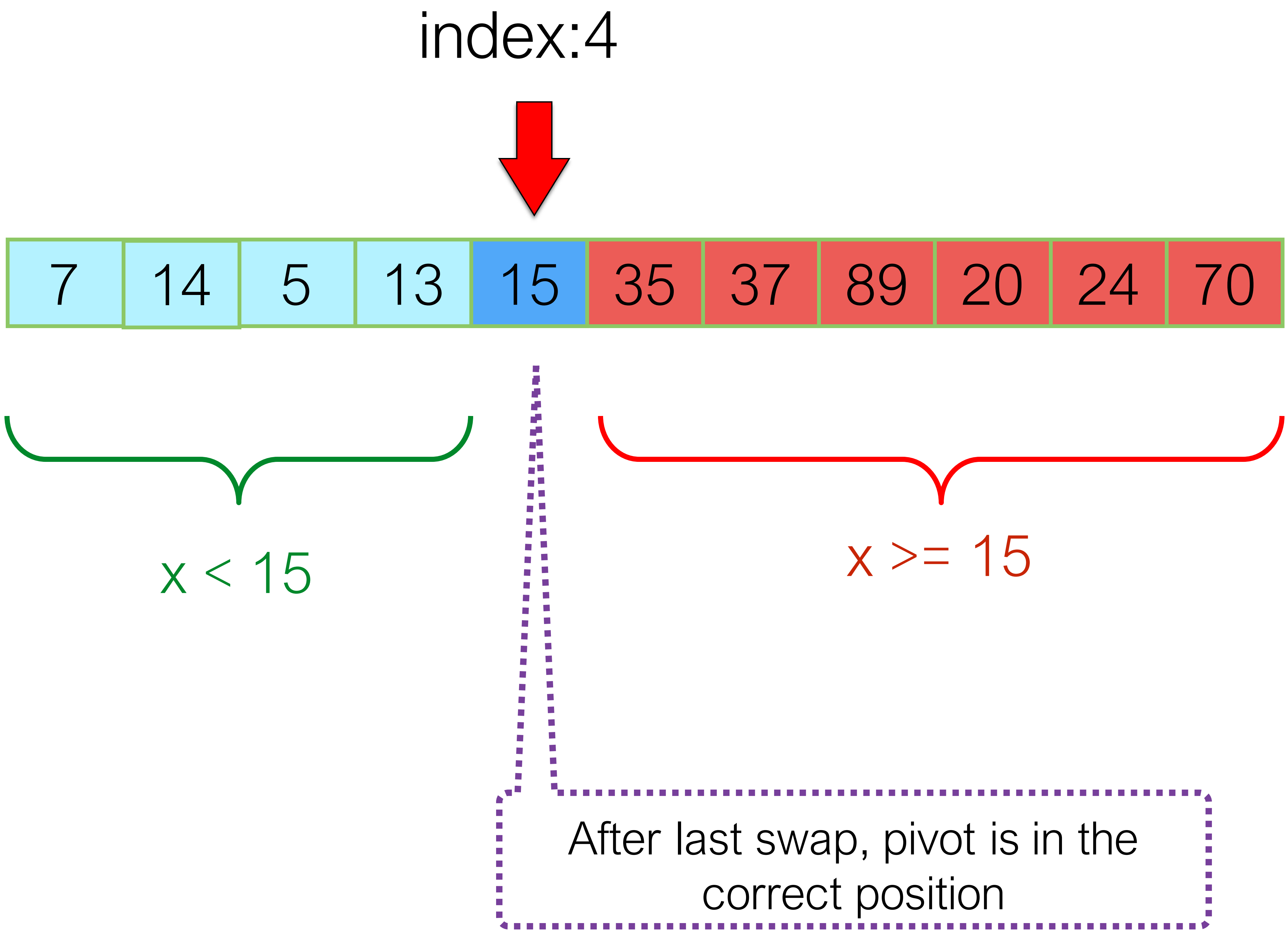
k:7

etc…

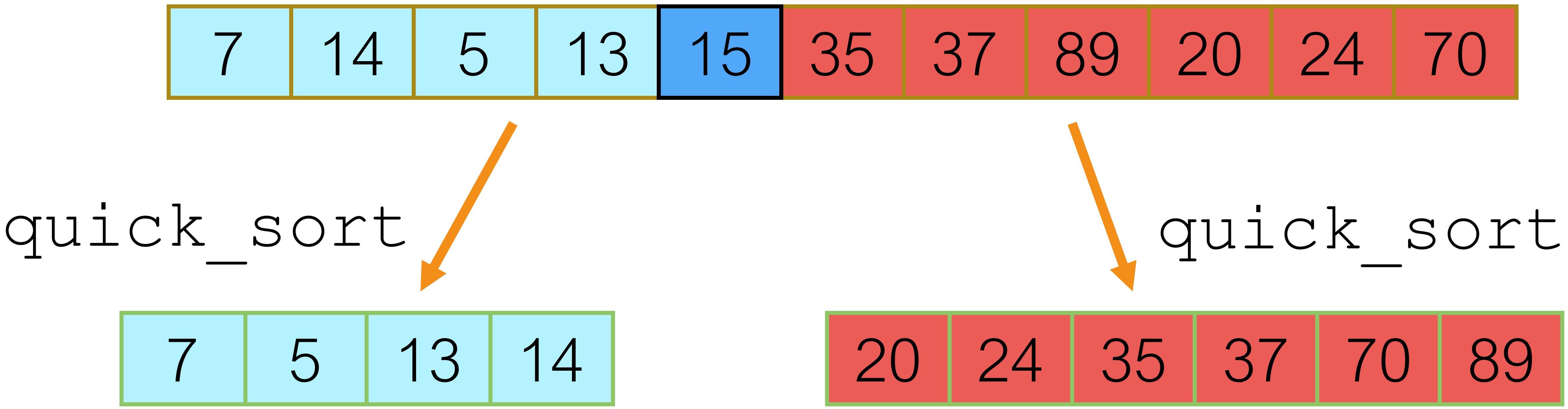
Example Partition

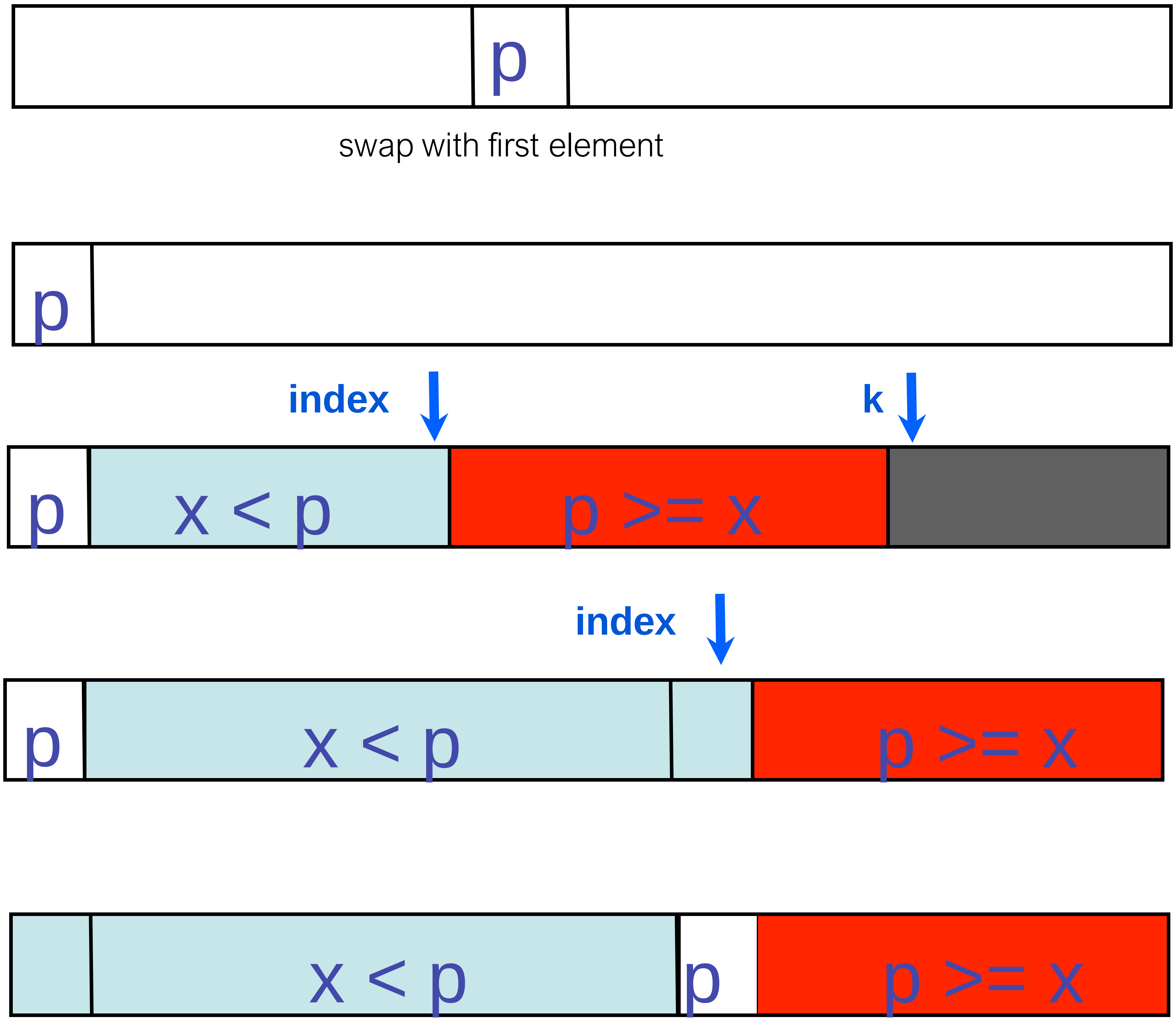


Example Partition



Example Partition




p
swap with first element
p
index
k
p
x < p
p >= x
index
p
x < p
p >= x
x < p
p
p >= x

```python
def swap(array, i, j):
    array[i], array[j] = array[j], array[i]
```

```python
def partition(array, start, end):
```

```python
def partition(array, start, end):
        mid = (start+end)//2
        pivot = array[mid]
        swap(array, start, mid)
        index = start
        for k in range(start+1,end+1):
            if array[k] < pivot:
                index += 1
                swap(array, k, index)
        swap(array, start, index)
        return index
```

Found an element that belongs in 1… index

swap and update index to maintain invariant

What is the time complexity of the partition method?

```python
def partition(array, start, end):
        mid = (start+end)//2
        pivot = array[mid]
        swap(array, start, mid)
        index = start
        for k in range(start+1,end+1):
            if array[k] < pivot:
                index += 1
                swap(array, k, index)
        swap(array, start, index)
        return index
```

A) O(log N)

B) O(N)

C) O(N log N)

D) $O(N^2)$

What is the best-case time complexity of quicksort?

```python
def quick_sort(array):
    start = 0
    end = len(array)-1
    quick_sort_aux(array, start, end)

def quick_sort_aux(array, start, end):
    if start < end:
        boundary = partition(array, start, end)
        quick_sort_aux(array, start, boundary-1)
        quick_sort_aux(array, boundary+1, end)
```

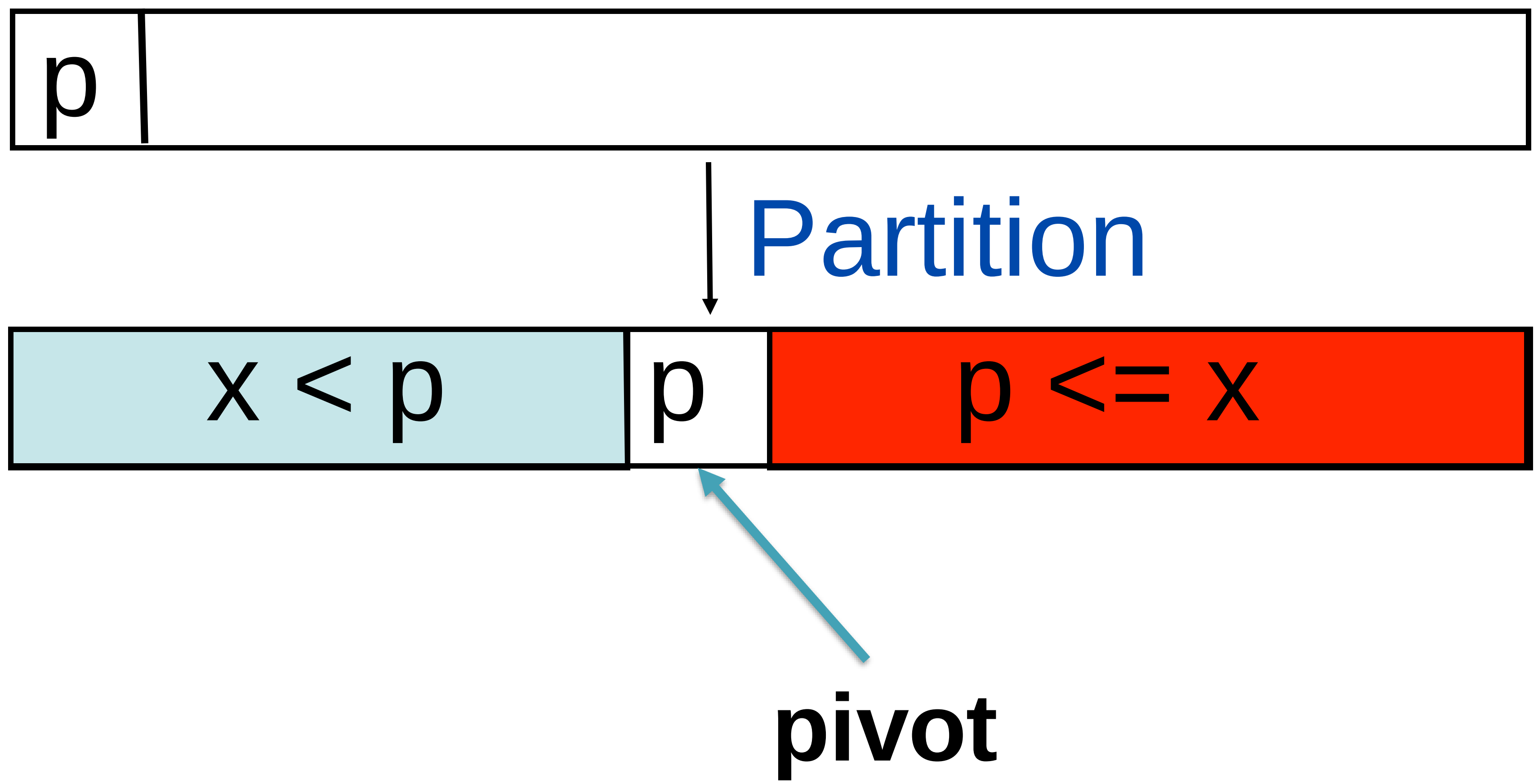
A) O(log N)

B) O(N)

C) O(N log N)

D) $O(N^2)$

Quicksort: Number of partitions depends on the pivot



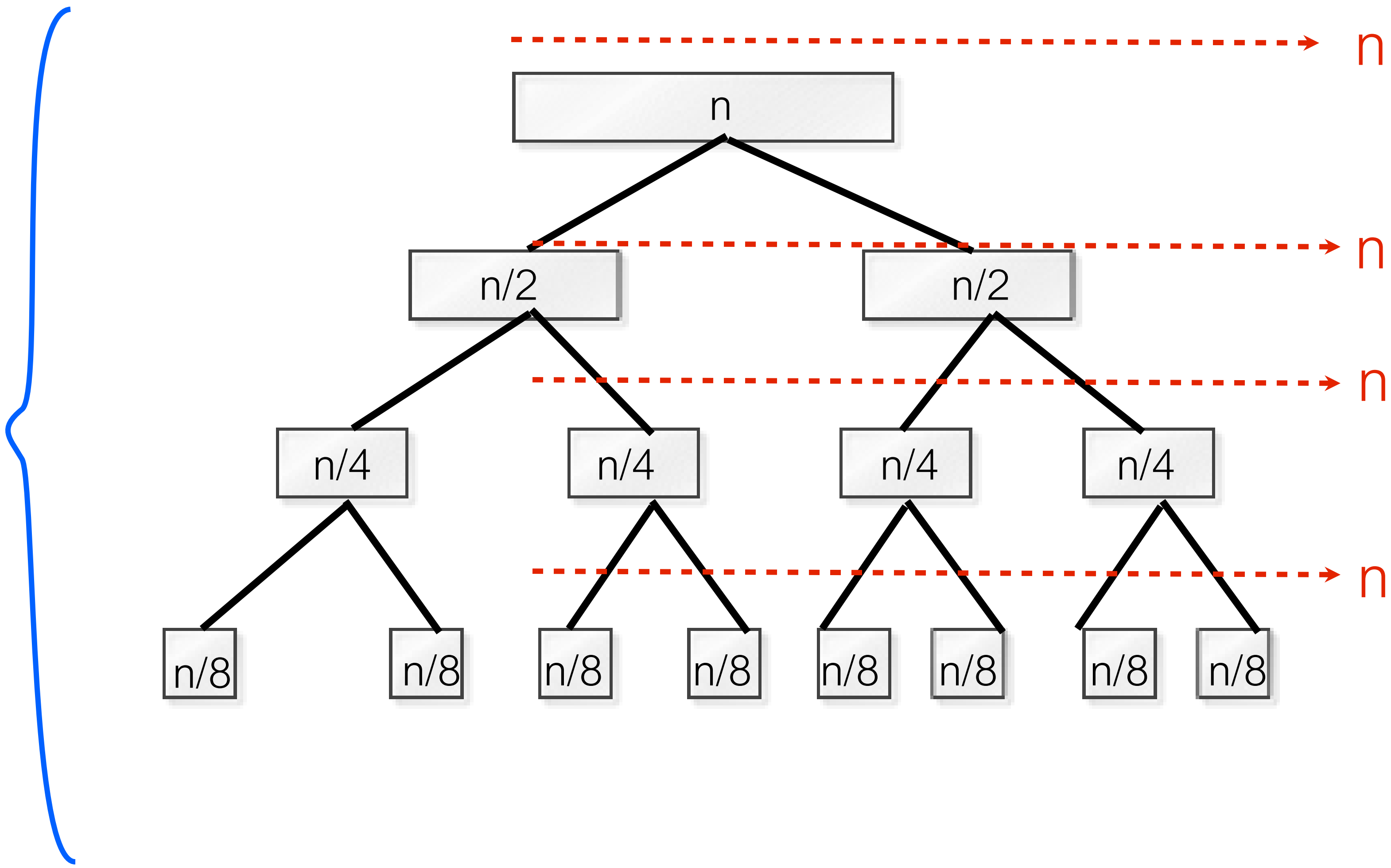
Best case: The size of the problem is reduced by half with every partition

Worst case: The size of the problem is reduced by 1 with every partition

Quick sort's best case

partition is O(n)

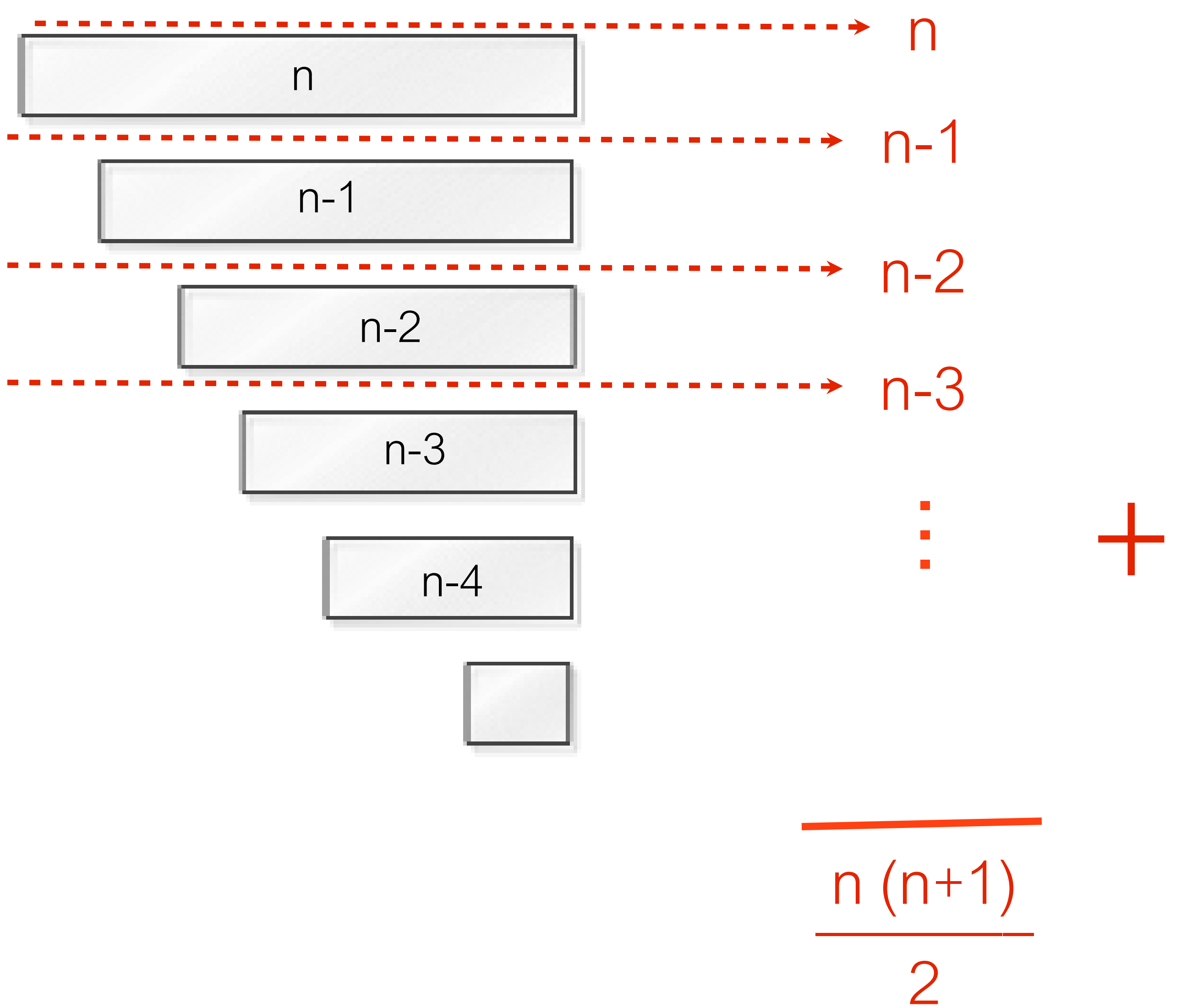
height is O(log n)



Running time in the best case: O(n log n)

Quick sort's worst case

partition is O(n)



Running time in the worst case: $O(n^2)$

Summary

	Best case	Worst case
Quicksort	O(n log n)	$O(n^2)$
Mergesort	O(n log n)	O(n log n)

How common is quicksort's worst case?

Not too common if choosing a random pivot.

Summary

Divide and Conquer and Recursive Algorithms (for sorting).

Merge Sort

- Easy: Split
- Elaborate: merge method

Quick Sort

- Elaborate split: partition method
- Easy combination