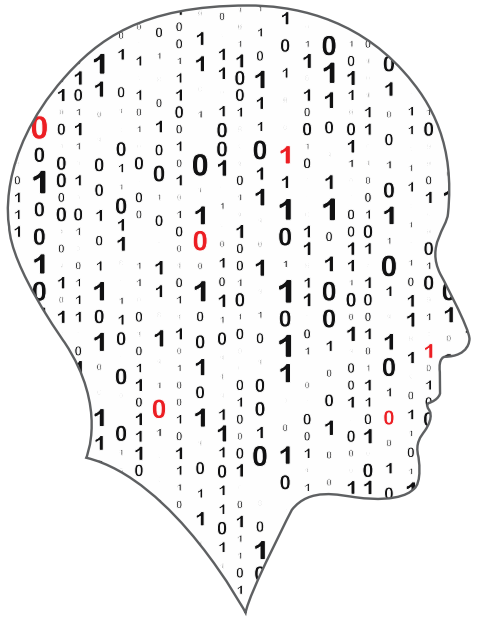
PDST
Professional Development Service for Teachers
An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

An Roinn Oideachais agus Scileanna
Department of Education and Skills

National Workshop 6



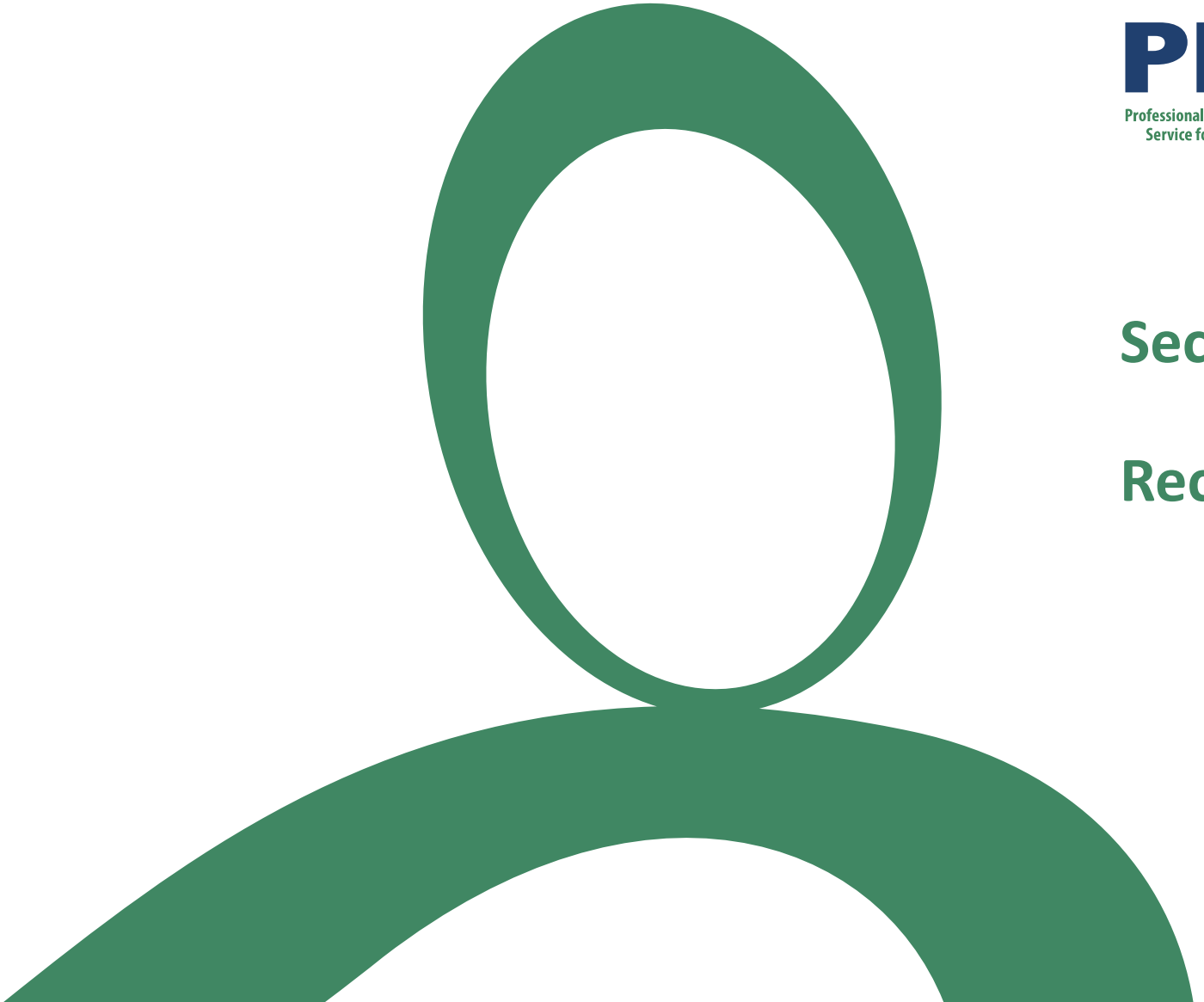
LEAVING CERTIFICATE
COMPUTER SCIENCE

Session Schedule

Section 1	*Recap on Algorithms*
Section 2	*Quicksort*
Section 3	*Introduction to Algorithmic Complexity*
Section 4	*Analysis of Searching Algorithms (Linear & Binary) - breakout*
Section 5	*Final Reflection*

By the end of this session participants will have:

- revisited what an algorithm is
- developed their understanding of the Quicksort sorting algorithm
- developed a deeper understanding of algorithmic complexity
- used an analysis framework to compare the time complexity of the aforementioned algorithms and in doing so deepened their understanding of these algorithms.
- reflected on ideas to facilitate the effective learning of algorithms in their own classrooms.



PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Section I

Recap on Algorithms

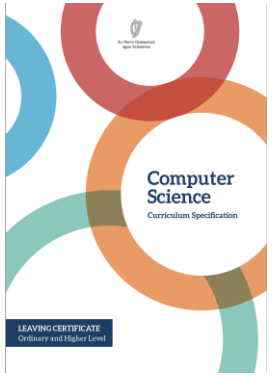
Algorithms and the Specification

"The core concepts are developed theoretically and applied practically. In this way, conceptual classroom-based learning is intertwined with experimental computer lab-based learning throughout the two years of the course."

NCCA Curriculum specification, Page 20

Strand 1: Practices and principles	Strand 2: Core concepts	Strand 3: Computer science in practice
▸ Computers and society ▸ Computational thinking ▸ Design and development	▸ Abstraction ▸ Algorithms ▸ Computer systems ▸ Data ▸ Evaluation/Testing	▸ Applied learning task 1 - Interactive information systems ▸ Applied learning task 2 - Analytics ▸ Applied learning task 3 - Modelling and simulation ▸ Applied learning task 4 - Embedded systems

LCCS Learning Outcomes



2.5 use pseudo code to outline the functionality of an algorithm

2.6 construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement

2.7 implement algorithms using a programming language to solve a range of problems

2.8 apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm

2.9 assemble existing algorithms or create new ones that use functions (**including recursive**), procedures, and modules

2.10 explain the common measures of algorithmic efficiency using any algorithms studied

Activity #1: Features of an algorithm

Instructions :

1. Watch Video (*The Secret Rules of Modern Living, Marcus Du Sautoy*)

2. Go to menti.com and use the code 6167 5944 – "What features of an algorithm are demonstrated in the video?"

https://www.youtube.com/watch?v=kiFfp-HAu64 (23:44 – 23:53)

3. In what other contexts do you think the Gale-Shapley algorithm could be used?

What is an algorithm?

> "An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined that it can be translated into computer language and executed by machine"
>
> *Source: Knuth, D The Art of Computer Programming (Vol. 1, Fundamental Algorithms, 3rd ed.)*

Donald Knuth

According to Knuth, an algorithm has five important features:

Finiteness	An algorithm must always terminate after a finite number of steps.
Definiteness	Each step must be precisely defined.
Input	An algorithm has zero or more inputs.
Output	An algorithm has one or more outputs, which have a specified relation to the inputs.
Effectiveness	All operations to be performed must be sufficiently basic that they can in principle be done exactly and in finite length of time by someone using pencil and paper.

PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Section II

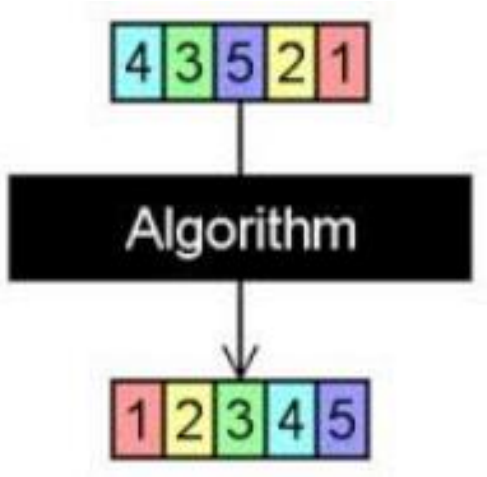
Sorting Algorithms - Quicksort

Quicksort

An algorithm that maps the following input/output pair is called a sorting algorithm:

Input: A list or array, A, that contains n orderable elements: $A[0, 1, \ \ldots \ , n-1]$.

Output: A sorted permutation of A, called B, such that $B[0] \leq B[1] \leq \cdots \leq B[n-1]$.



A general sorting algorithm devised by

Tony Hoare in the late 1950s

Quicksort (The Basic Idea)

PDST Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

DIVIDE

1. Pick some number p from the list – called the pivot

2. Partition all the data into:

A. The values less than the pivot (call this the left list)
B. The pivot (call this the middle list)
C. The values greater than the pivot (call this the right list)

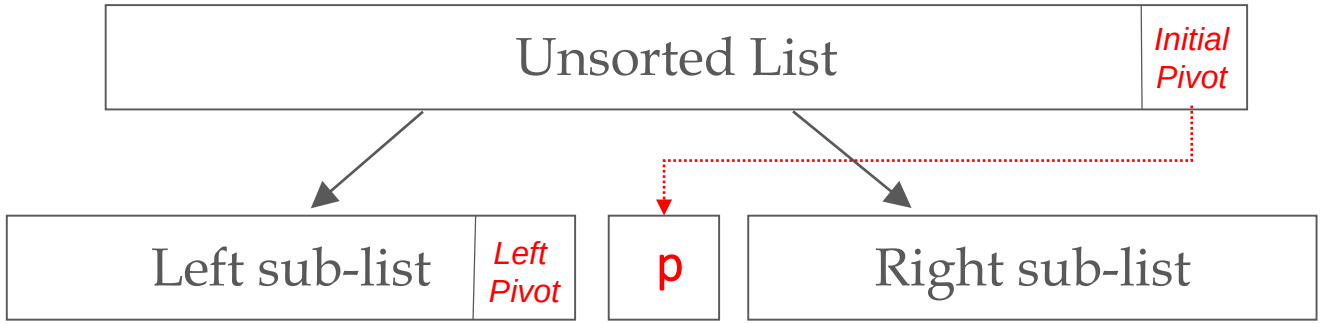
CONQUER

3. Quicksort the left list (A)

4. Quicksort the right list (B)

5. The answer is left list + middle list + right list

Partitioning



STEP 1. Choose the rightmost element in the list as the `pivot`

STEP 2. Create three empty lists called `left_list`, `middle_list` and `right_list`

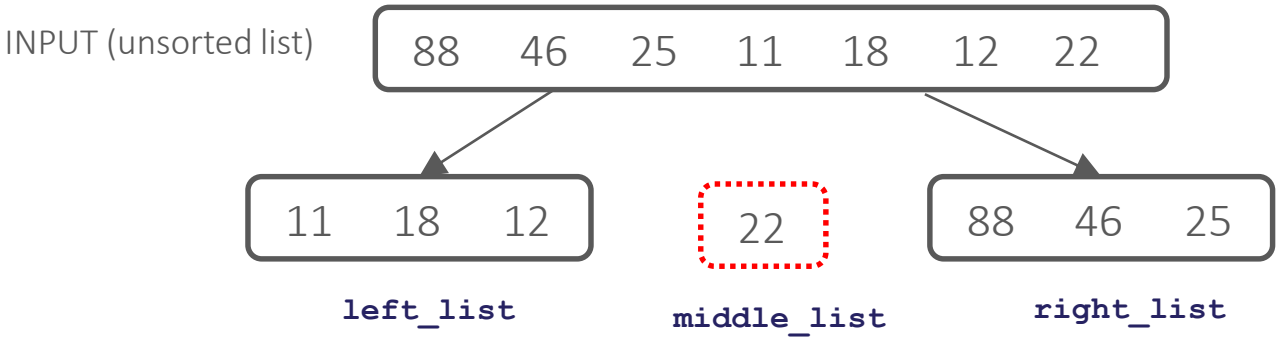
STEP 3. for each `key` (item) in the list

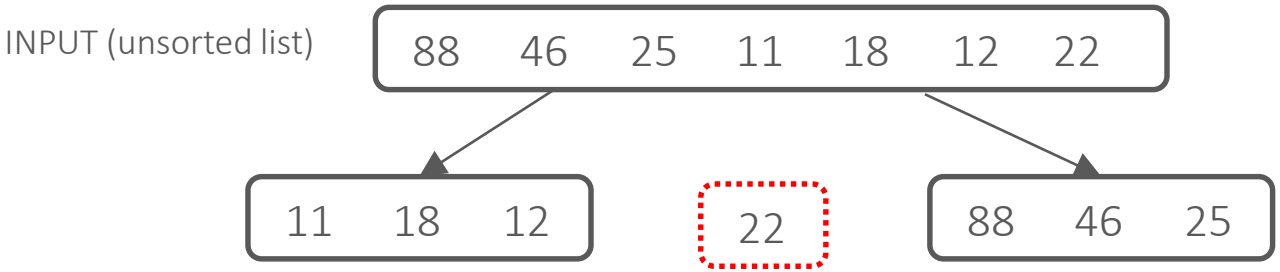
- if `key` is < `pivot` add it to `left_list`
- if `key` is == `pivot` add it to `middle_list`
- if `key` is > `pivot` add it to `right_list`

```
def quick_sort(L):
    left_list = []
    middle_list = []
    right_list = []

    # Base case
    if len(L) <=1:
        return(L)

    # Set pivot to the last element in the list
    pivot = L[len(L)-1]

    # Iterate through all elements (keys) in L
    for key in L:
        if key < pivot:
            left_list.append(key)
        elif key == pivot:
            middle_list.append(key)
        else:
            right_list.append(key)

    # Repeat the quicksort on the sub-lists and combine the results
    return quick_sort(left_list) + middle_list + quick_sort(right_list)
```

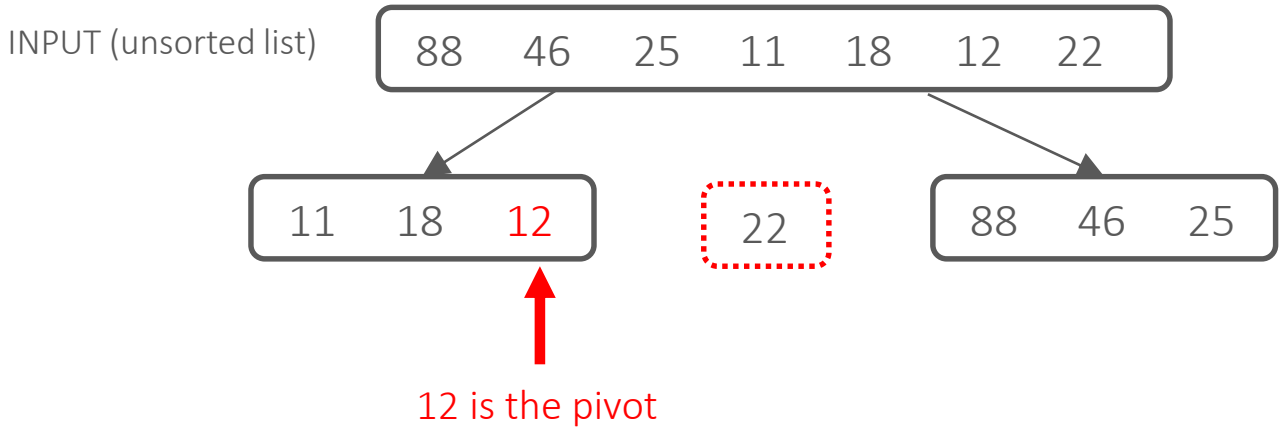
Example

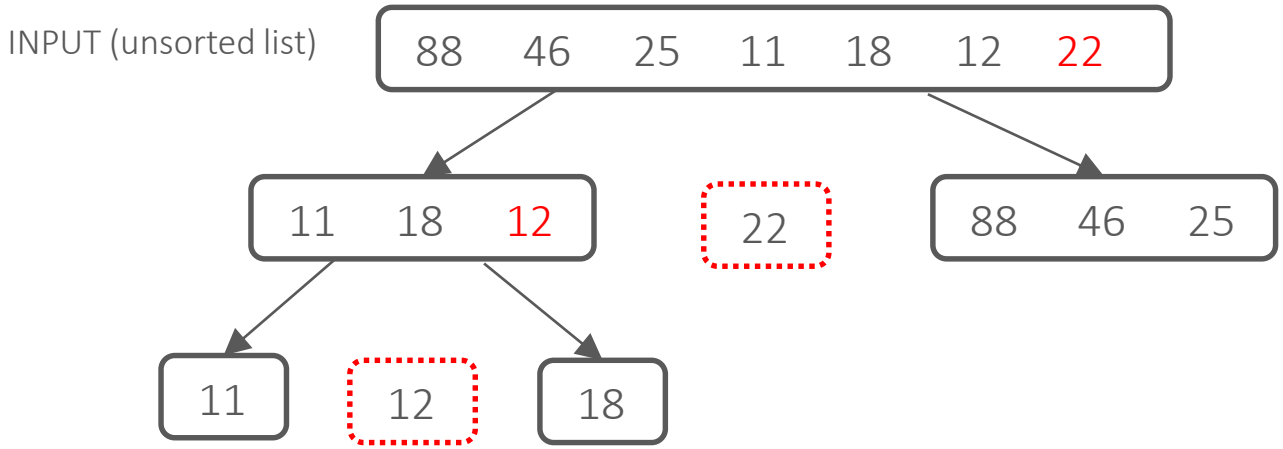

```
pivot = L[len(L)-1]
```

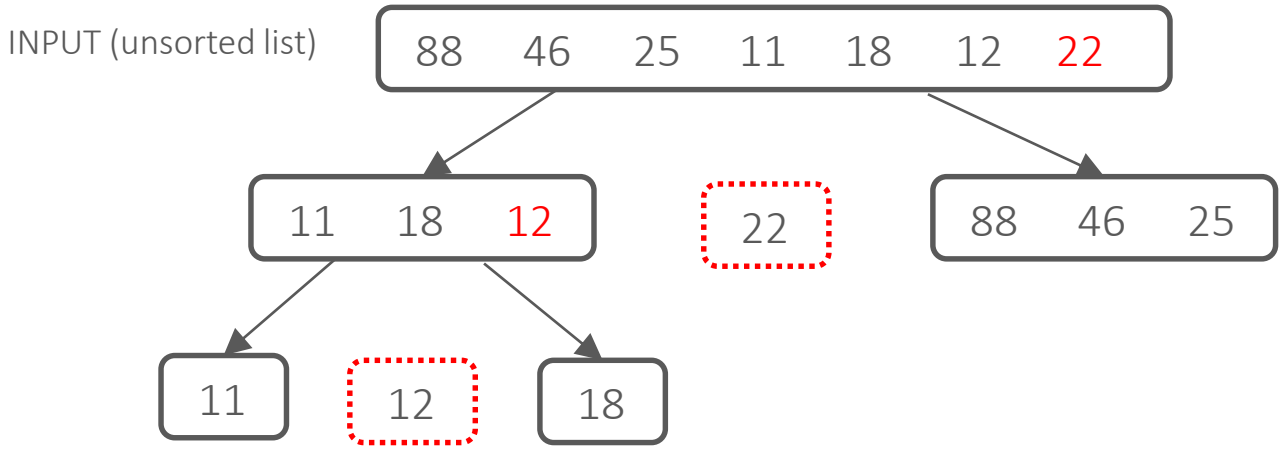

INPUT (unsorted list)
88 46 25 11 18 12 22
11 18 12
22
88 46 25
left_list
middle_list
right_list



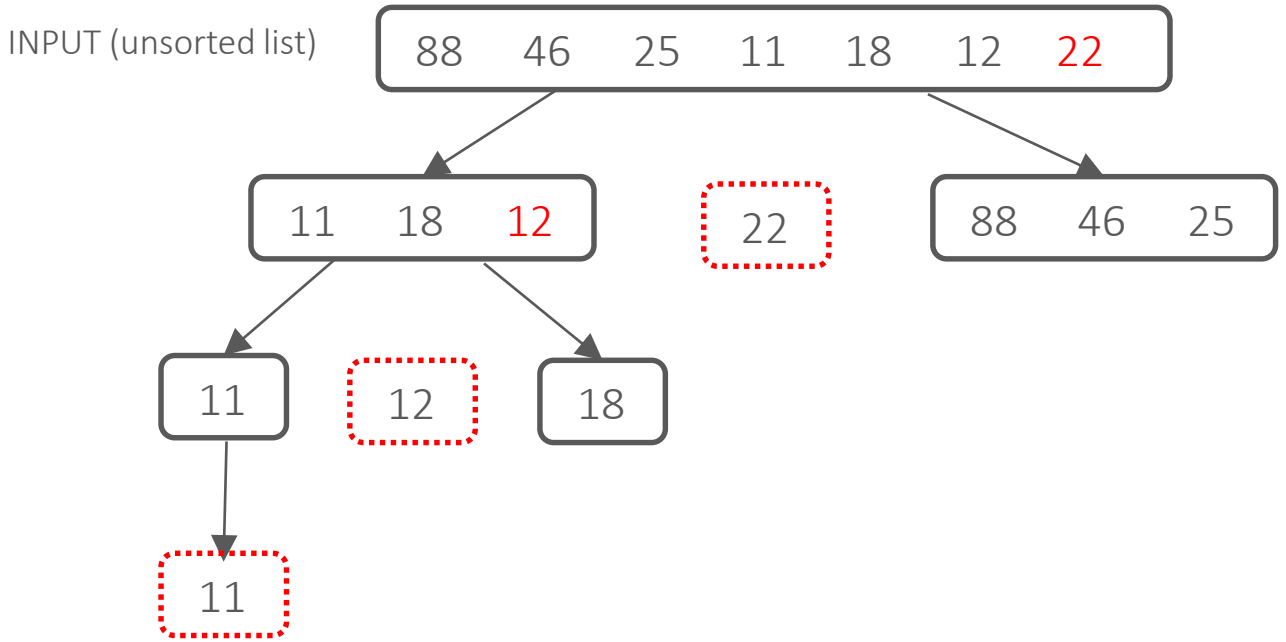
Now quicksort `left_list`


INPUT (unsorted list)
88 46 25 11 18 12 22
11 18 12
22
88 46 25
12 is the pivot


INPUT (unsorted list)
88 46 25 11 18 12 22
11 18 12
22
88 46 25
11
12
18

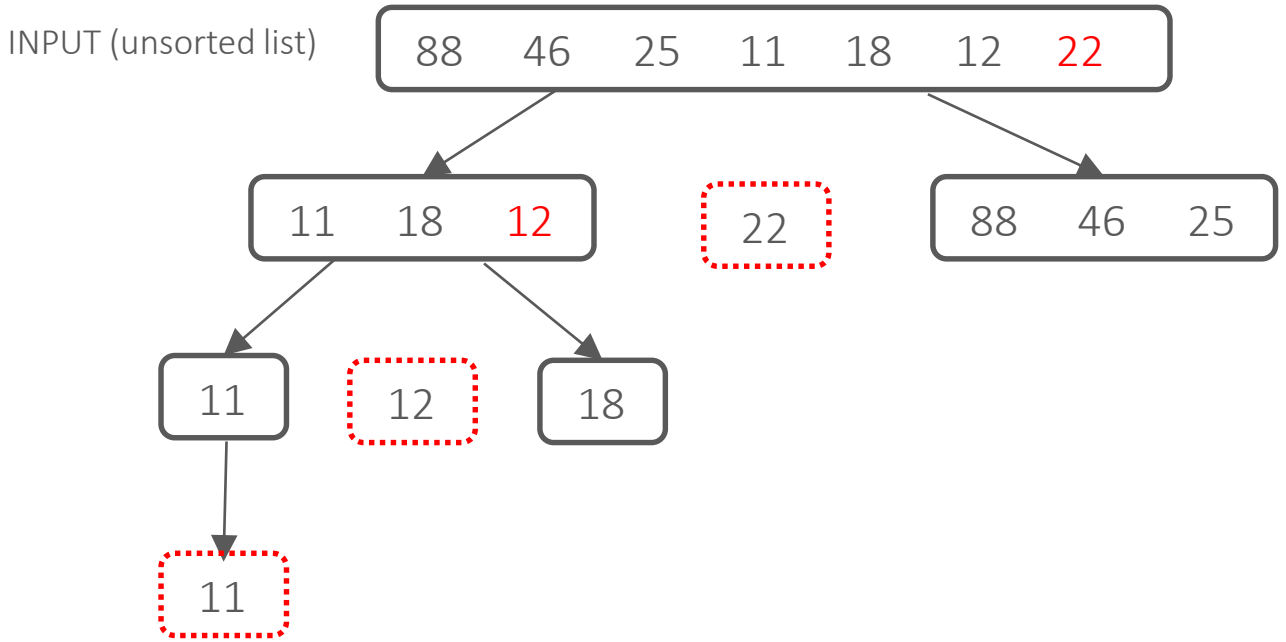


Now quicksort `left_list`

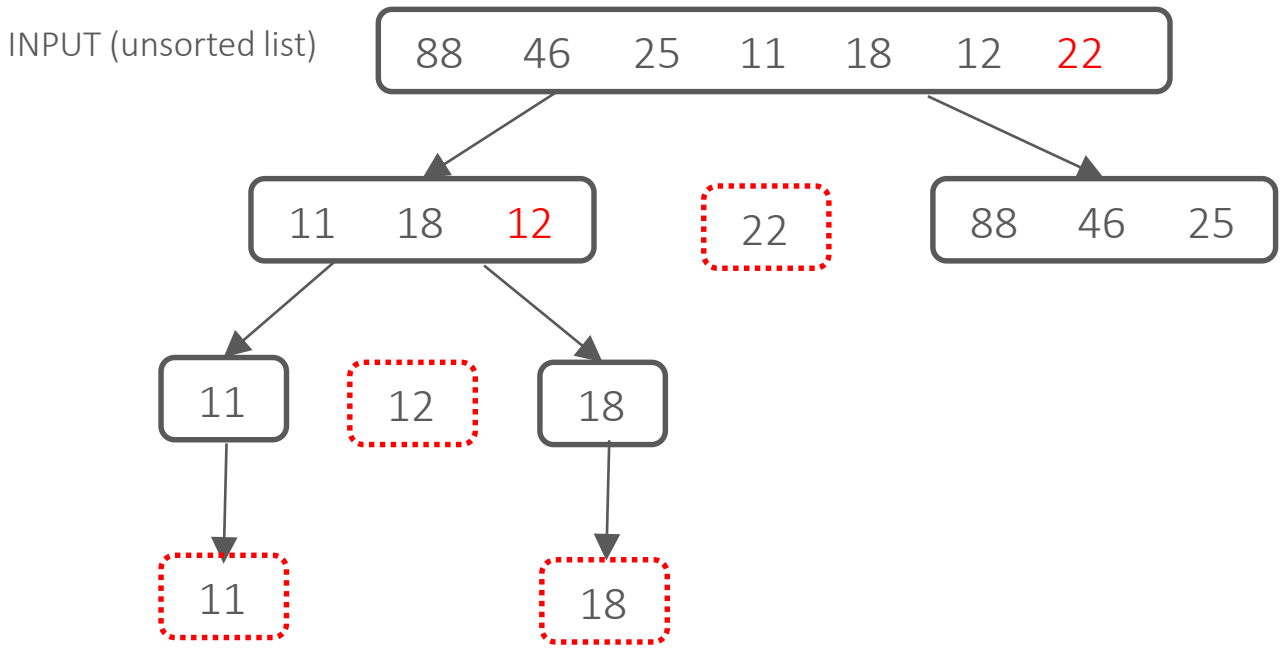


Now quicksort `left_list`

Base case – `len(L) <= 1` so return 11

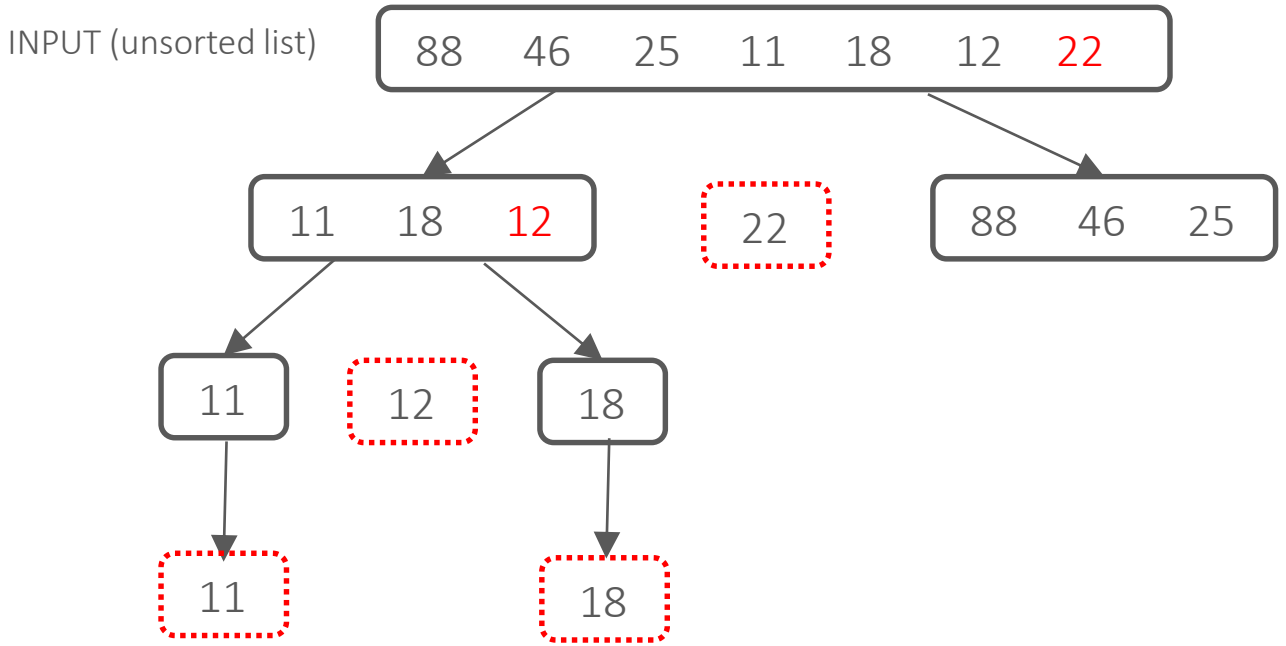


Now quicksort `right_list`

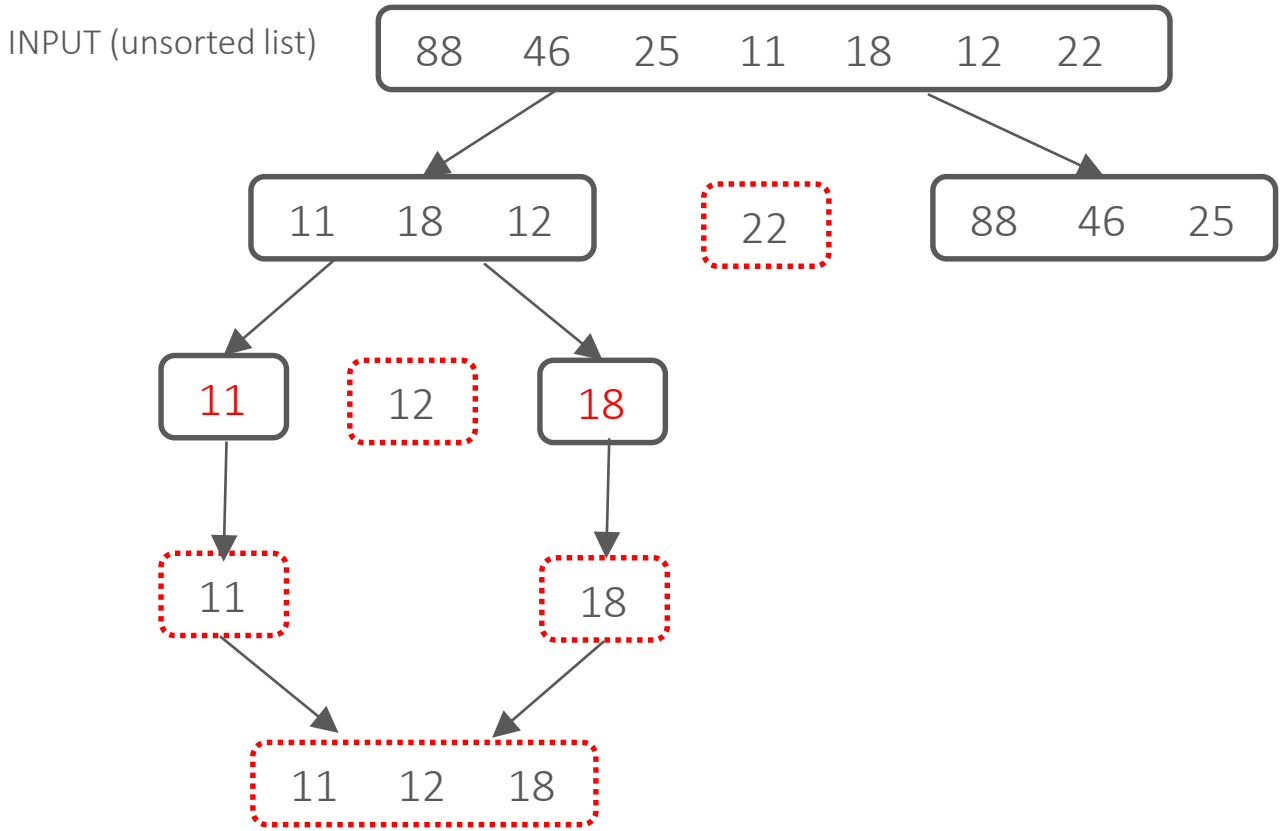


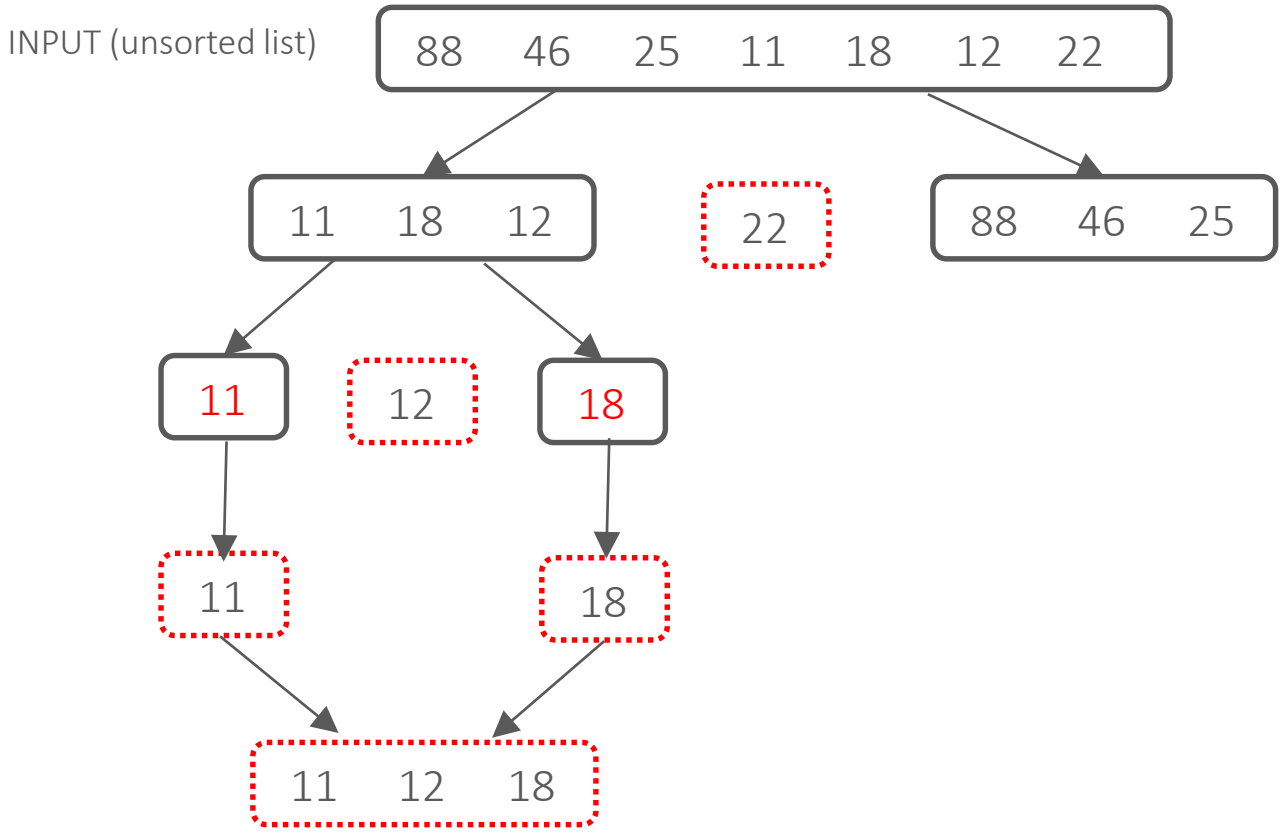
Now quicksort `right_list`

Base case – `len(L) <= 1` so return 18

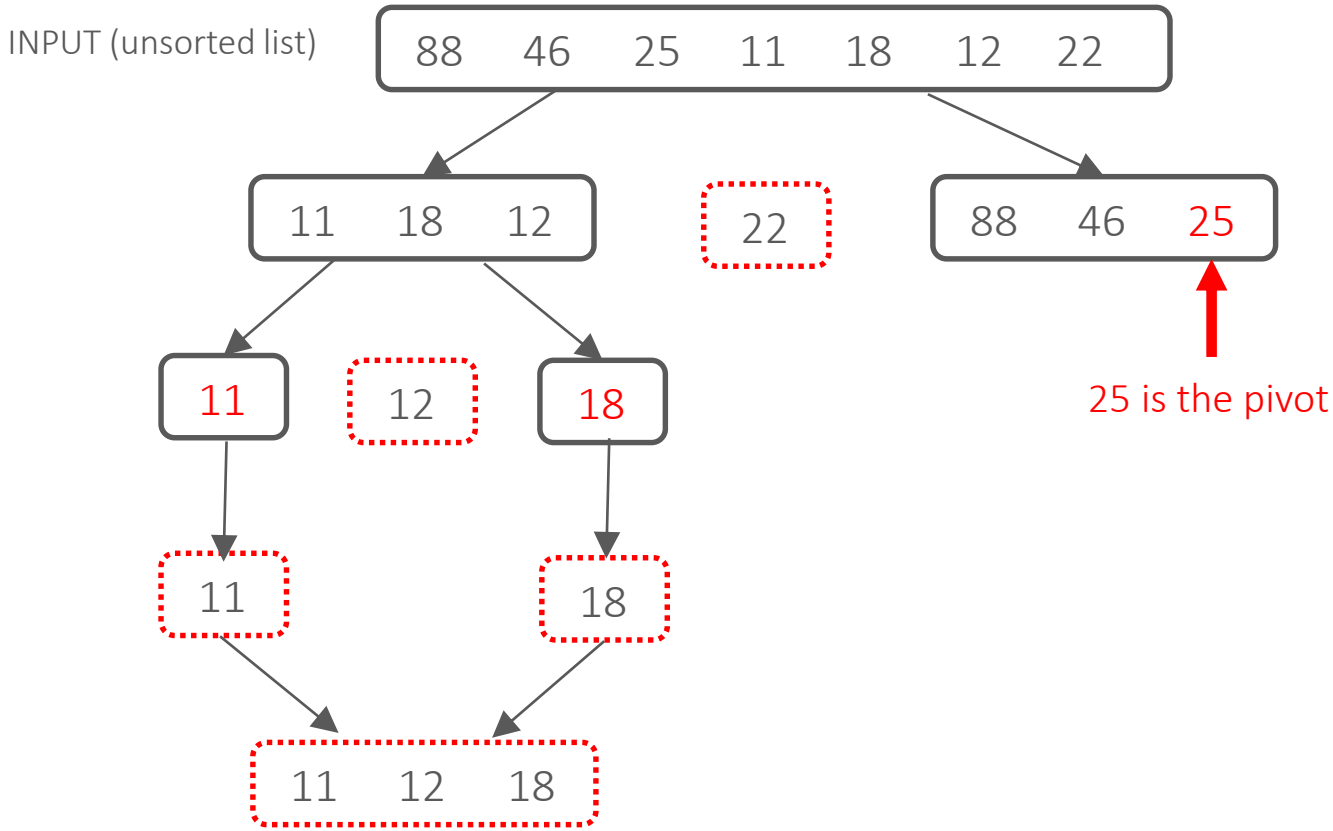


Result is **`left + middle + right`** so return 11 12 18

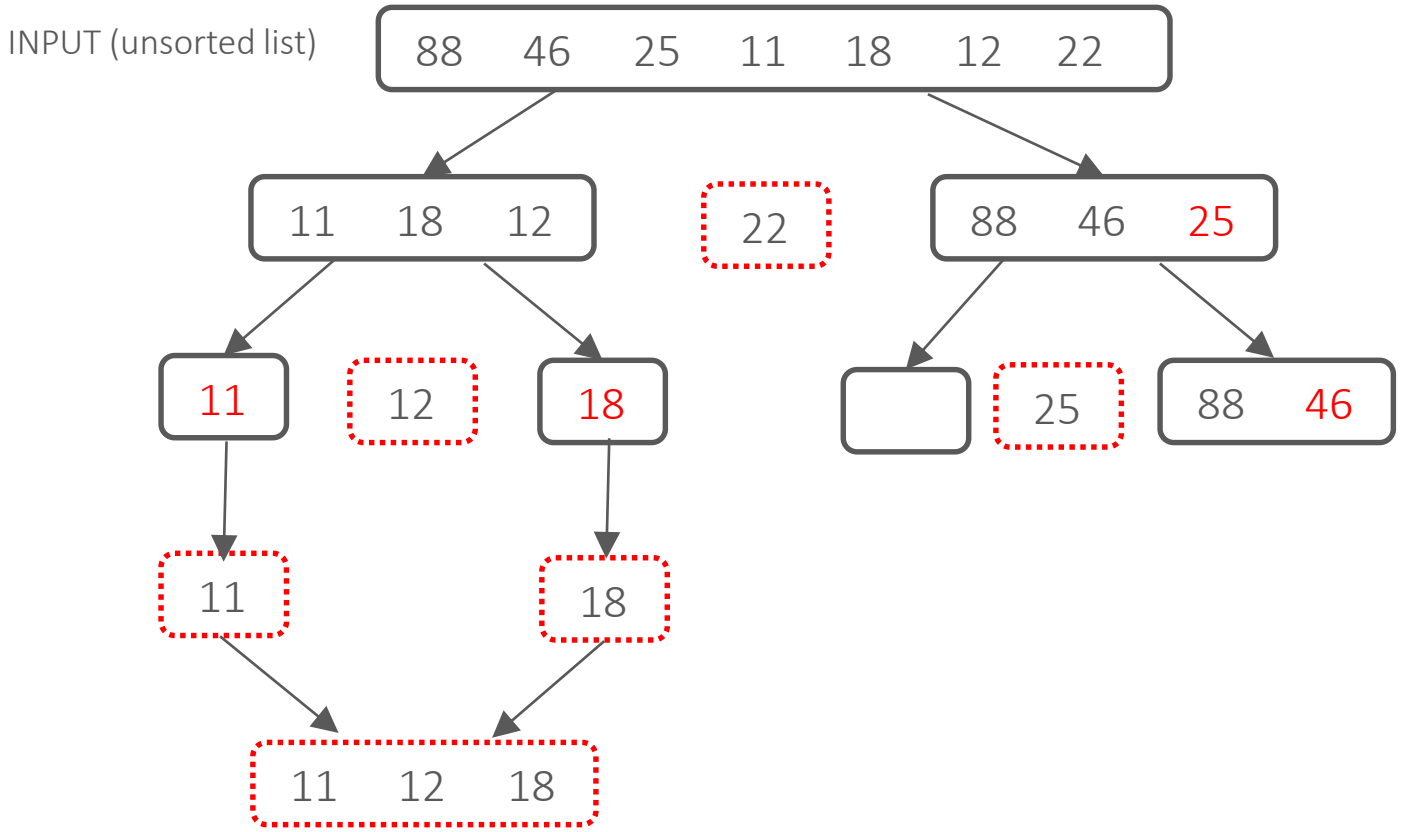

INPUT (unsorted list)
88 46 25 11 18 12 22
11 18 12
22
88 46 25
11
12
18
11
18
11 12 18



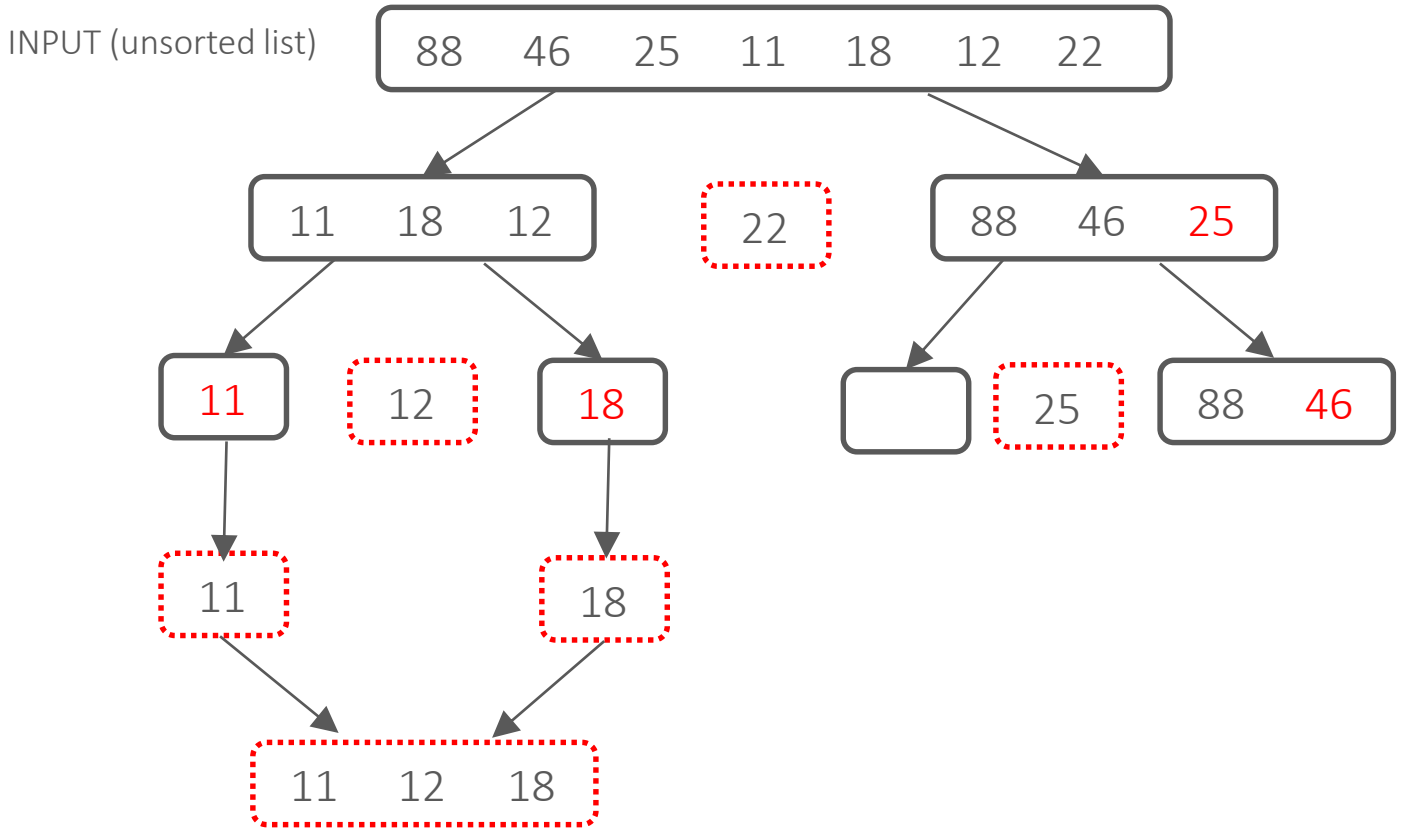
Now quicksort **`right_list`**



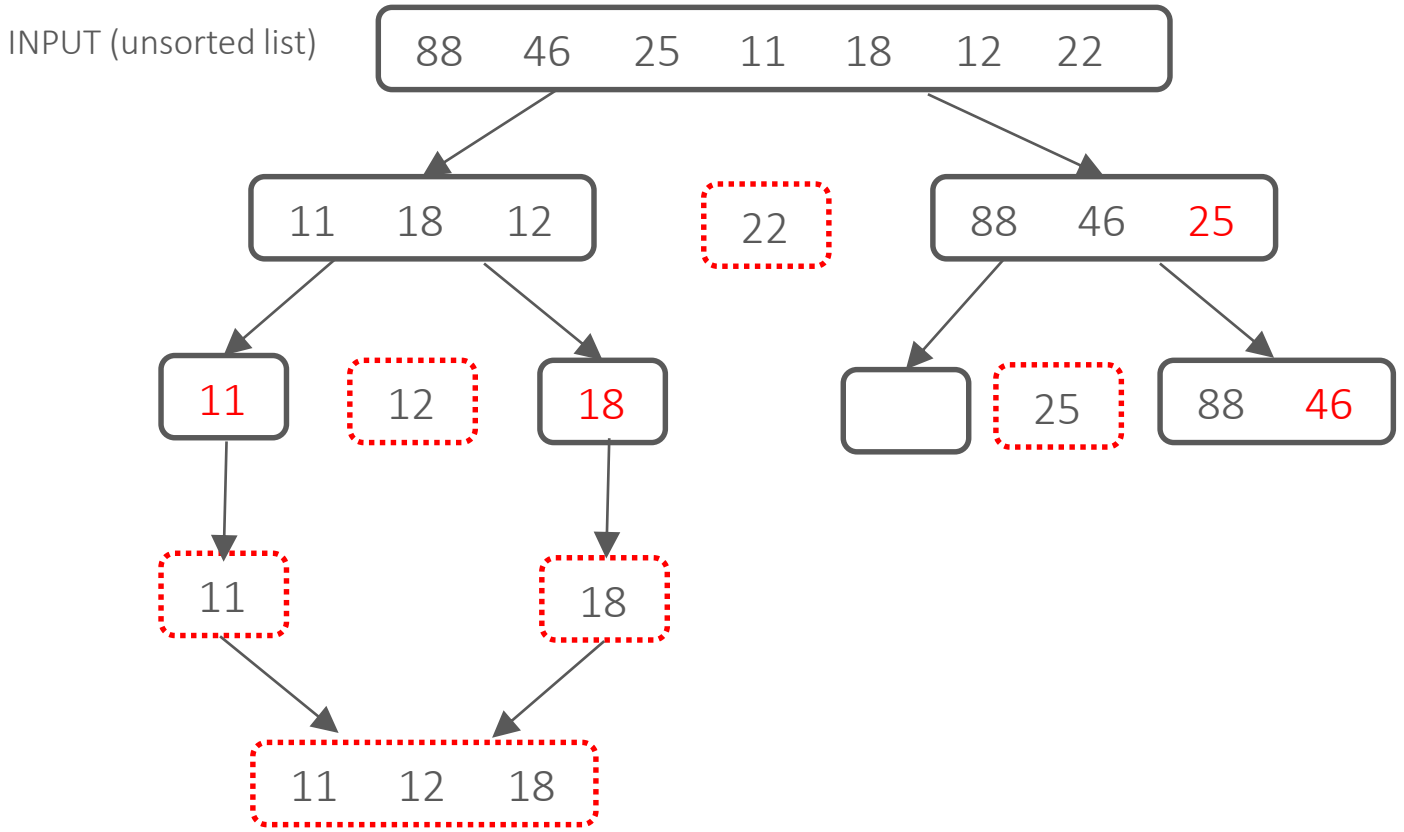
Now quicksort **`right_list`**



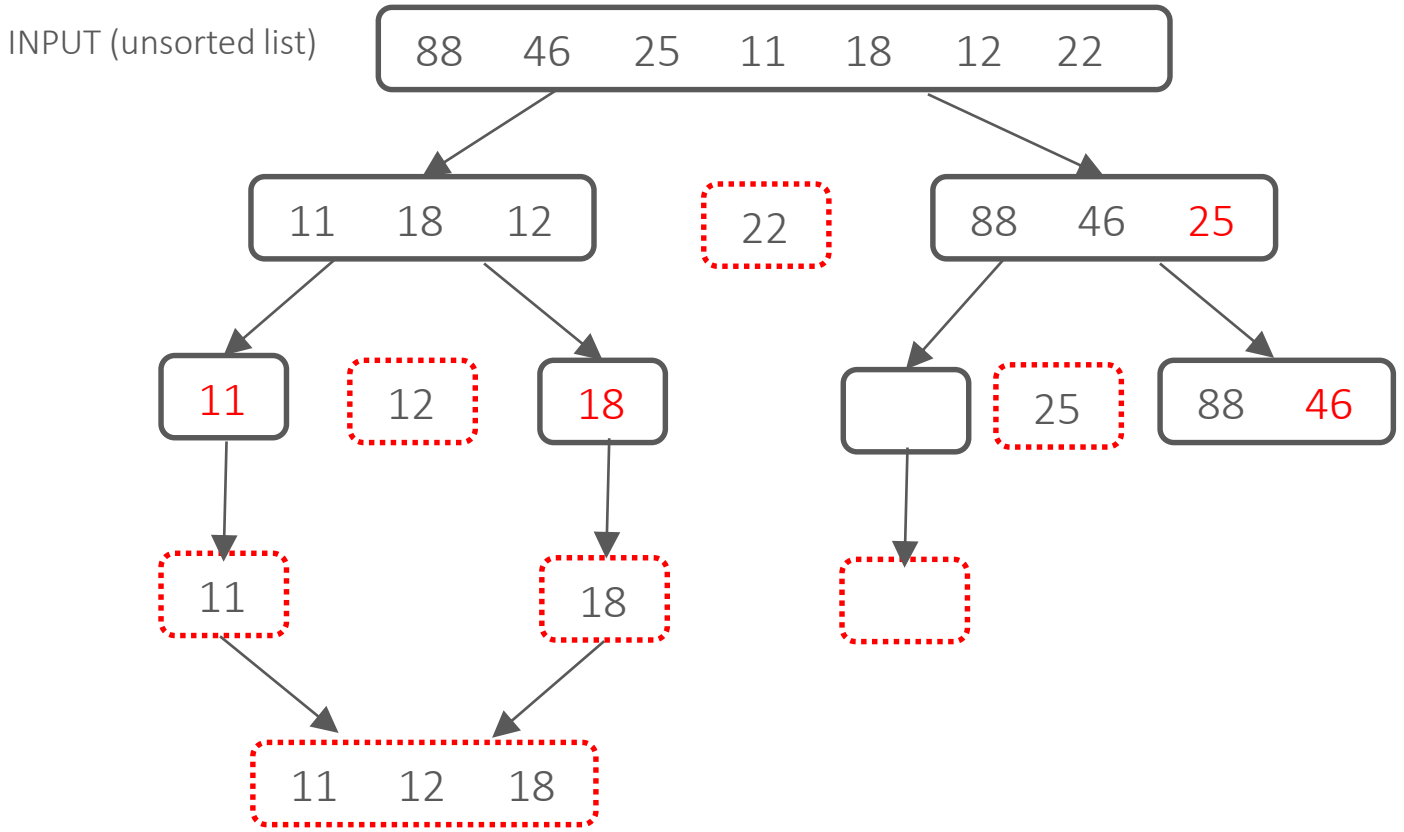
Partition around 25



Now quicksort **`left_list`**

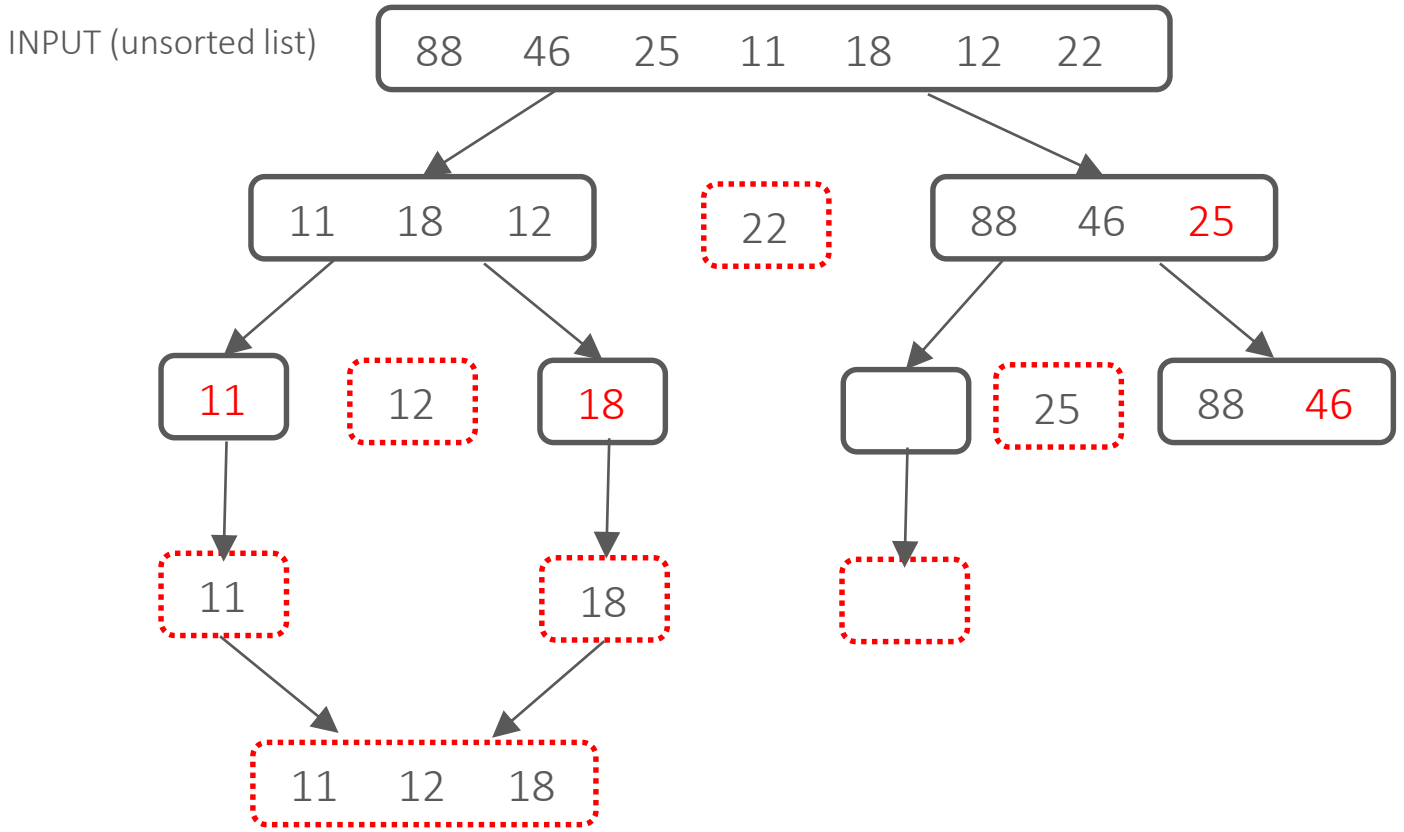


Now quicksort **`left_list`**

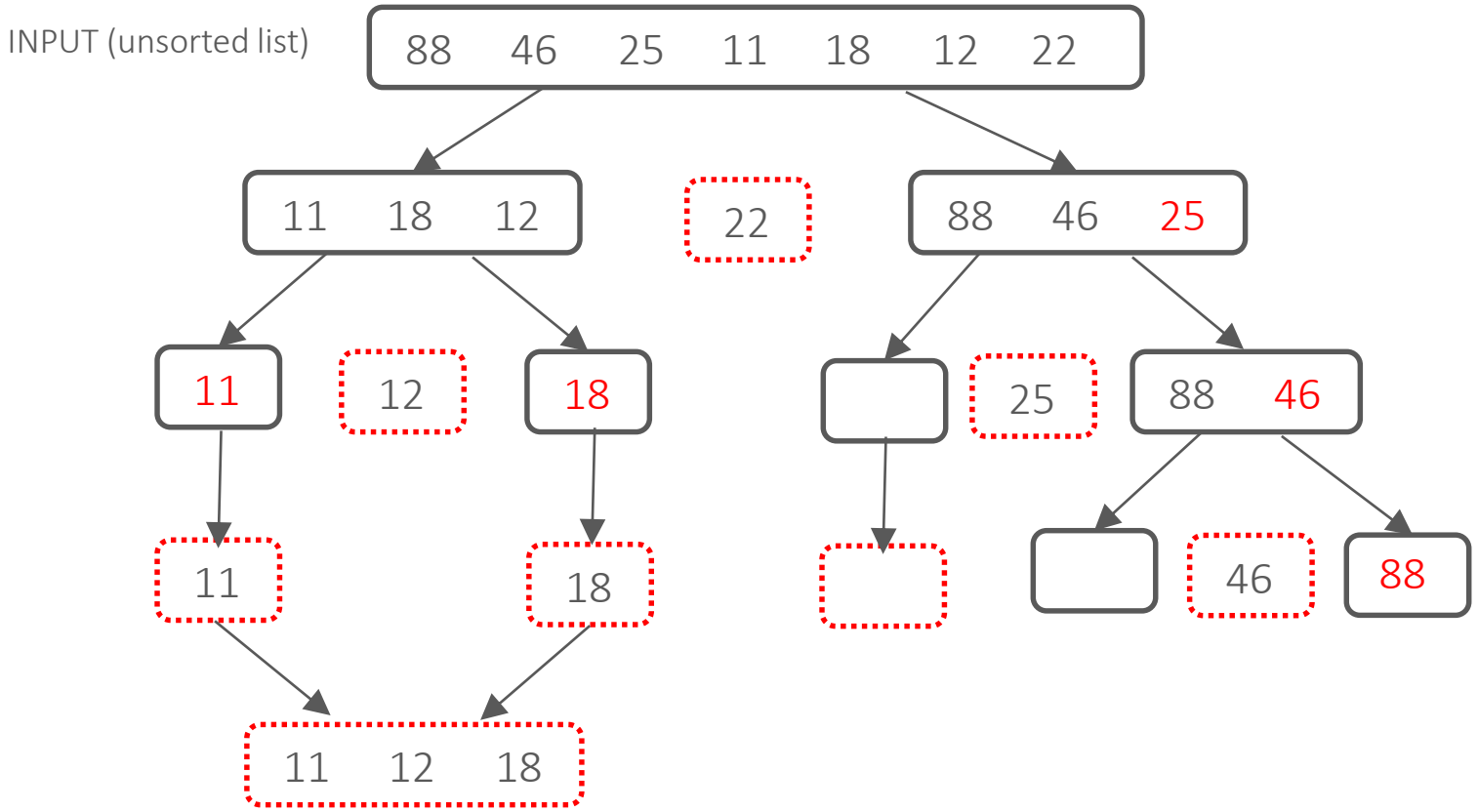


Now quicksort `left_list`

Base case – `len(L) <= 1` so return []

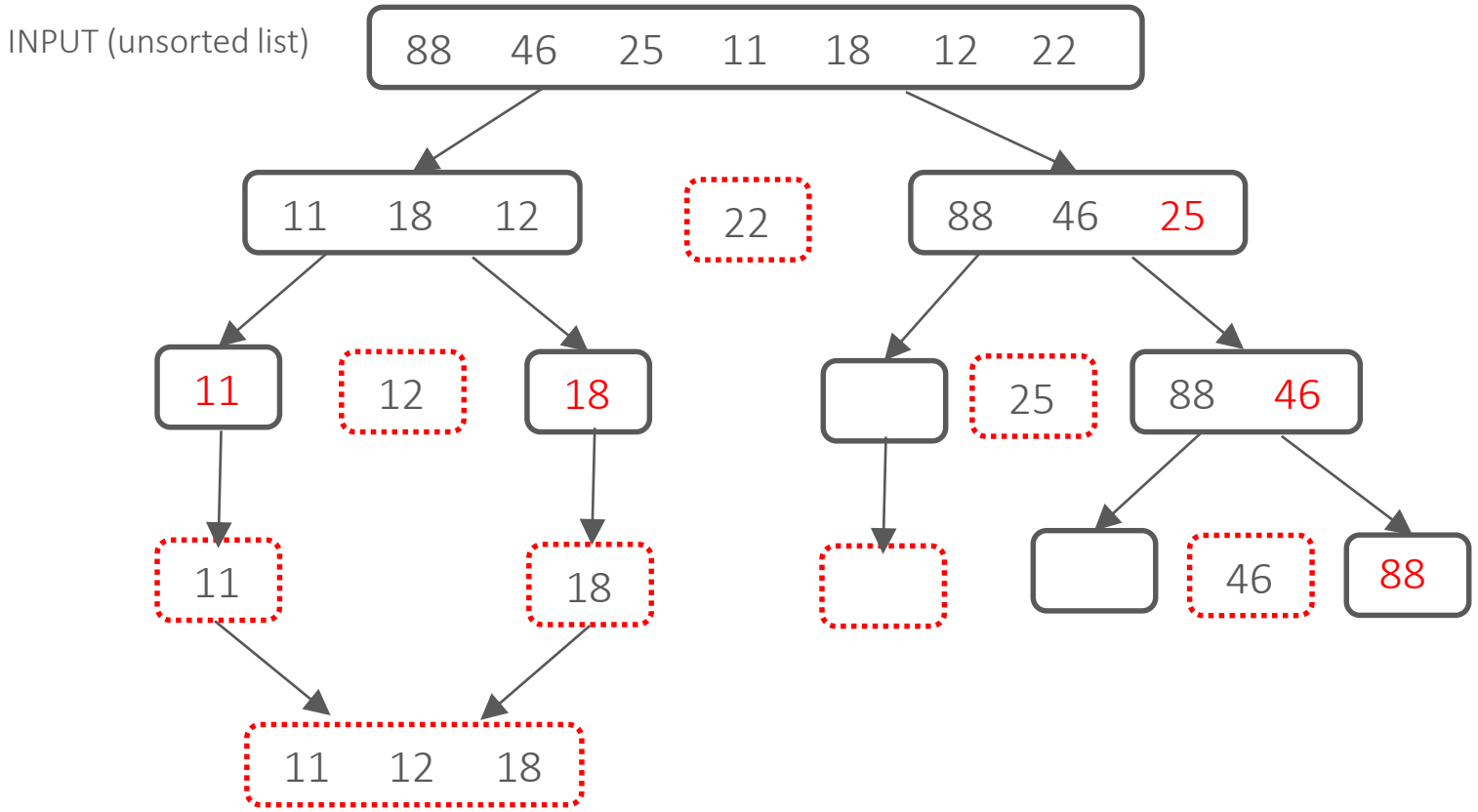


Now quicksort **`right_list`**



Now quicksort **`right_list`**

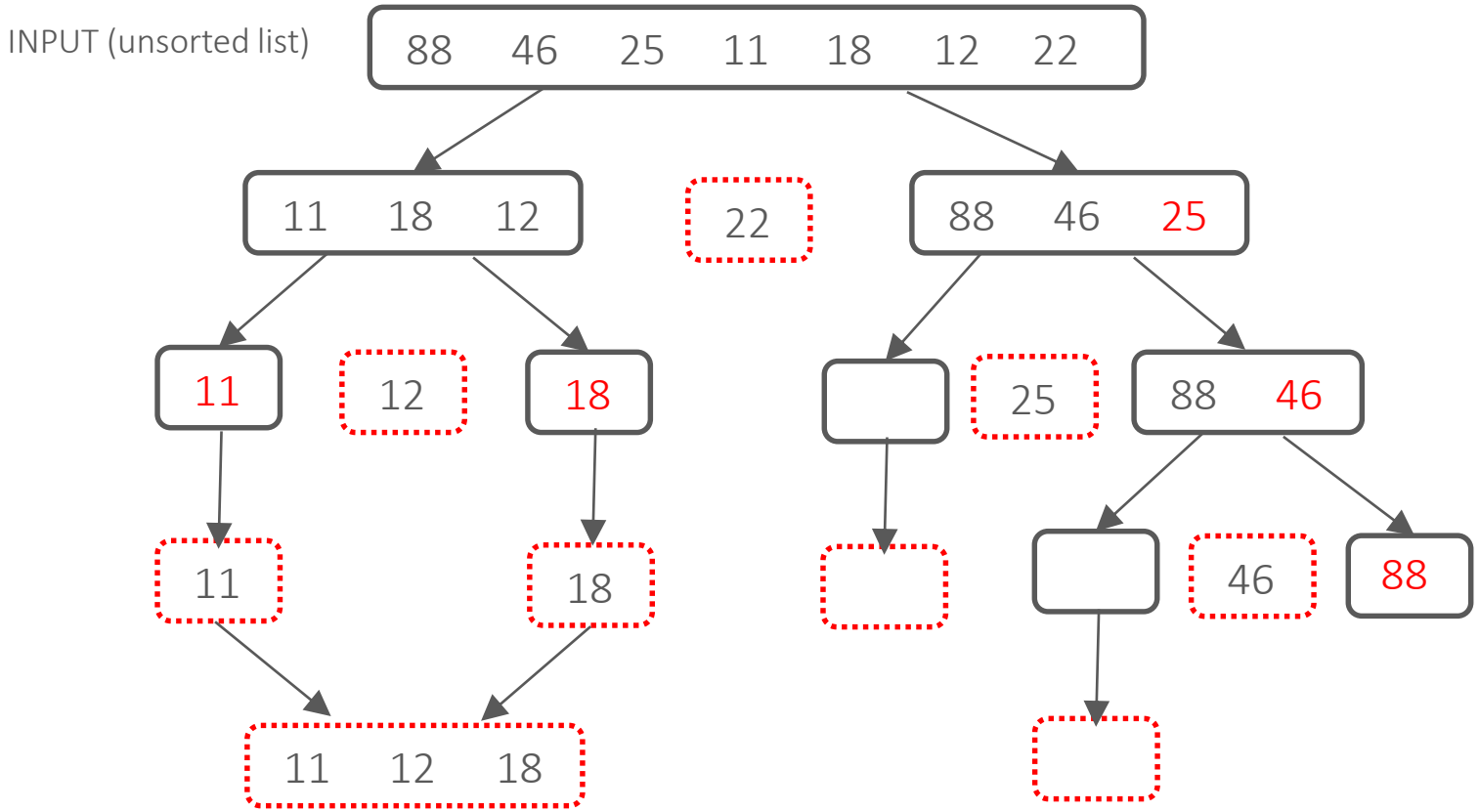
Partition around 46



Now quicksort **`right_list`**

Partition around 46

Now quicksort **`left_list`**

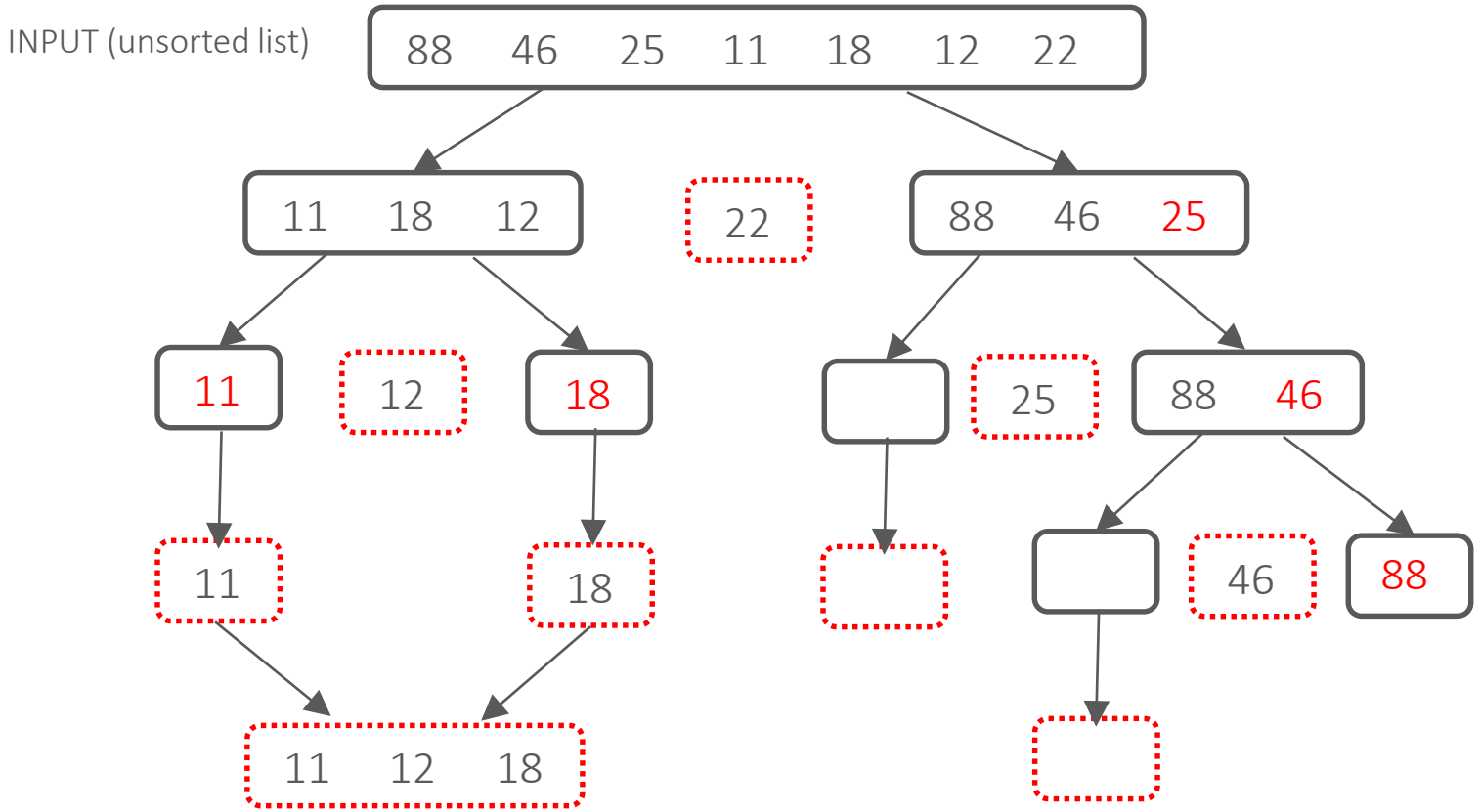


Now quicksort **`right_list`**

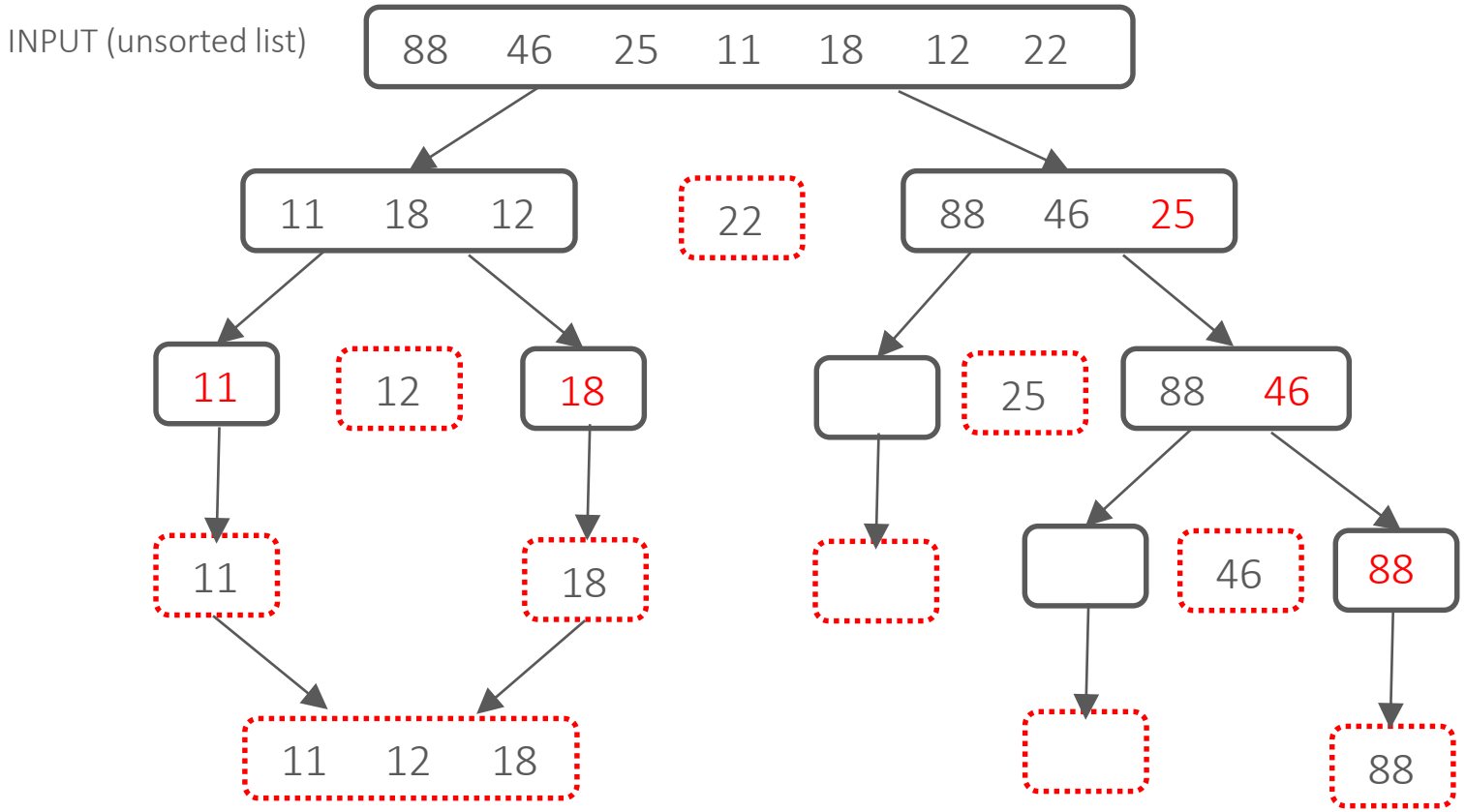
Partition around 46

Now quicksort **`left_list`**

Base case – **`len(L) <= 1`** so return []

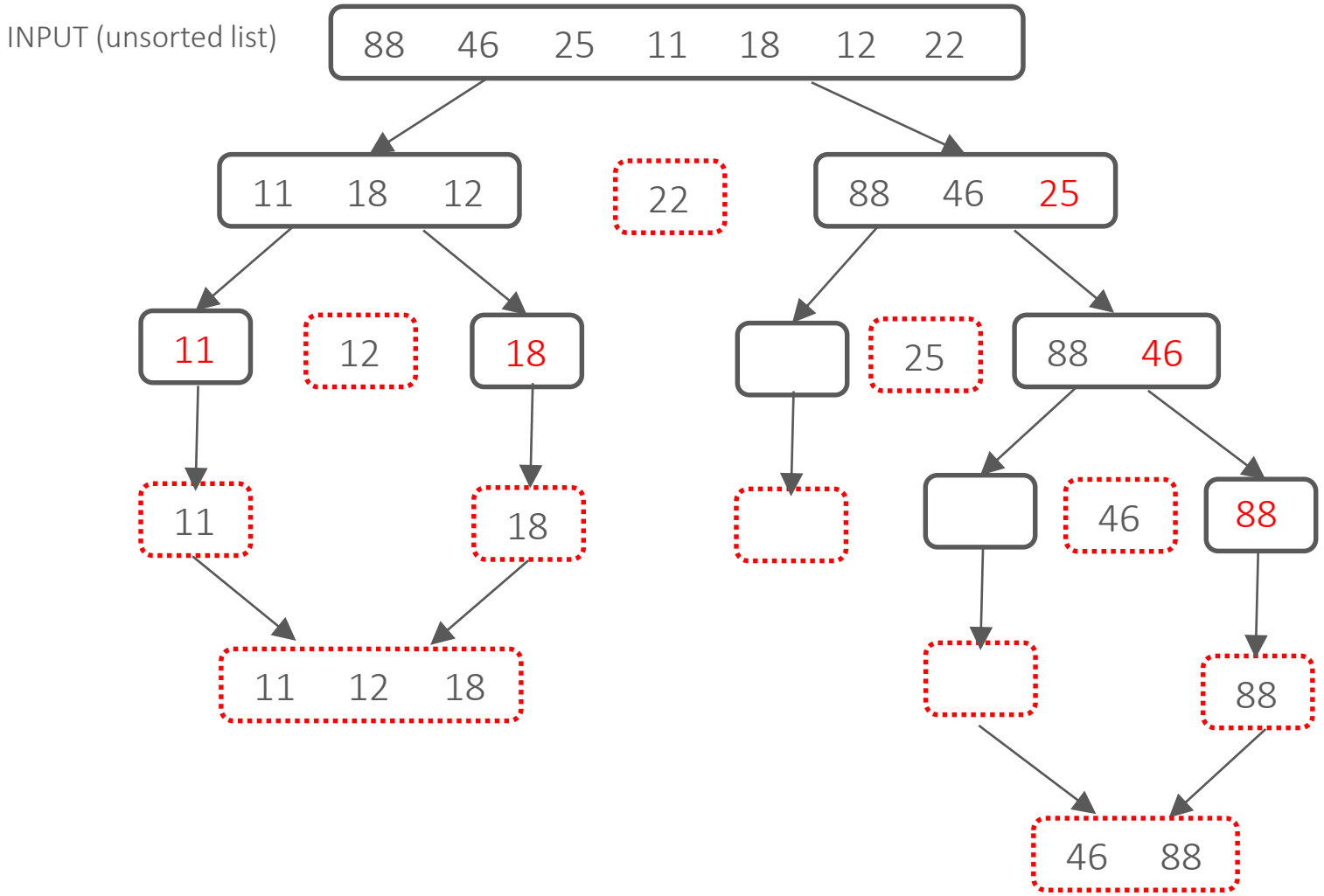


Now quicksort **`right_list`**

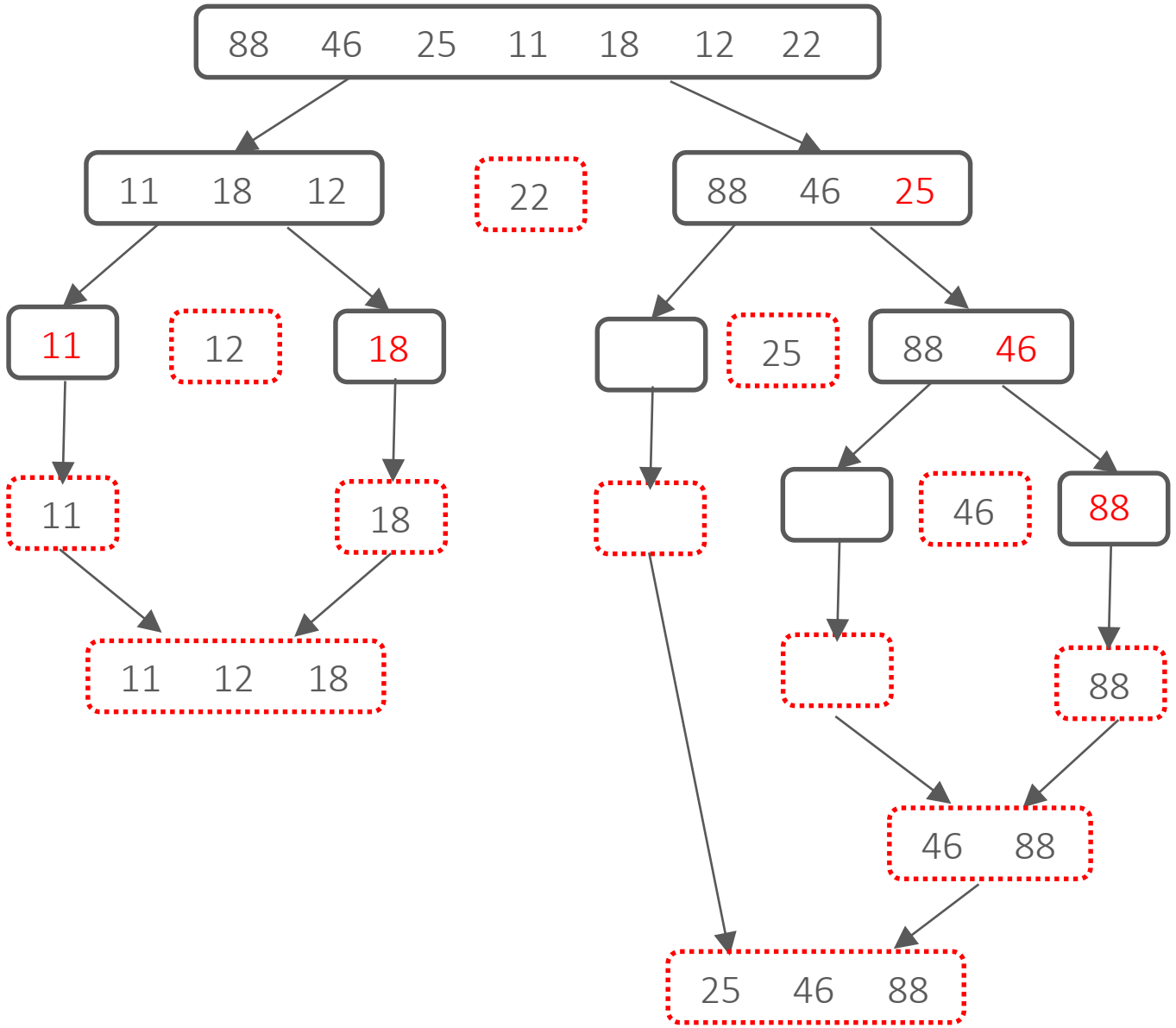


Now quicksort **`right_list`**

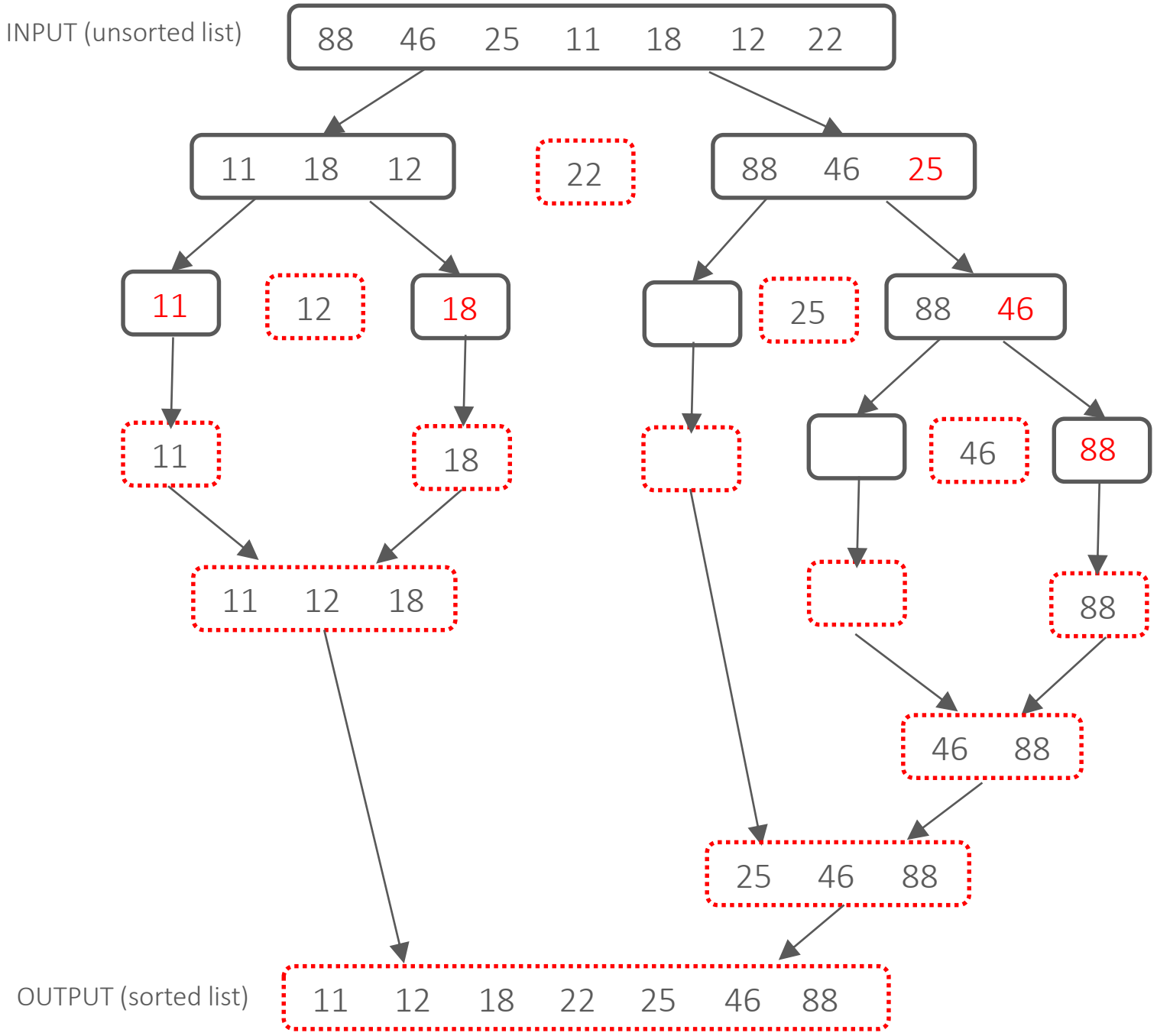
Base case – **`len(L) <= 1`** so return 88



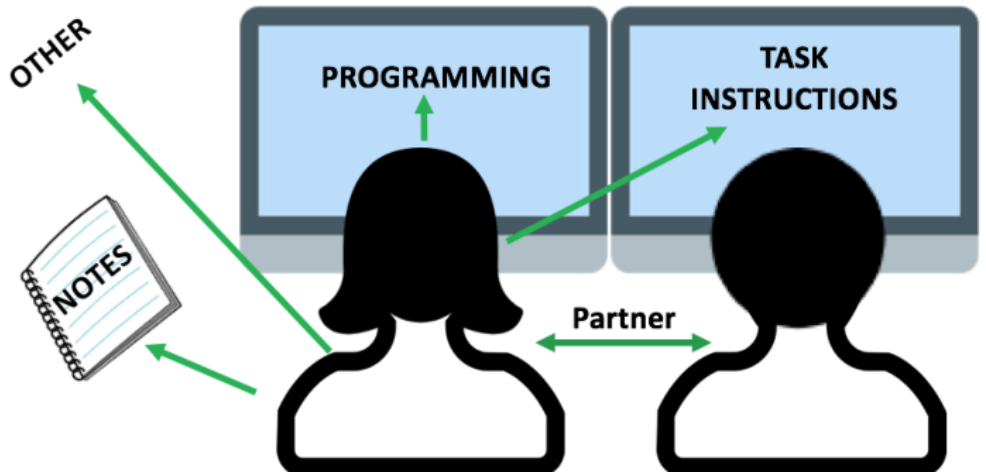
Result is `left + middle + right` so return 46 88



Result is **`left + middle + right`** so return 25 46 88

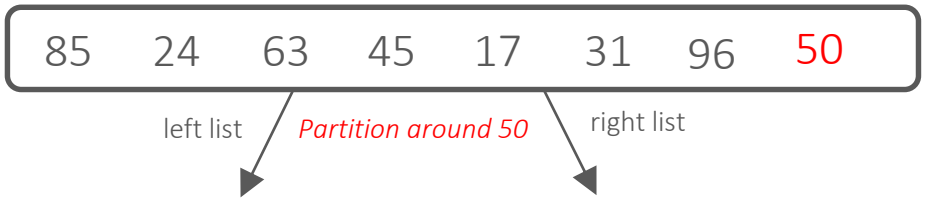

INPUT (unsorted list)
88 46 25 11 18 12 22
11 18 12
22
88 46 25
11
12
18
25
88 46
11
18
46
88
11 12 18
88
46 88
25 46 88
OUTPUT (sorted list)
11 12 18 22 25 46 88

Think Pair Share Activity



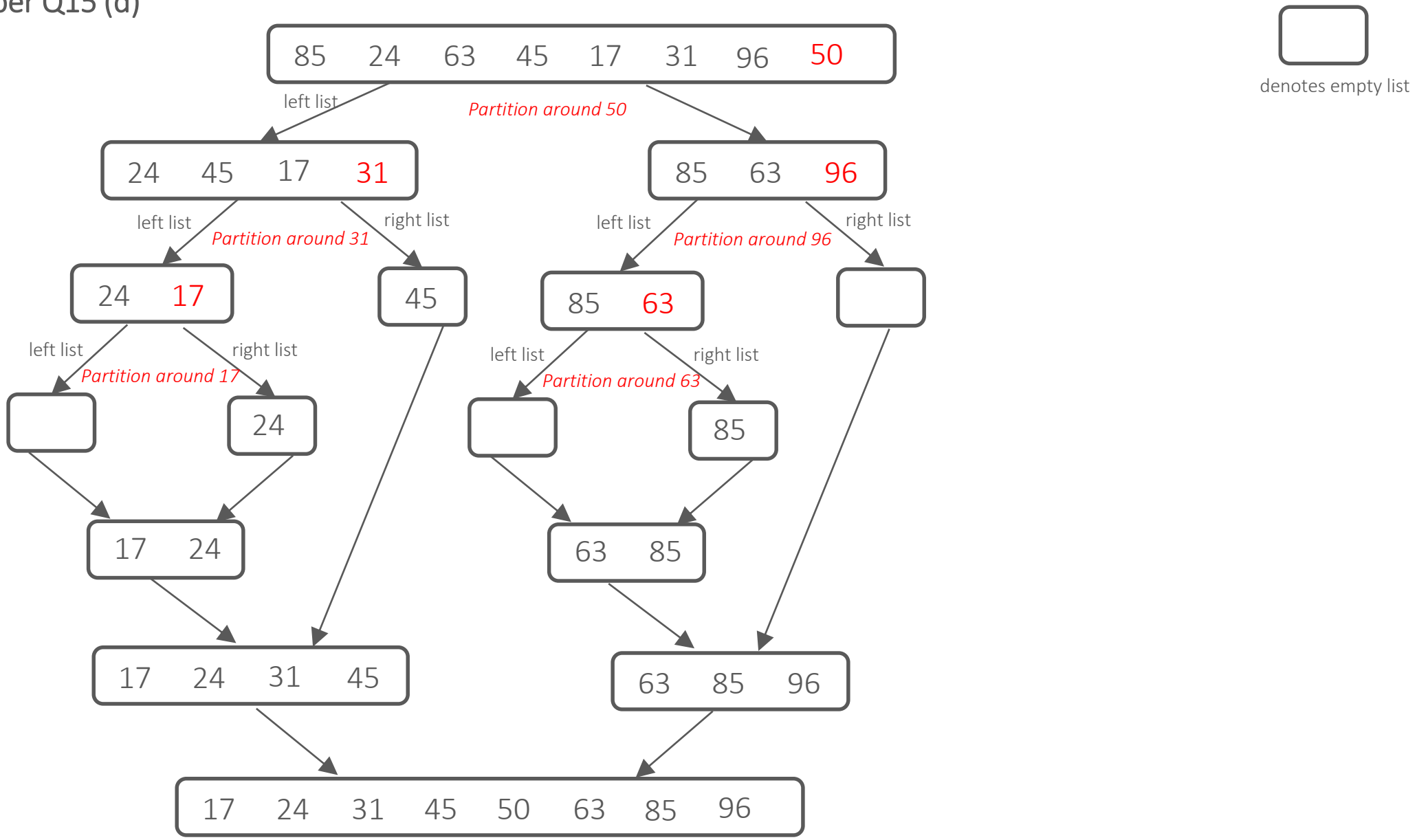
LCCS Sample Paper Q15 (d)

Perform a quicksort on the following:



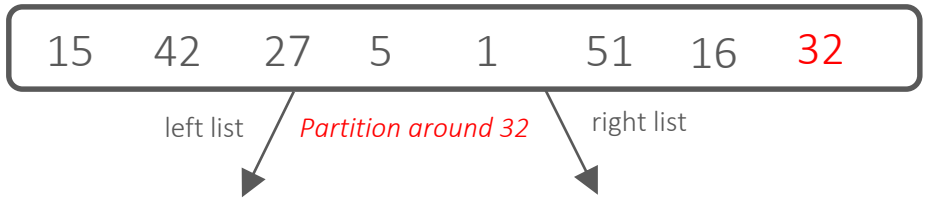
LCCS Sample Paper Q15 (d)

Sample Solution



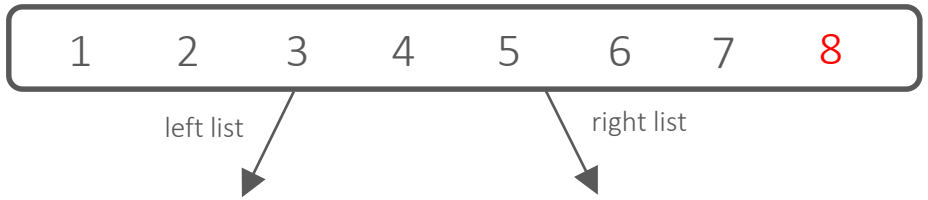
Exercise

Perform a quicksort on the following:



Exercise

Investigate why this scenario leads to the worst case performance for the quicksort



Algorithmic Complexity (An introduction)

PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Algorithmic Complexity

- Complexity is about analysing algorithms
- Choice between algorithms often comes down to efficiency
- We need to define objective measures that can be applied to each algorithm
  - Execution time?
  - Number of statements/instructions executed?
  - Number of times a fundamental operation is executed?

Time complexity

Time Complexity

- Time complexity gives the number of operations an algorithm performs when processing an imput of size n.
- An algorithm can have different time complexity values for the same n
- We consider 3 cases:
    1. Best Case: minimum number of operations required for a given input
    2. Worst Case: maximum number of operations required for a given input
    3. Average Case: average number of operations required for a given input

Q. Why are computer scientists mostly interested in worst case?
A. Worst case analysis lets us make hard guarantees regarding upper bounds on the amount of time it will take a critical process/task to complete.

Big-O

- Big O is a notation used in Computer Science to describe the worst case running time (or space requirements) of an algorithm in terms of the size of its input usually denoted by n.
- Big-O notation provides a way to talk about the kind of relationship between the size of the problem and the program running time.

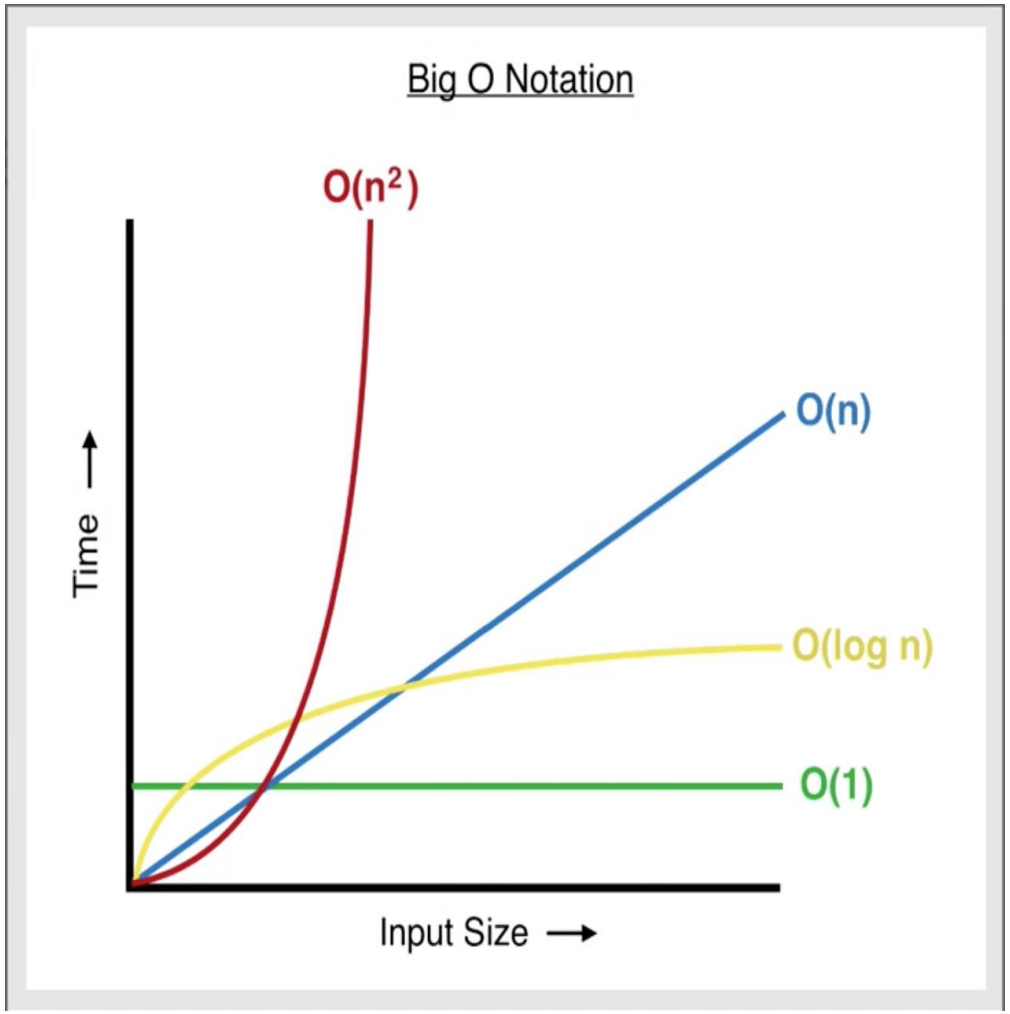
Complexity allows us to classify algorithms (as 'good', 'fair' or 'poor' in terms of performance) and therefore provides a basis to compare algorithms

Big-O classifications

- $O(1)$ Constant Complexity
- $O(n)$ Linear Complexity
- $O(n^2)$ Quadratic Complexity
- $O(\log_2 n)$ Logarithmic Complexity
- $O(n \log_2 n)$ Linearithmic Complexity

Also,

- $O(2^n)$ Exponential Complexity
- $O(n!)$ Factorial Complexity



Summary: Algorithmic Time Complexity

Always consider the running time and the expected format of the input list before choosing a search or sorting algorithm for a particular problem.

	Best Case	Average Case	Worst Case
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(log_2 n)$	$O(log_2 n)$
Simple (selection) Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n\ log_2 n)$	$O(n\ log_2 n)$	$O(n^2)$

PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Searching Algorithms: Linear Search

Linear Search

Question: does the list below contain the number 14?

If **14 is found**, what should the result to be?
True?
The position of 14?

If **14 is not found**, what should the result be?
False?
-1?
The length of the list?

Input:

A list L and a *target value* of 14

© jeng

Input:

A list L and a *target value* of 14

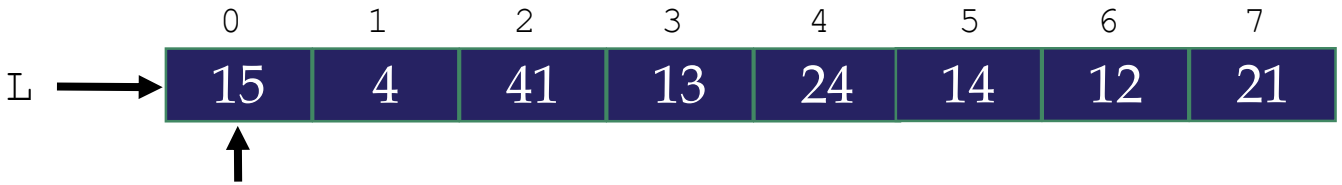
Required Output:

If the *target value* is found in L, its index

If the *target value* is not found in L, -1 is returned

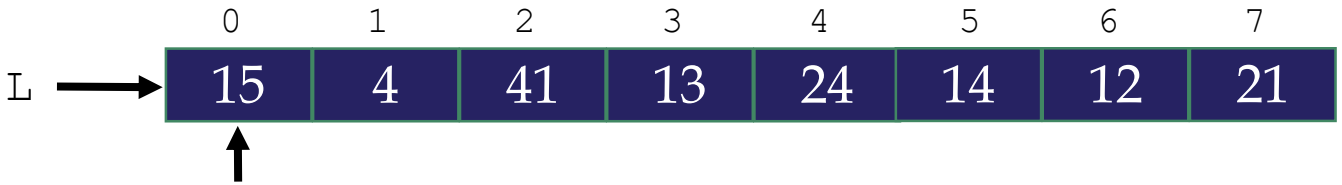
© jeng

Start at the first element and ask is L[0] == the *target value*?



© jeng

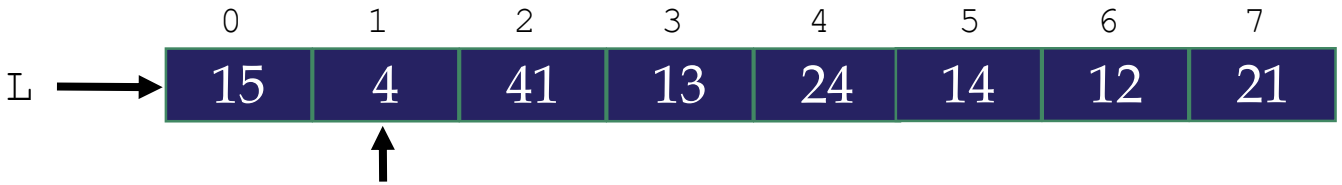
Start at the first element and ask is L[0] == the *target value*?



`L[0] == 14?` **NO!**

© jeng

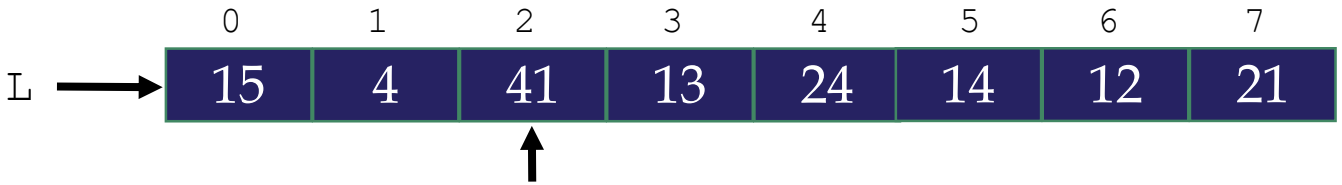
Move to the next element and ask is L[1] == the *target value*?



`L[1] == 14?` **NO!**

© jeng

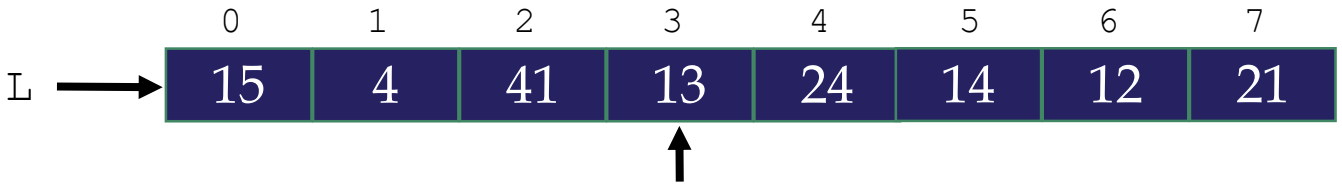
Move to the next element and ask is L[2] == the *target value*?



`L[2] == 14?` **NO!**

© jeng

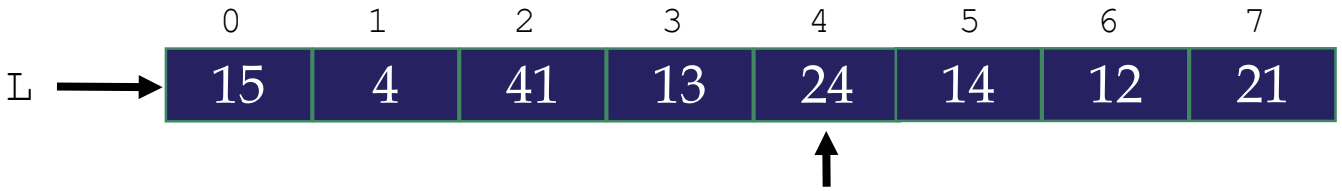
Move to the next element and ask is L[3] == the *target value*?



`L[3] == 14?` **NO!**

© jeng

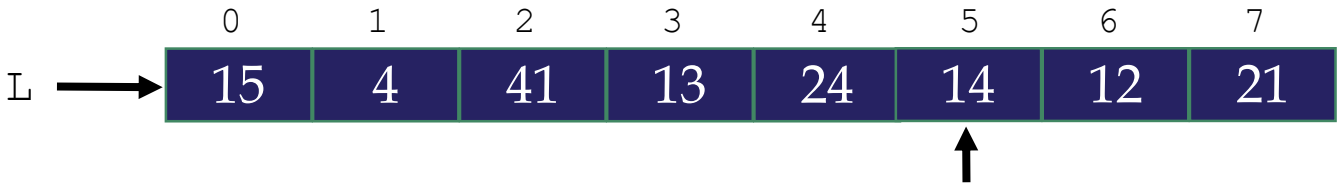
Move to the next element and ask is L[4] == the *target value*?



`L[4] == 14?` **NO!**

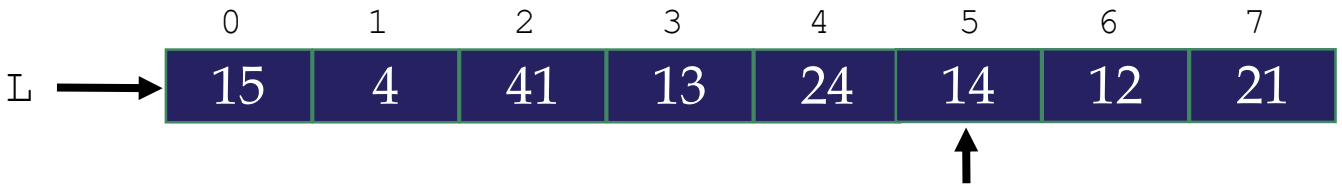
© jeng

Move to the next element and ask is L[5] == the *target value*?



`L[5] == 14?` **YES!**

© jeng



`L[5] == 14?` **YES!**

We have found the *target value* at index 5

The result of the search is 5

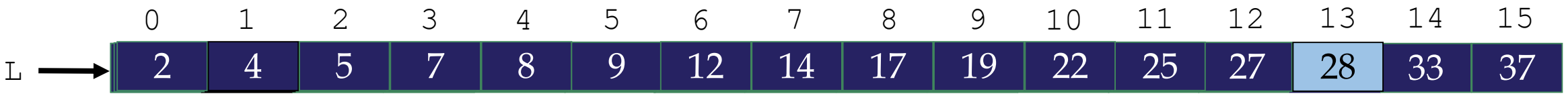
© jeng

PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Searching Algorithms: Binary Search

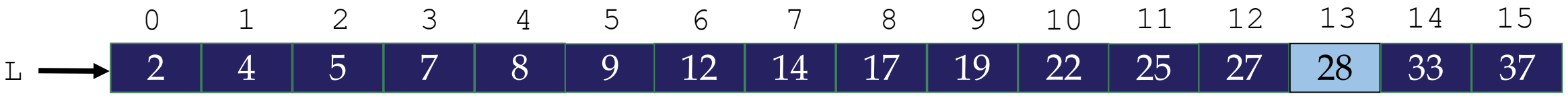
Input:

A list L and a target value of 28



Input:

A list L and a *target value* of 28

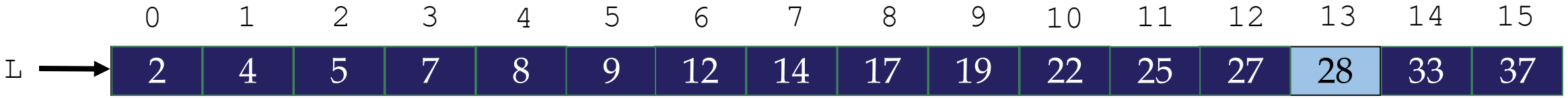


Required Output:

If the *target value* is found in L, its index

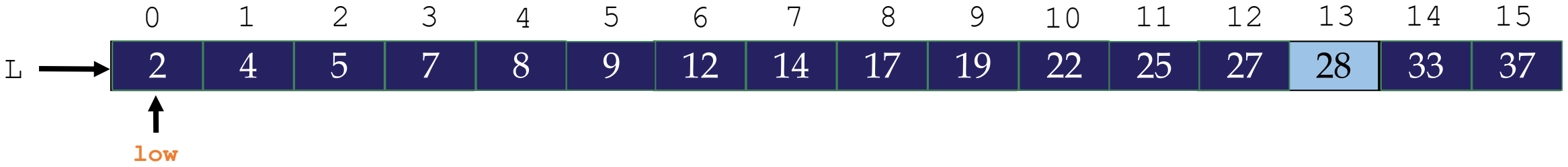
If the *target value* is not found in L, we return -1

Binary Search Algorithm



Pseudo-code: (target value is 28)

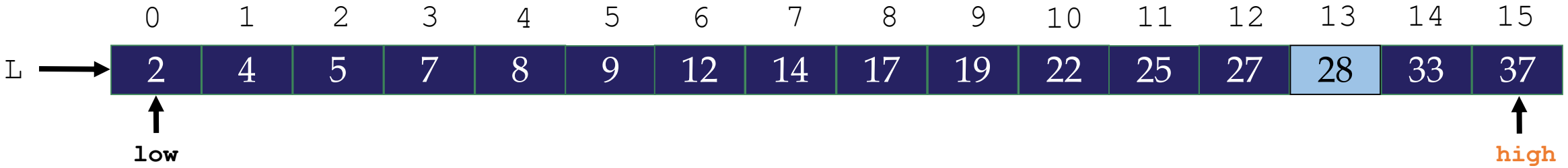
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0

Binary Search Algorithm

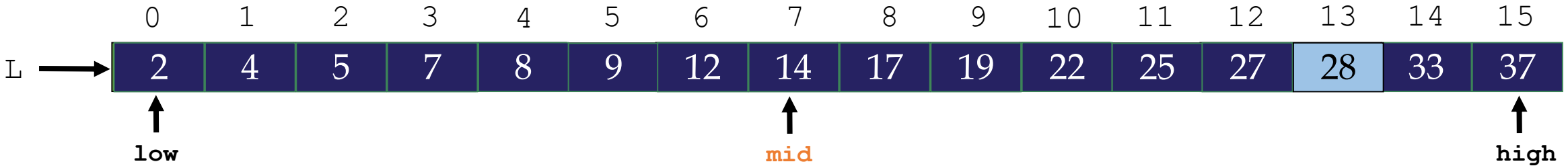


Pseudo-code: (target value is 28)

1. Set low = 0

2. Set high = length of list - 1

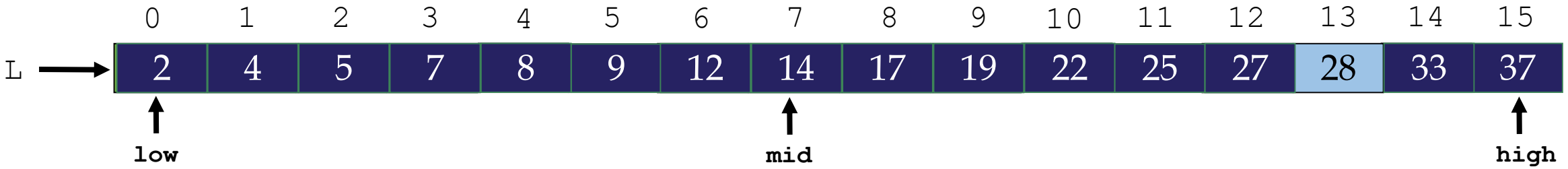
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. **Set low = 0**
2. **Set high = length of list - 1**
3. **Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer**

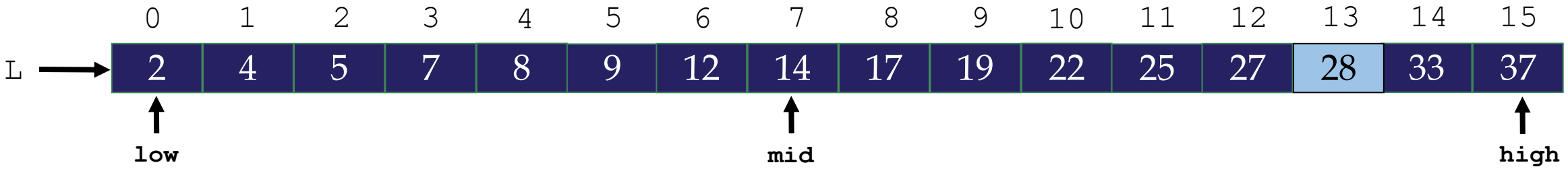
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. **Set low = 0**
2. **Set high = length of list - 1**
3. **Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer**
4. **If the value at the mid position is the same as the target value**
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1

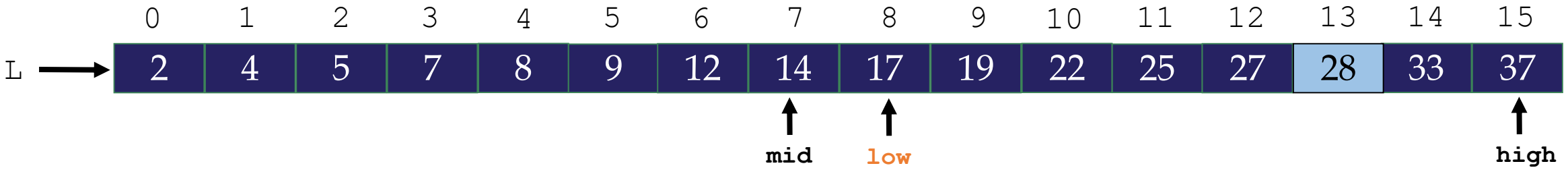
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1

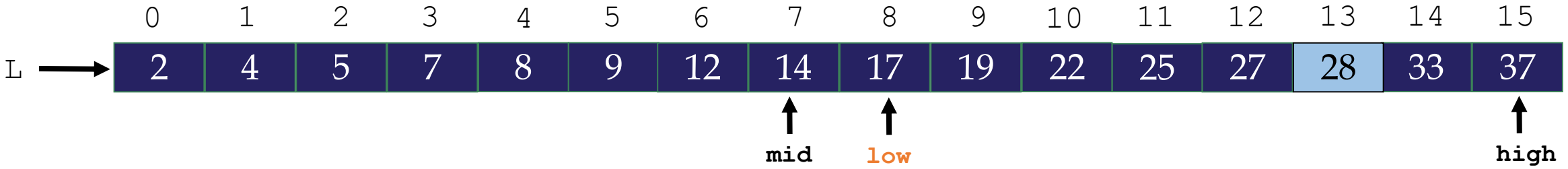
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1

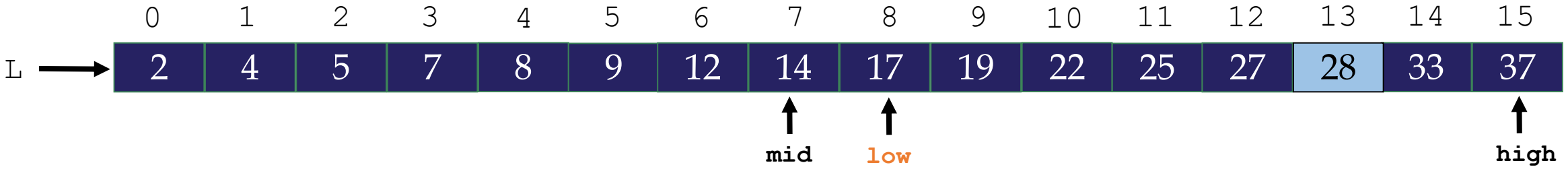
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

Binary Search Algorithm

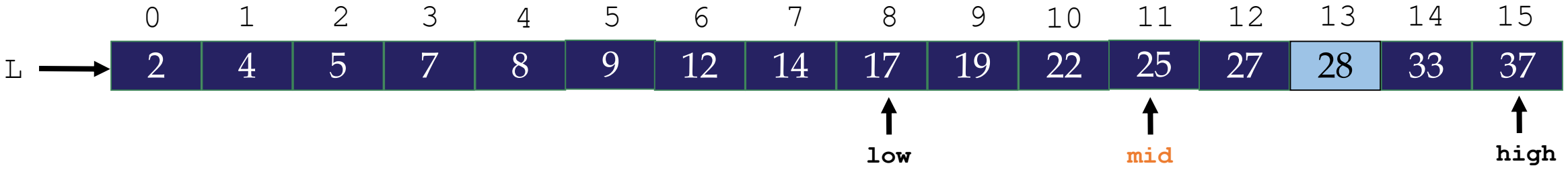


Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

In Python this means, `while low <= high:`

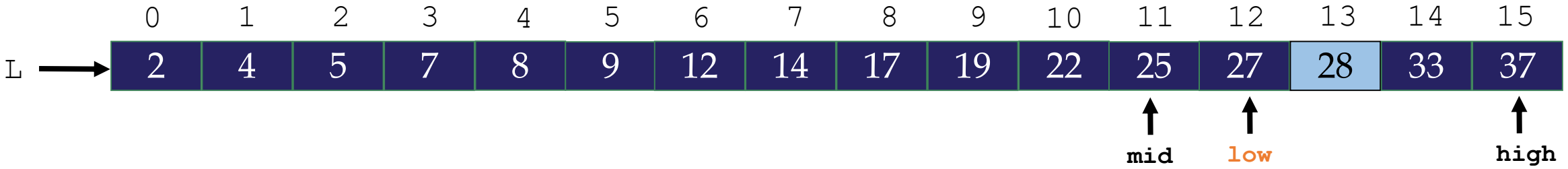
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

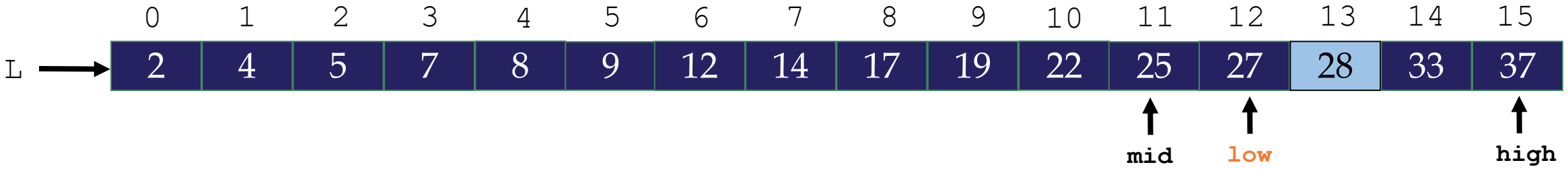
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

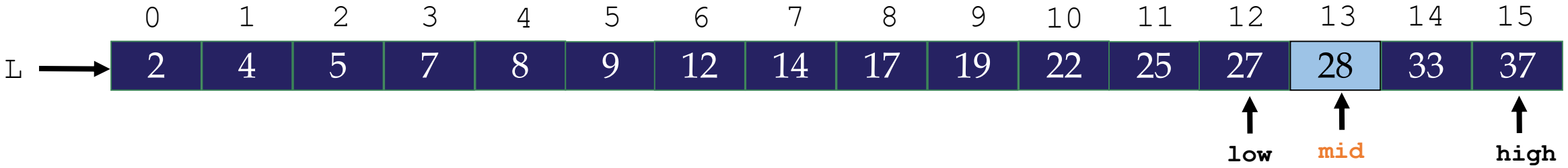
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

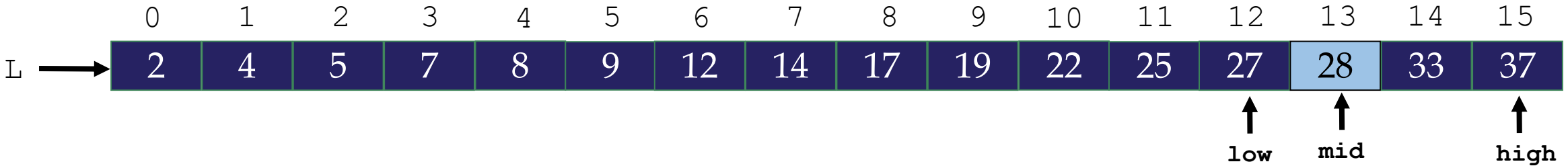
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

Binary Search Algorithm

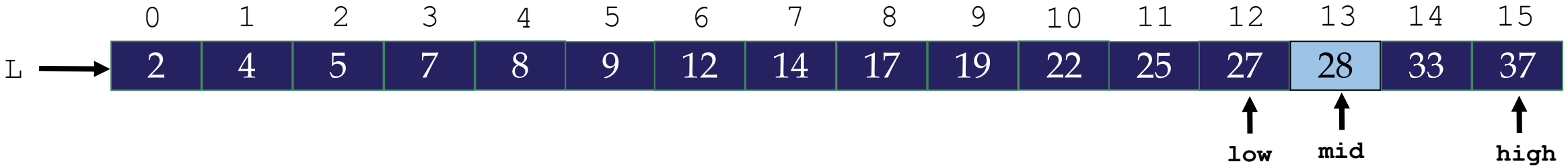


Pseudo-code: (target value is 28)

1. **Set low = 0**
2. **Set high = length of list - 1**
3. **Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer**
4. **If the value at the mid position is the same as the target value**
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. **As long as low doesn't 'cross over' high, go back to step 3 above**

PDST

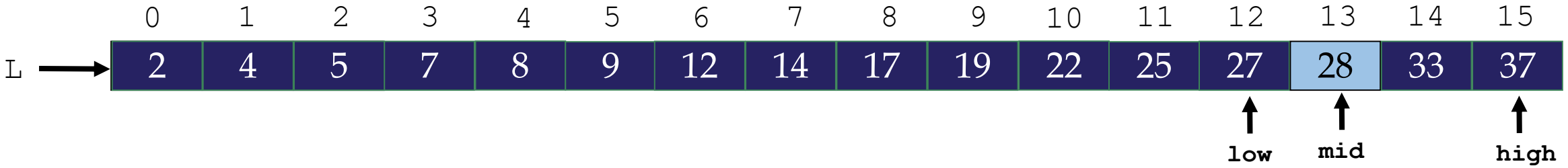
Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

Binary Search Algorithm



Pseudo-code: (target value is 28)

1. Set low = 0
2. Set high = length of list - 1
3. Set mid = $\frac{\text{low+high}}{2}$, rounded down to an integer
4. If the value at the mid position is the same as the target value
   Return mid
   Else If the value at the mid position is less than the target value
   Set low = mid + 1
   Else If the value at the mid position is greater than the target value
   Set high = mid - 1
5. As long as low doesn't 'cross over' high, go back to step 3 above

13 is returned (as it is the value of mid)
This is the index of the target element.

Q. How many comparisons were needed?

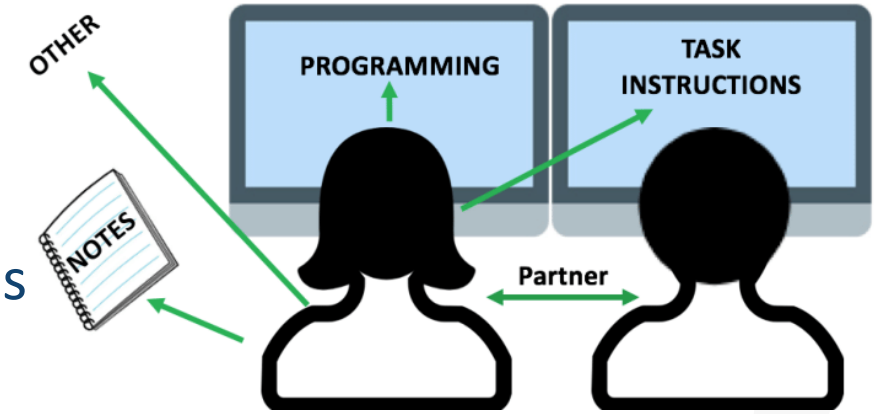
Q. How many comparisons would be needed for the linear search?

Breakout Activity: Analysis of Search Algorithms

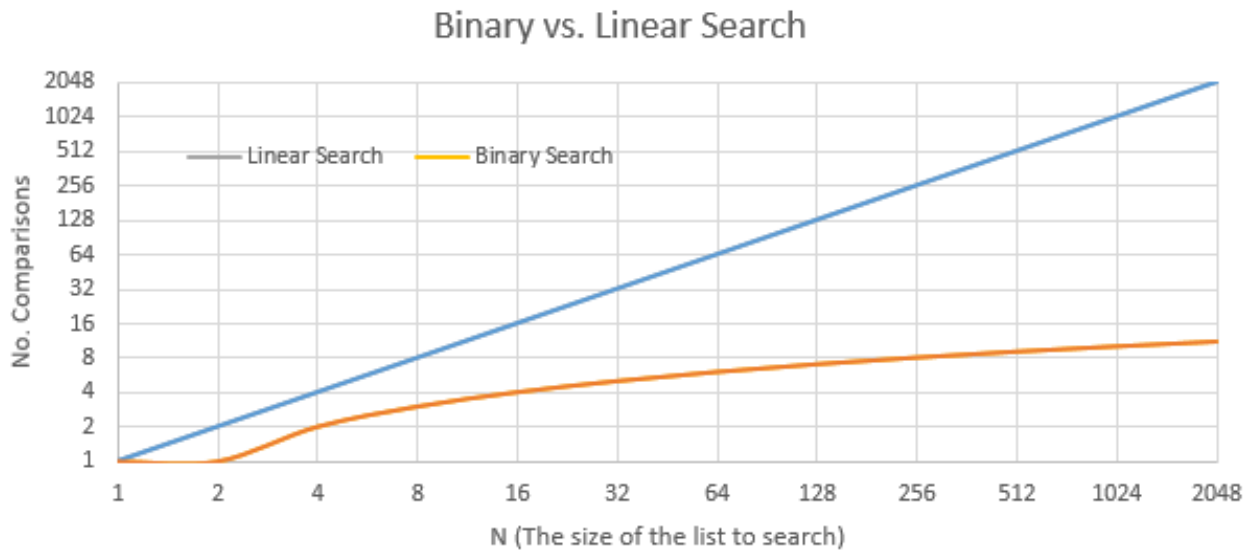
Instructions :

1. Participants work in pairs (pair programming)

2. Each pair opens the Google doc provided and completes the tasks



TASK:
Use the analysis framework provided to test the assertion that the binary search is exponentially faster than the linear search (see graph)



Summary: Algorithmic Time Complexity

Always consider the running time and the expected format of the input list before choosing a search or sorting algorithm for a particular problem.

	Best Case	Average Case	Worst Case
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(log_2 n)$	$O(log_2 n)$
Simple (selection) Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n\ log_2 n)$	$O(n\ log_2 n)$	$O(n^2)$

An Roinn Oideachais
Department of Education

© PDST 2022

PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Appendix 1

Quicksort (example 2)

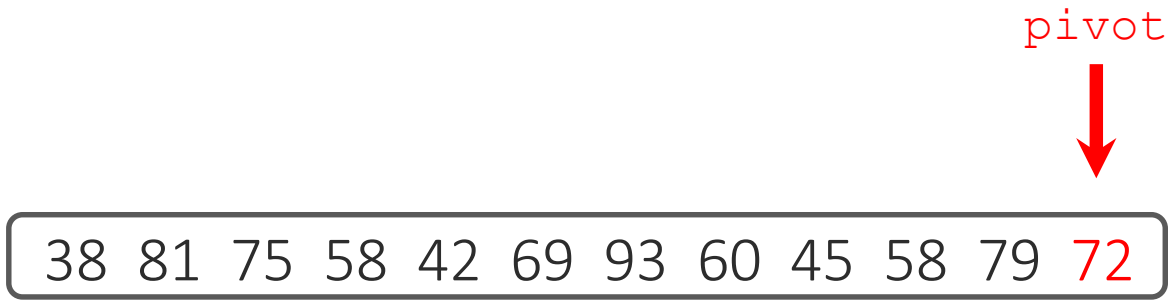
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

38 81 75 58 42 69 93 60 45 58 79 72

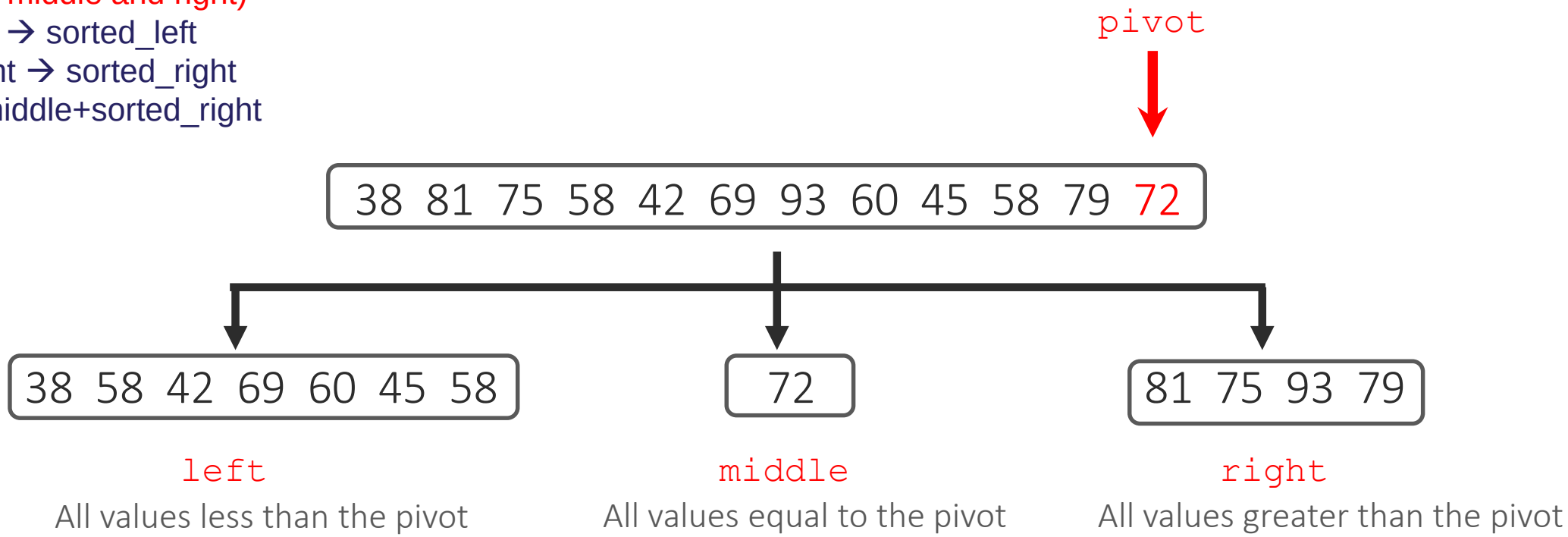
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

38 81 75 58 42 69 93 60 45 58 79 72

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

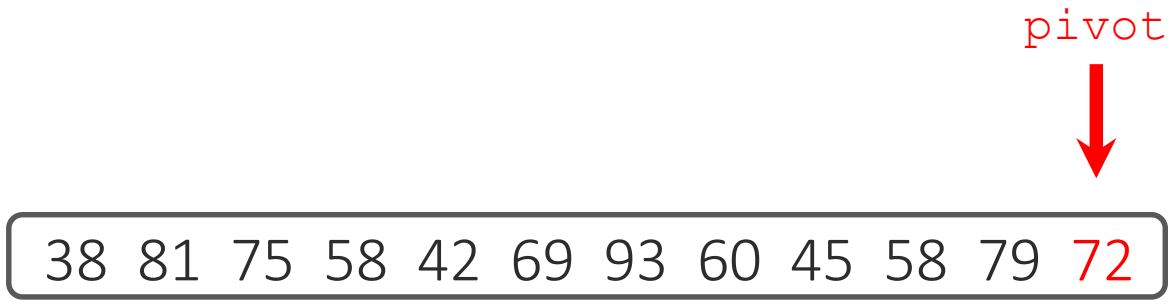


If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



Partitioning

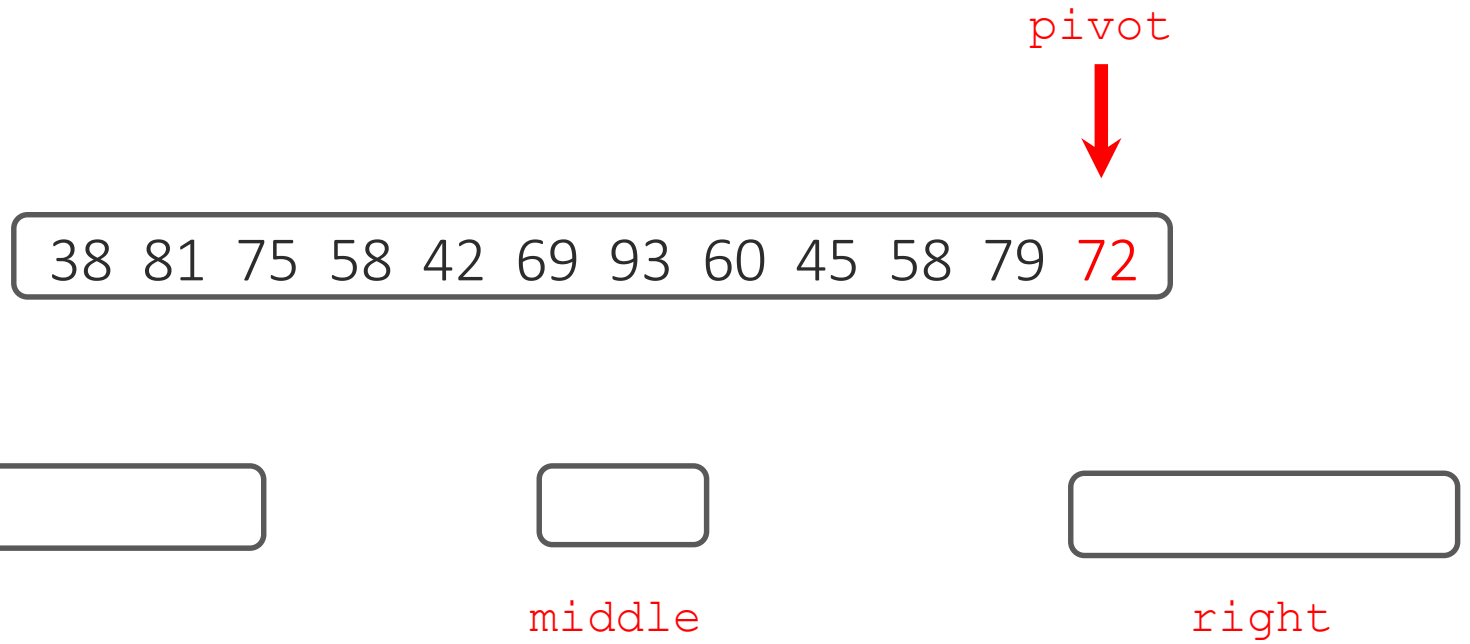
```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



Partitioning

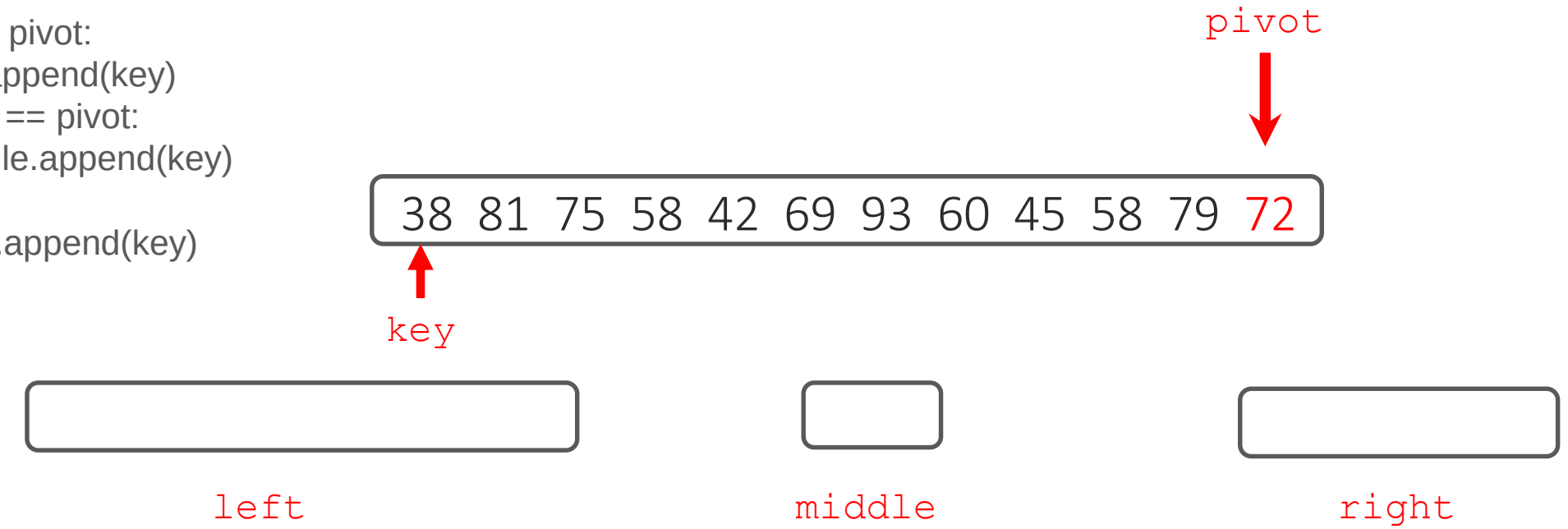
left = []; middle = []; right = []

```
for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



Partitioning

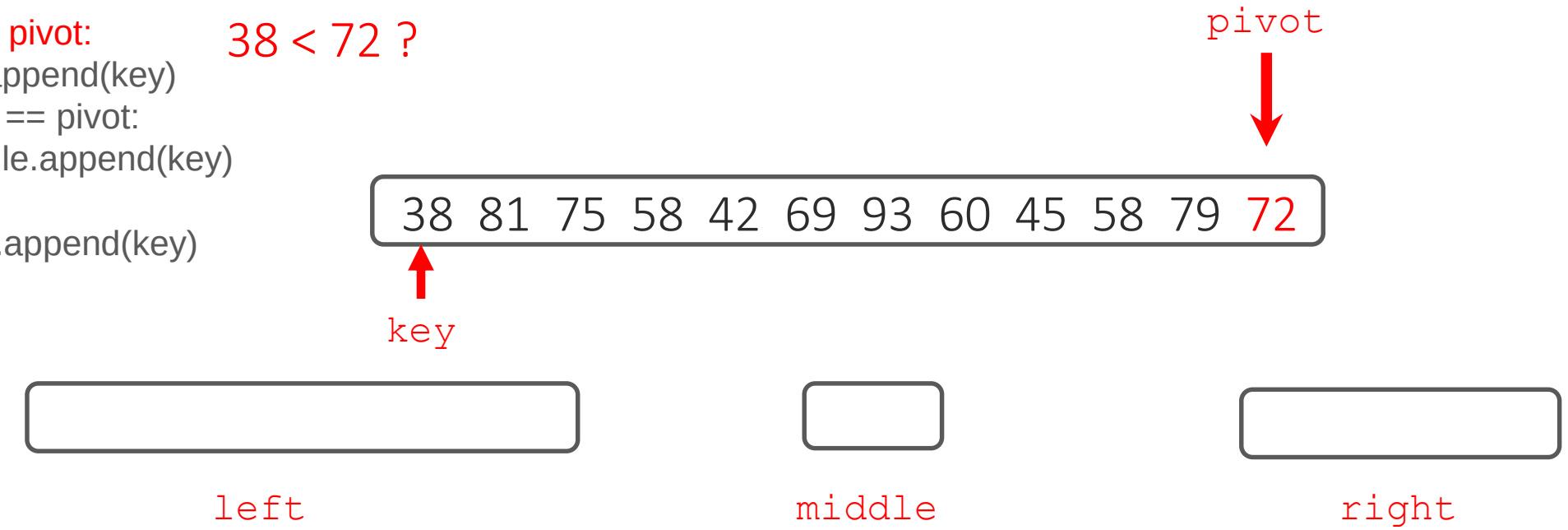
```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



Partitioning

```
left = []; middle = []; right = []

for key in L:
    if key < pivot:
        left.append(key)
    elif key == pivot:
        middle.append(key)
    else:
        right.append(key)
```

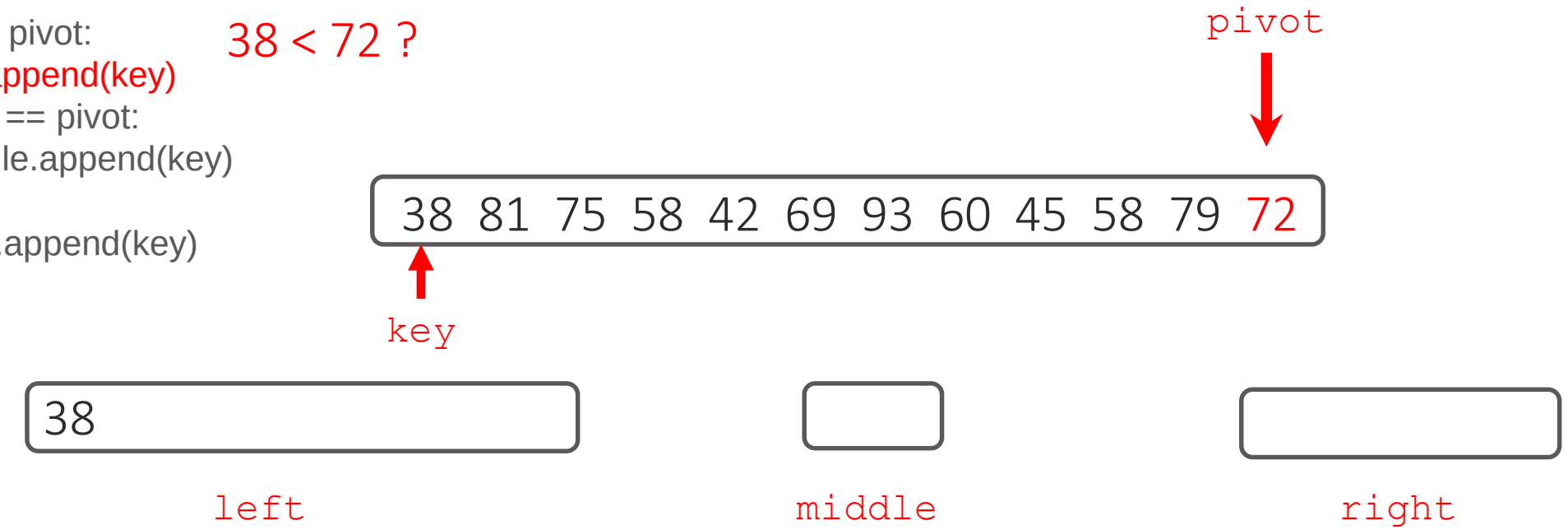
38 < 72 ?



Partitioning

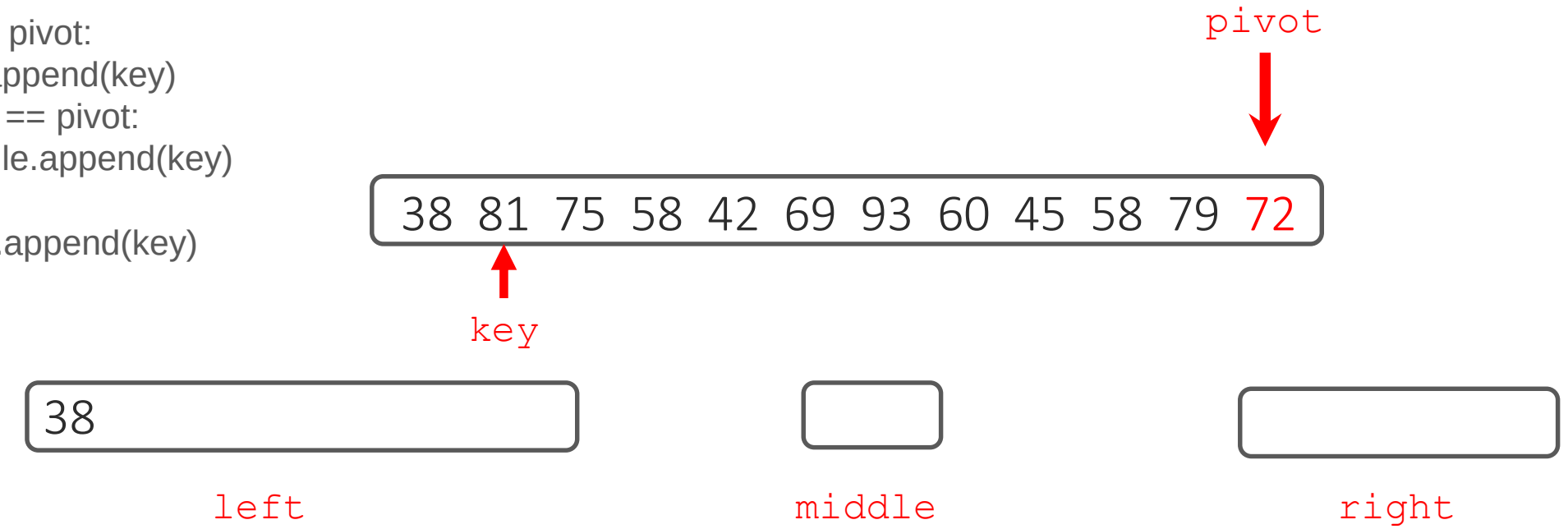
```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```

38 < 72 ?



Partitioning

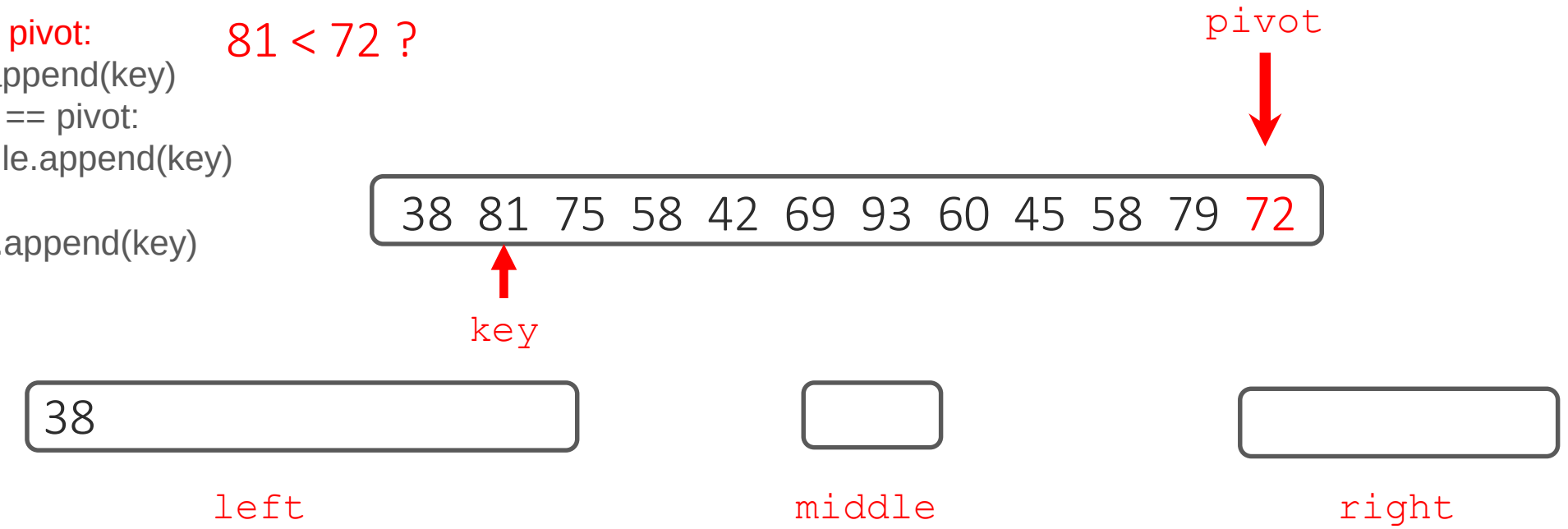
```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



Partitioning

```
left = []; middle = []; right = []

for key in L:
    if key < pivot:
        left.append(key)
    elif key == pivot:
        middle.append(key)
    else:
        right.append(key)
```

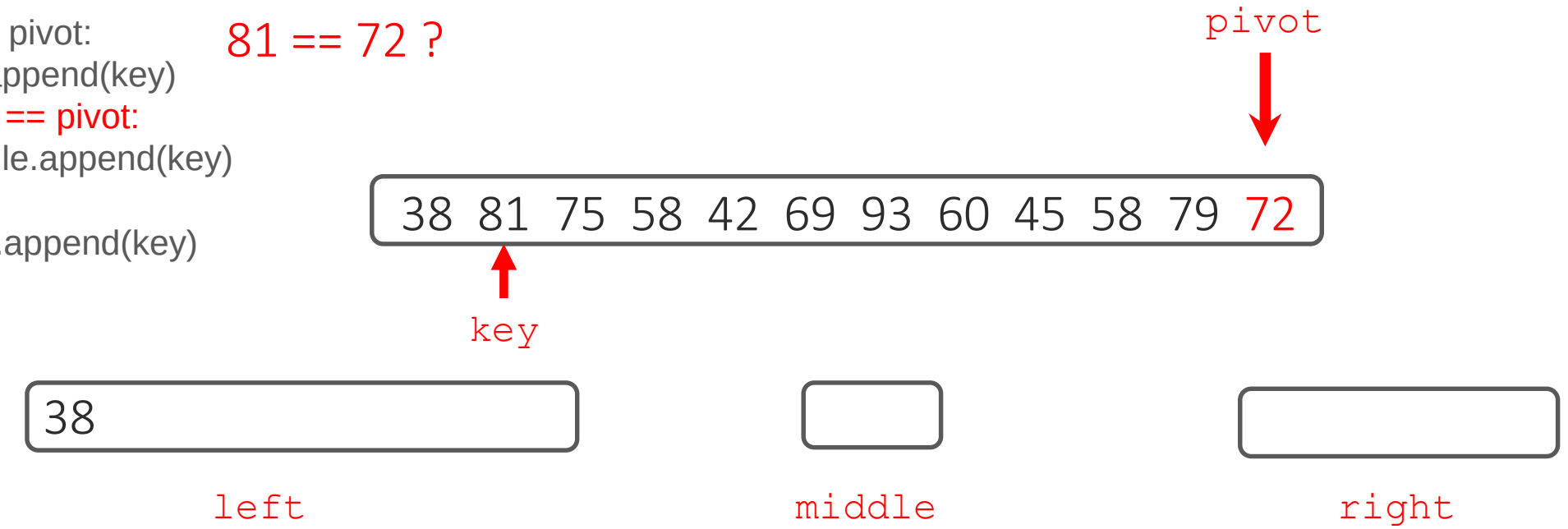
81 < 72 ?



Partitioning

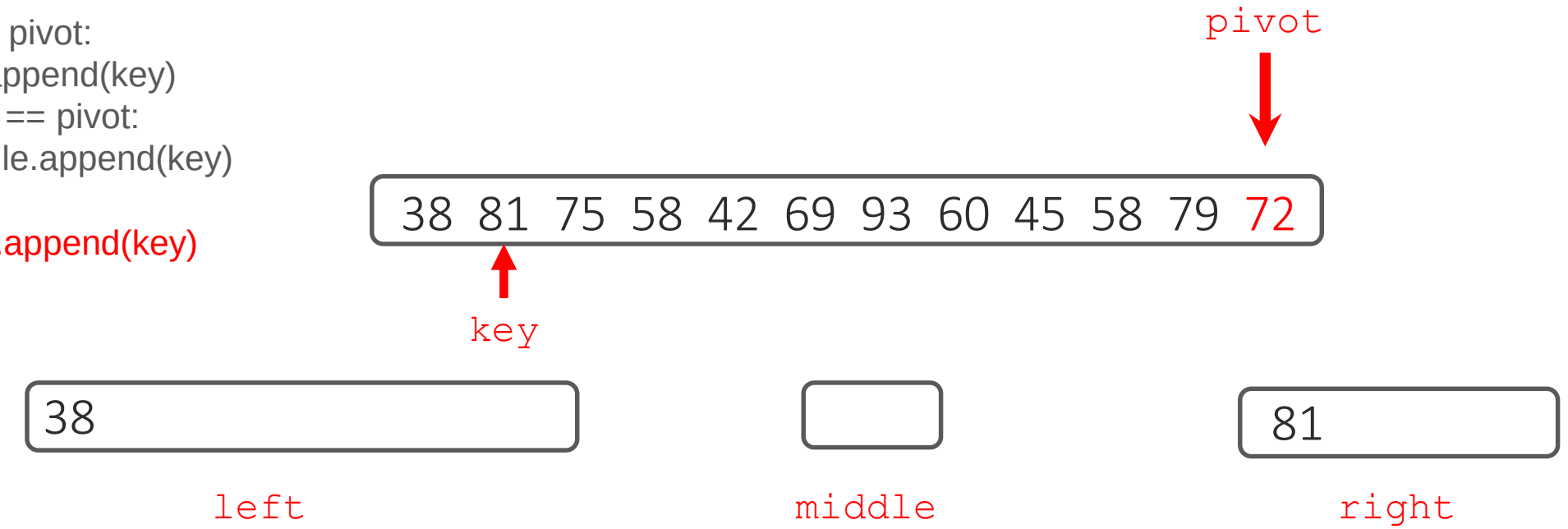
```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```

81 == 72 ?



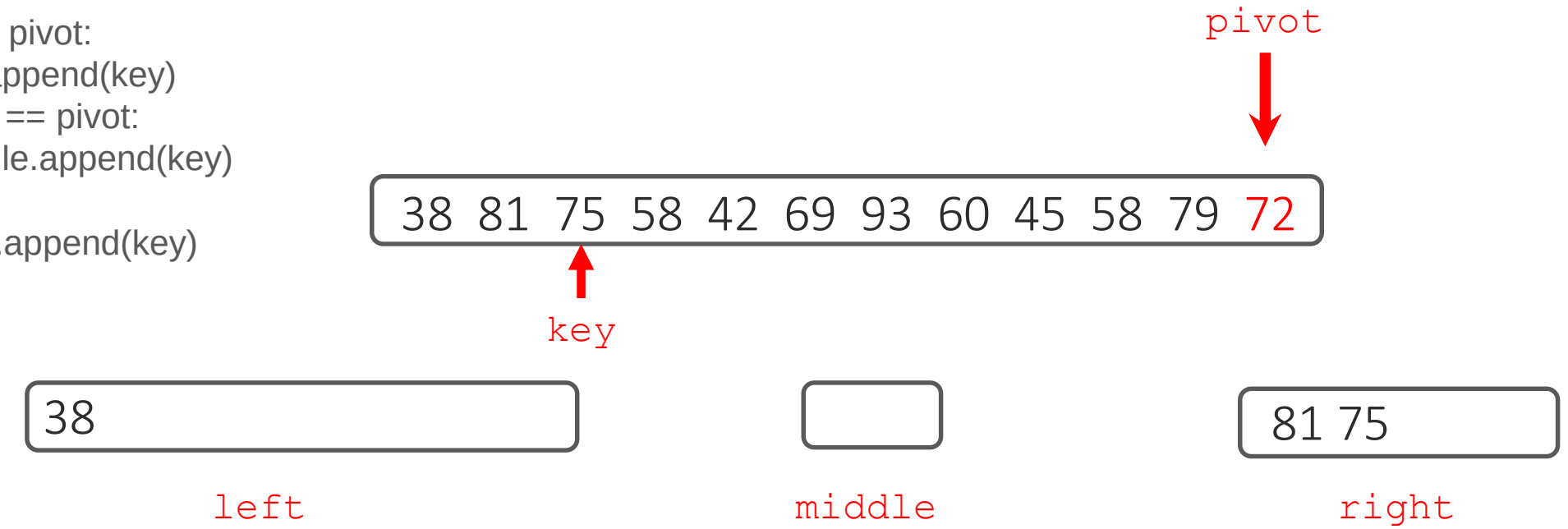
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



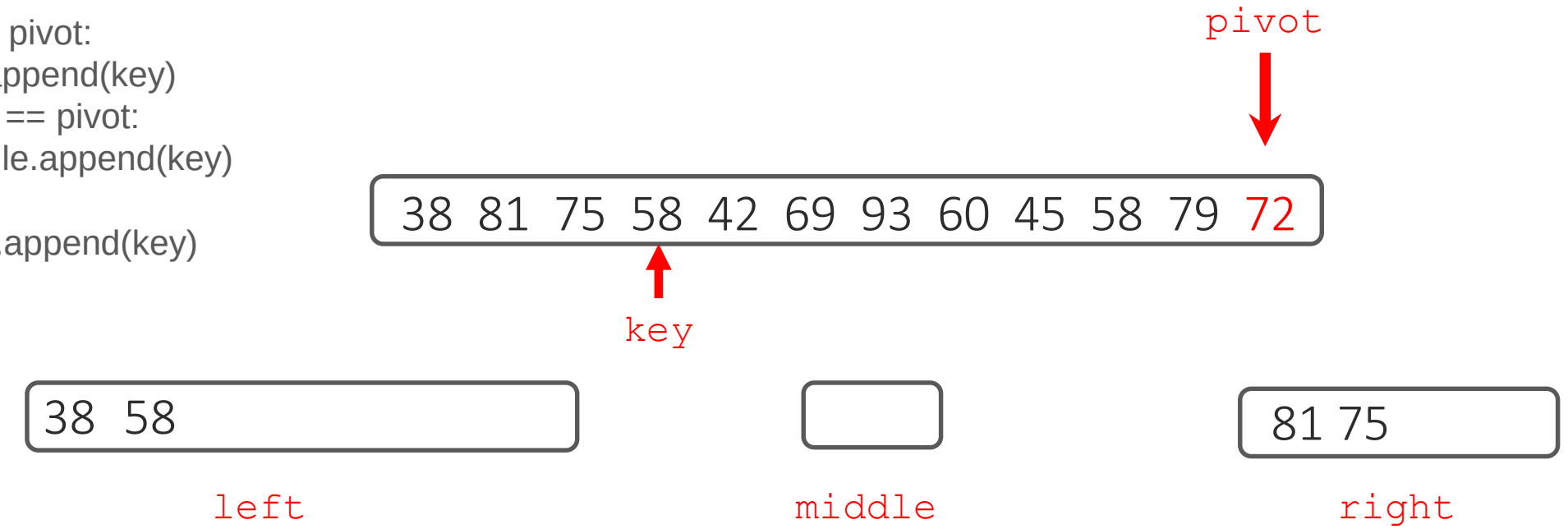
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



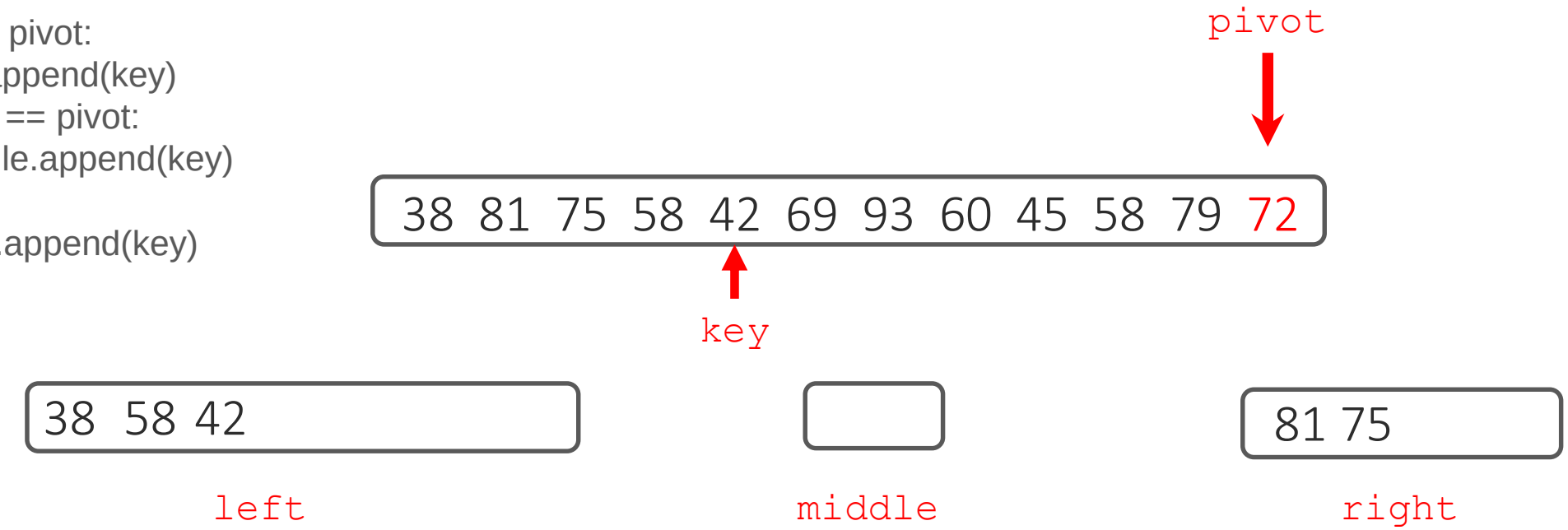
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



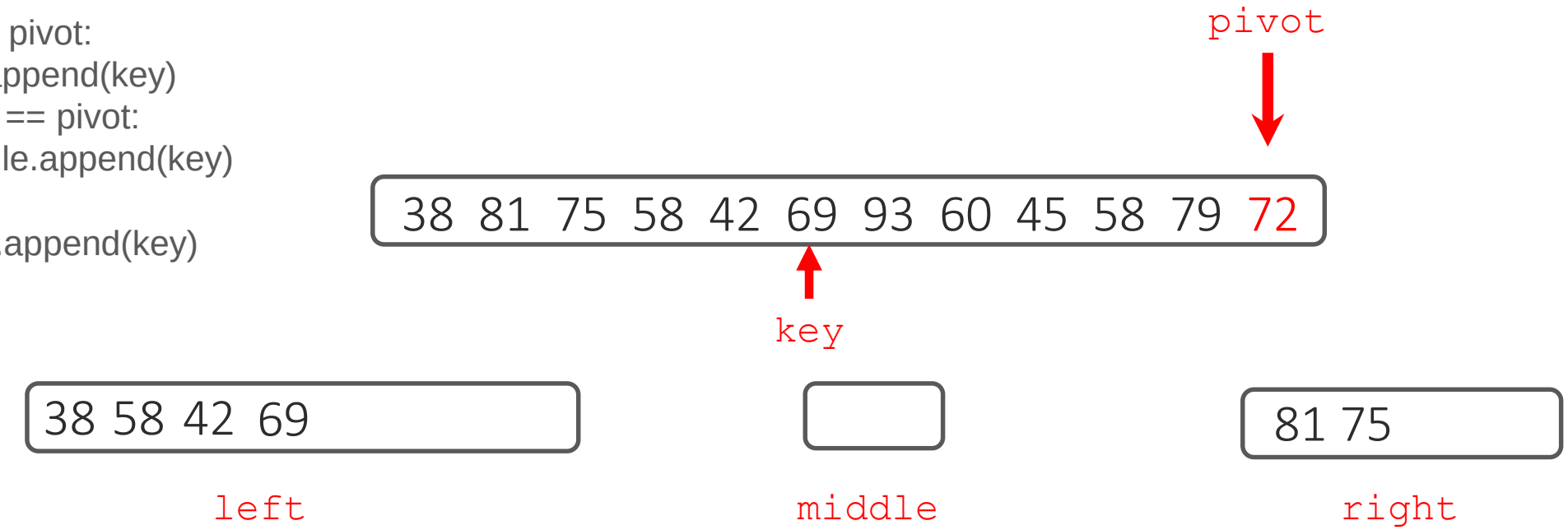
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



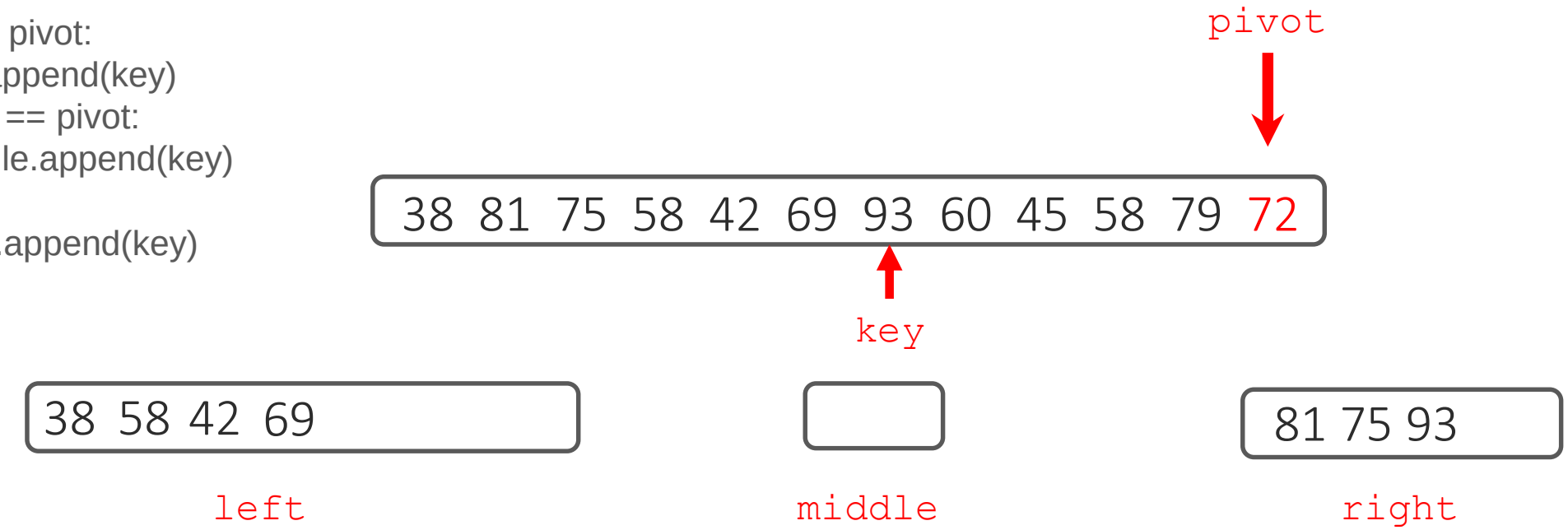
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



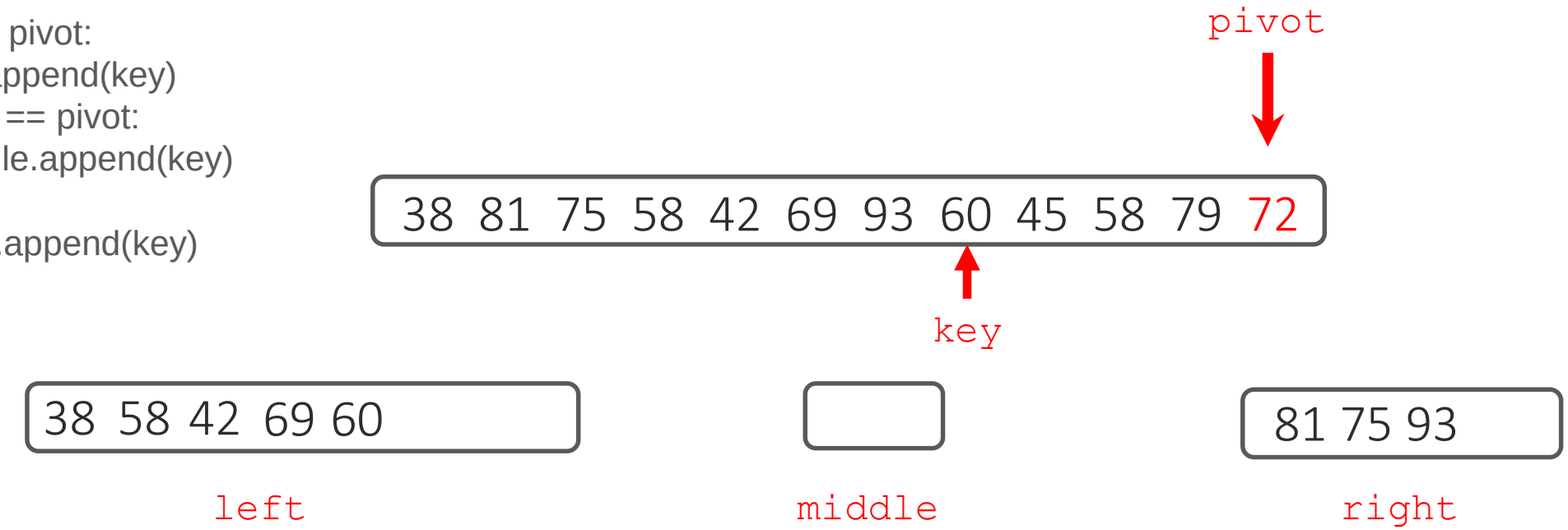
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



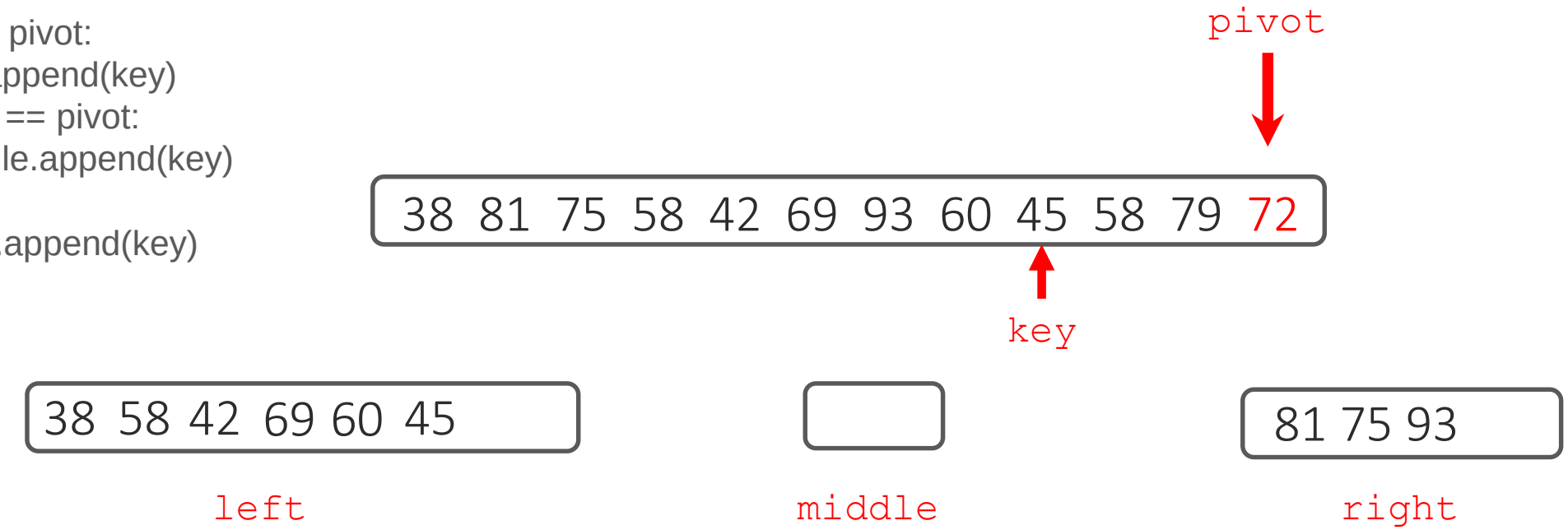
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



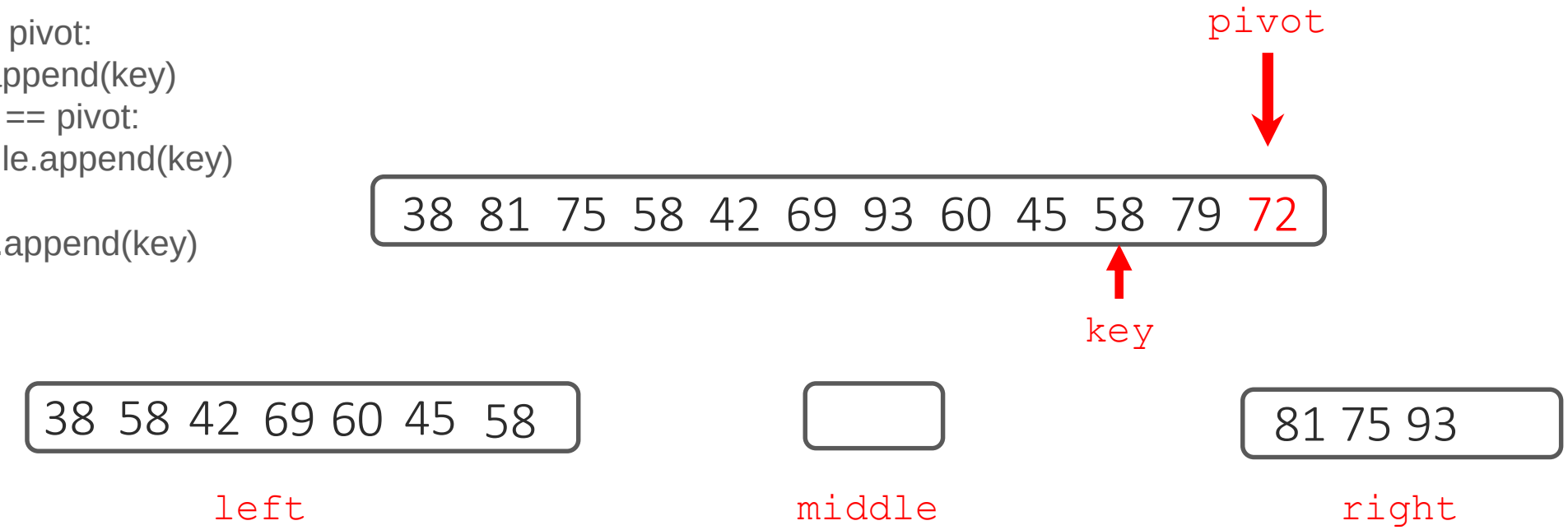
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



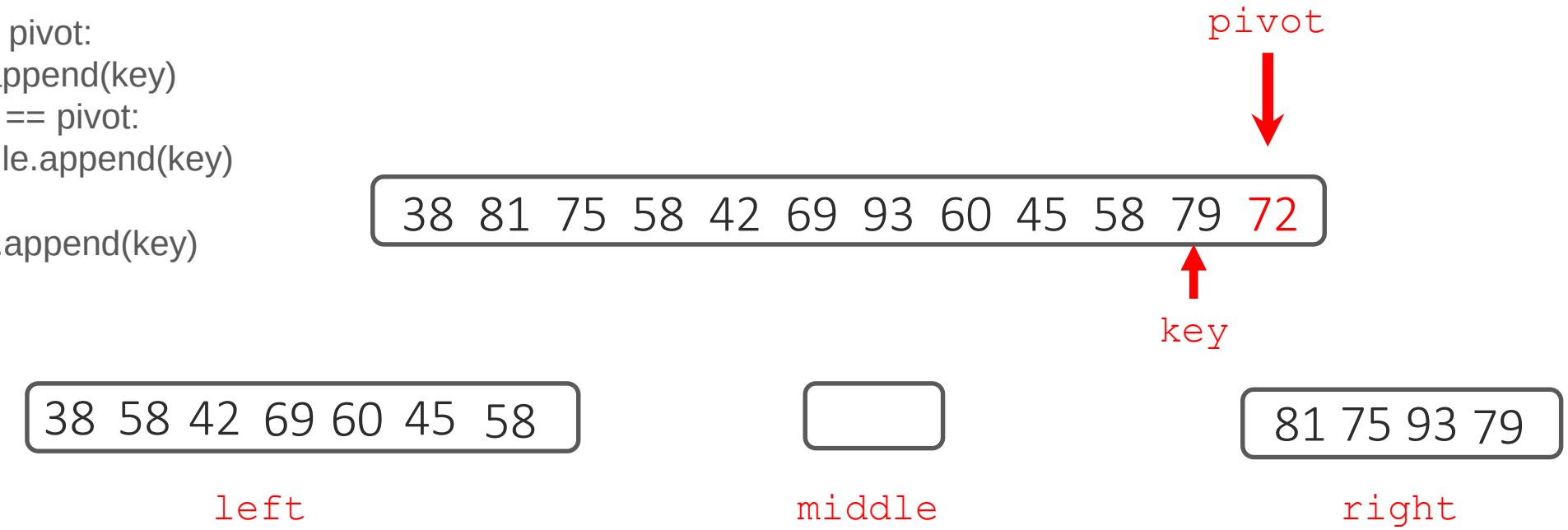
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



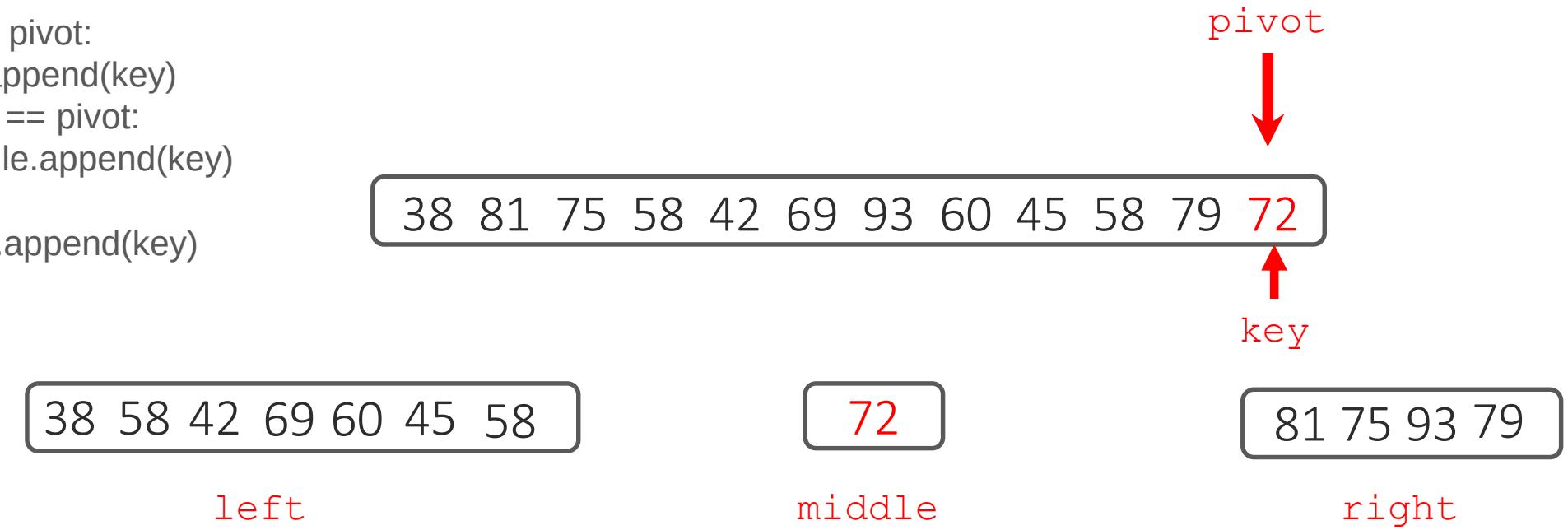
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```



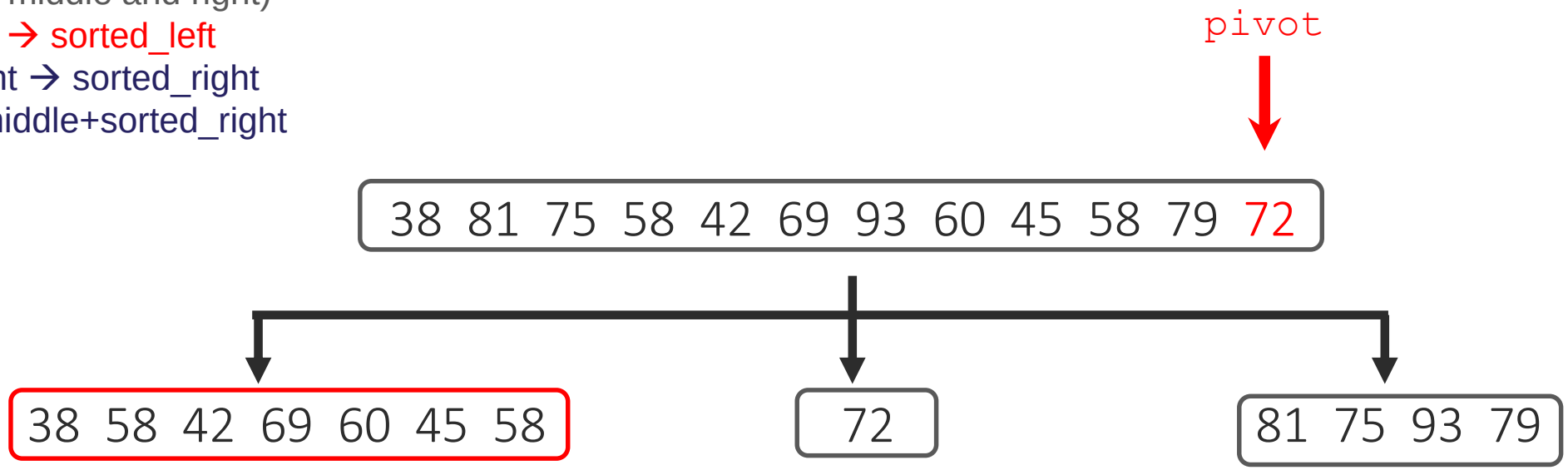
Partitioning

```
left = []; middle = []; right = []

for key in L:
        if key < pivot:
            left.append(key)
        elif key == pivot:
            middle.append(key)
        else:
            right.append(key)
```

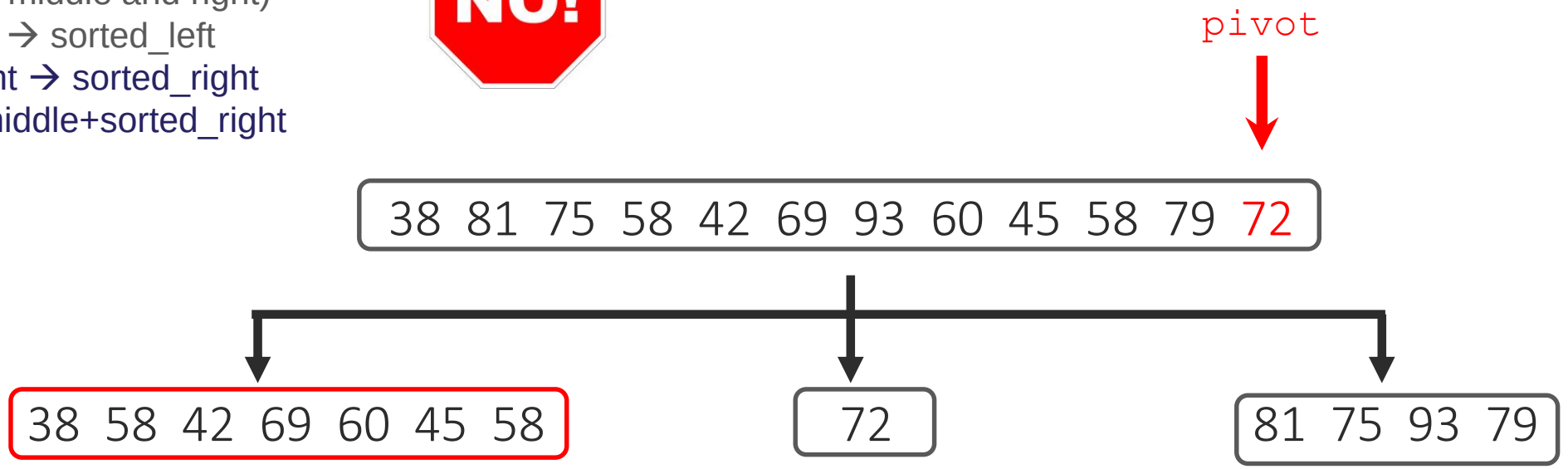


At this point the pivot is sorted into its final position

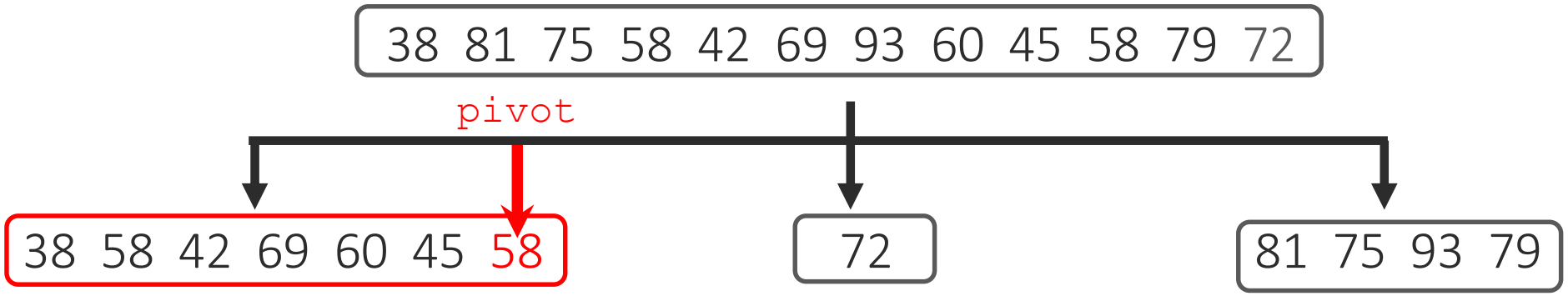
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



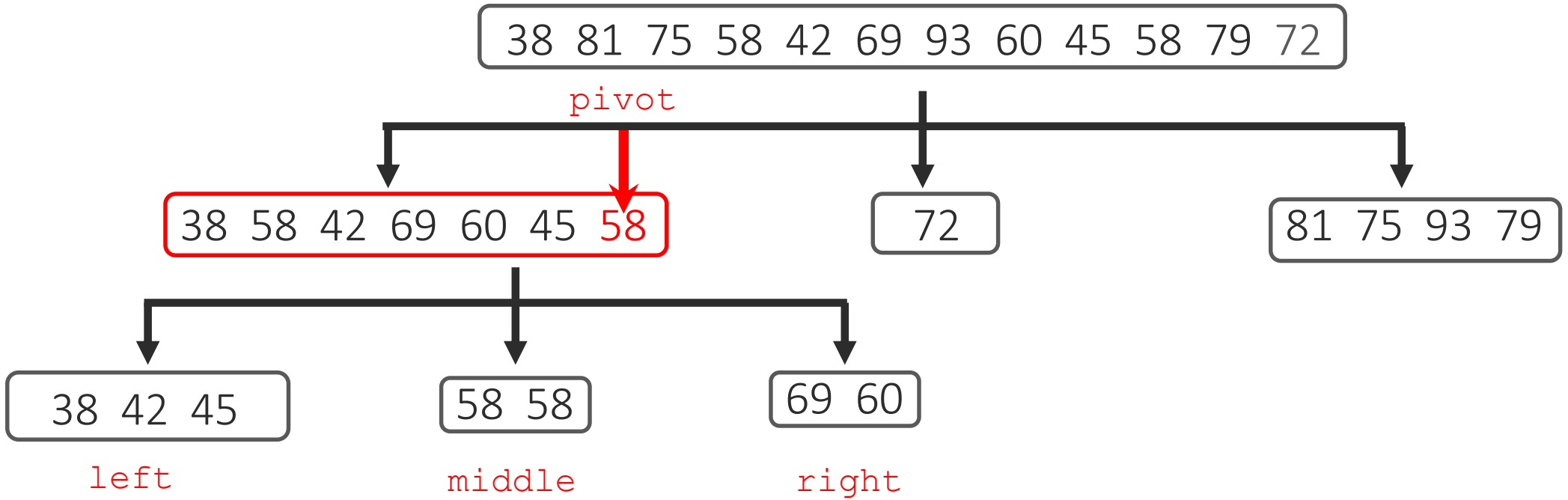
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



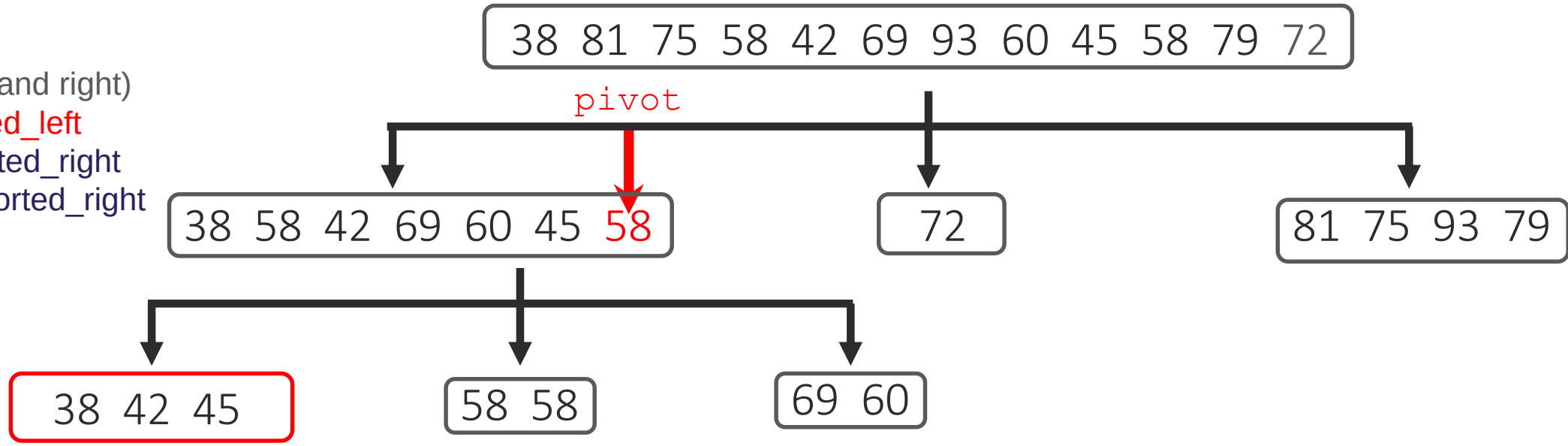
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L

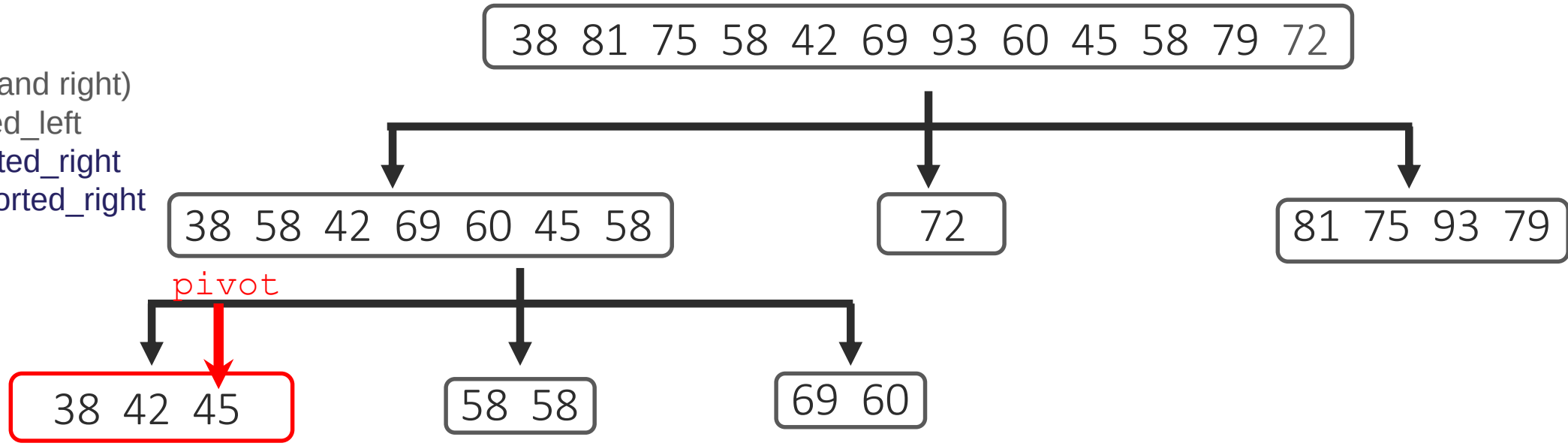
Choose the `pivot`

Partition (left, middle and right)

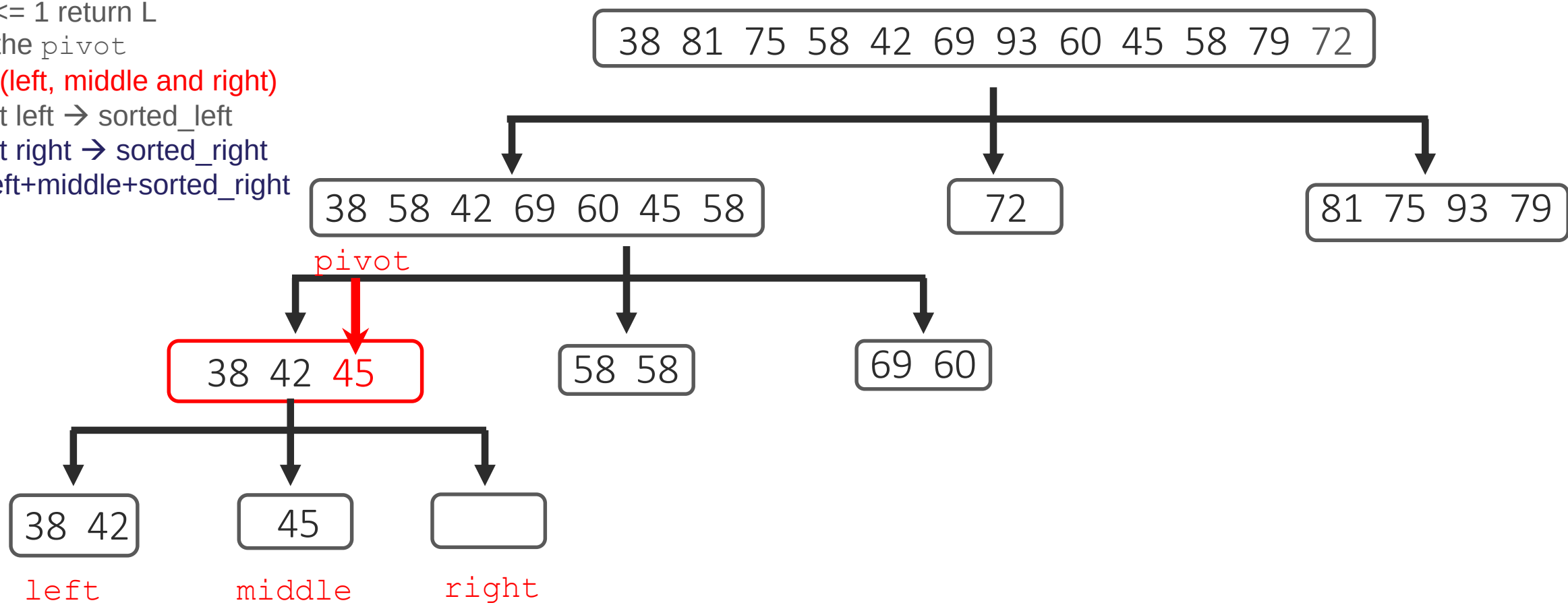
Quicksort left → sorted_left

Quicksort right → sorted_right

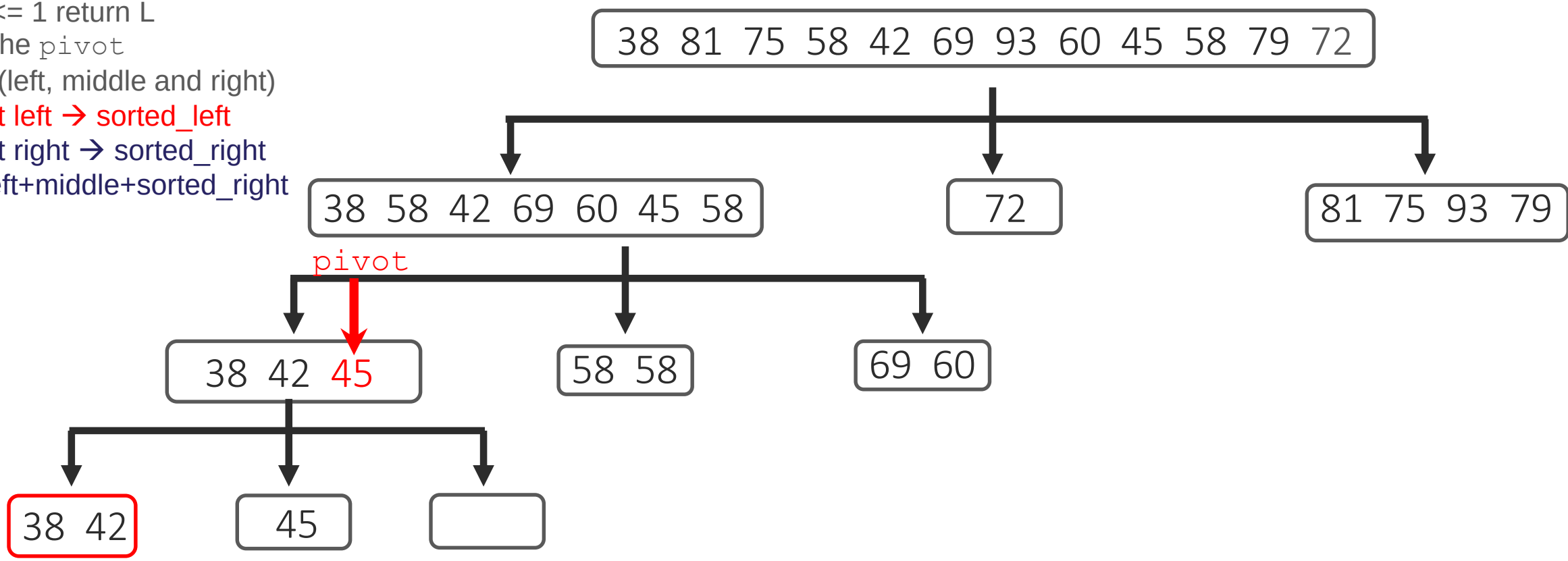
sorted_left+middle+sorted_right



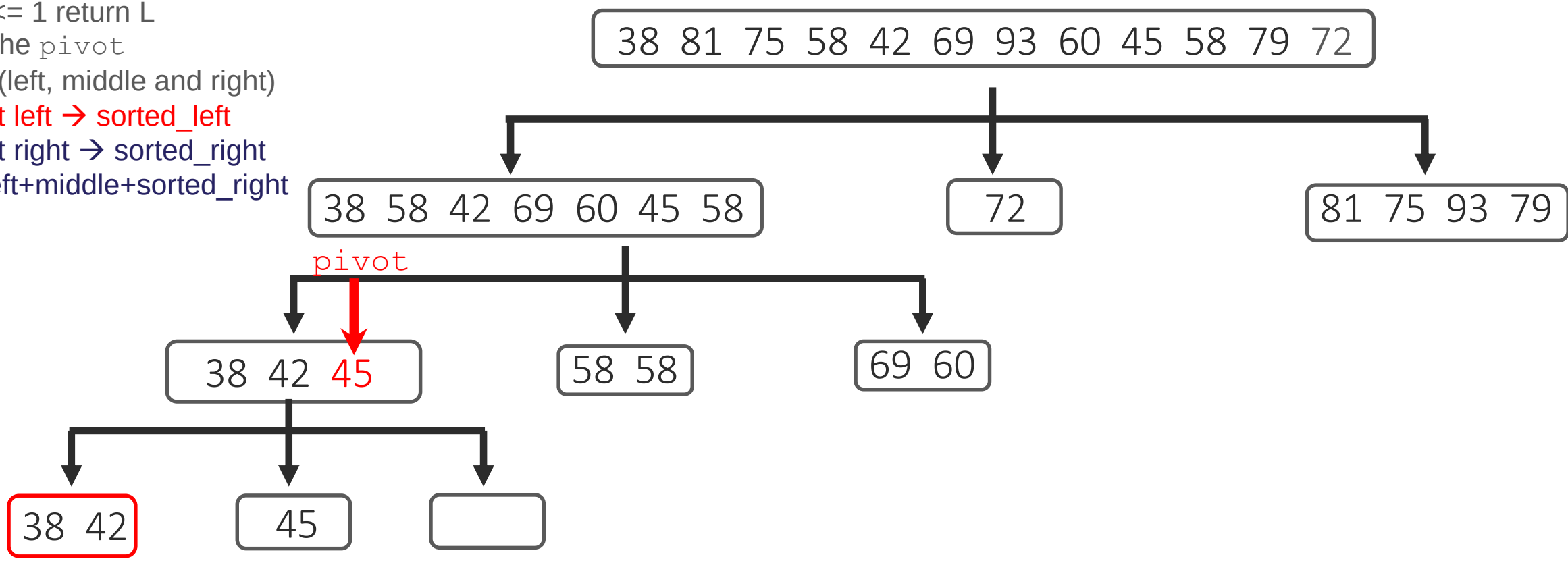
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



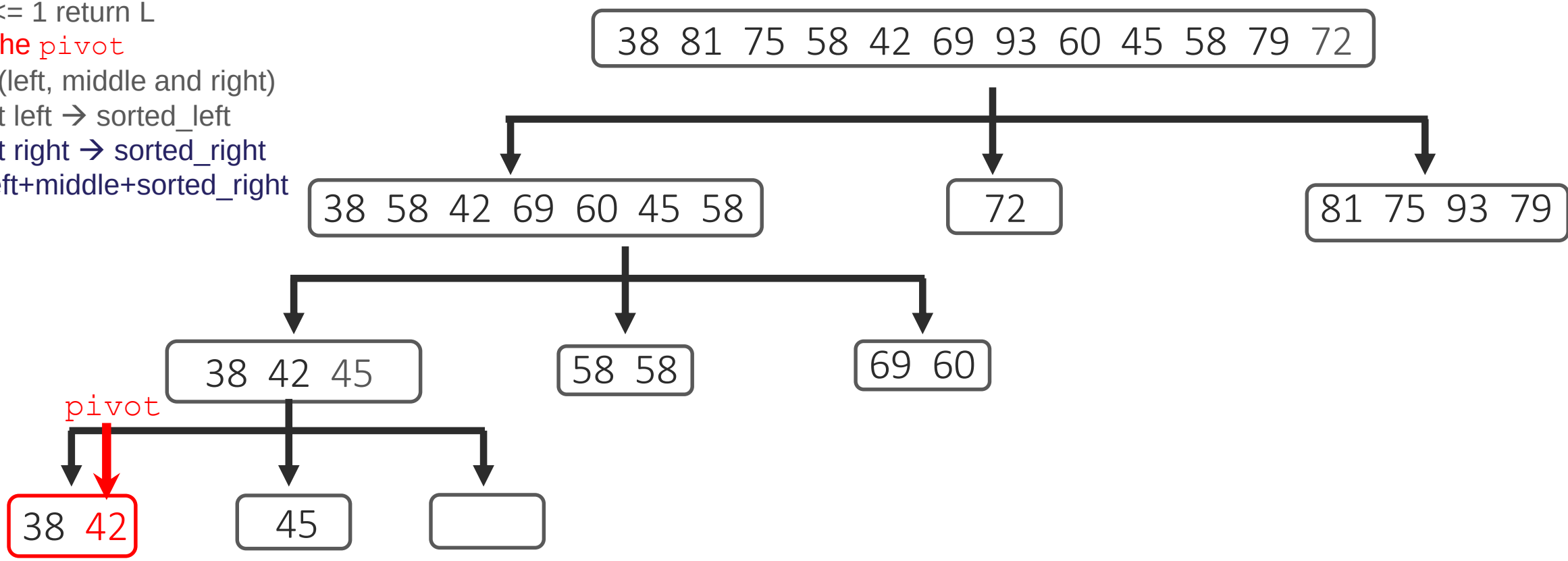
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



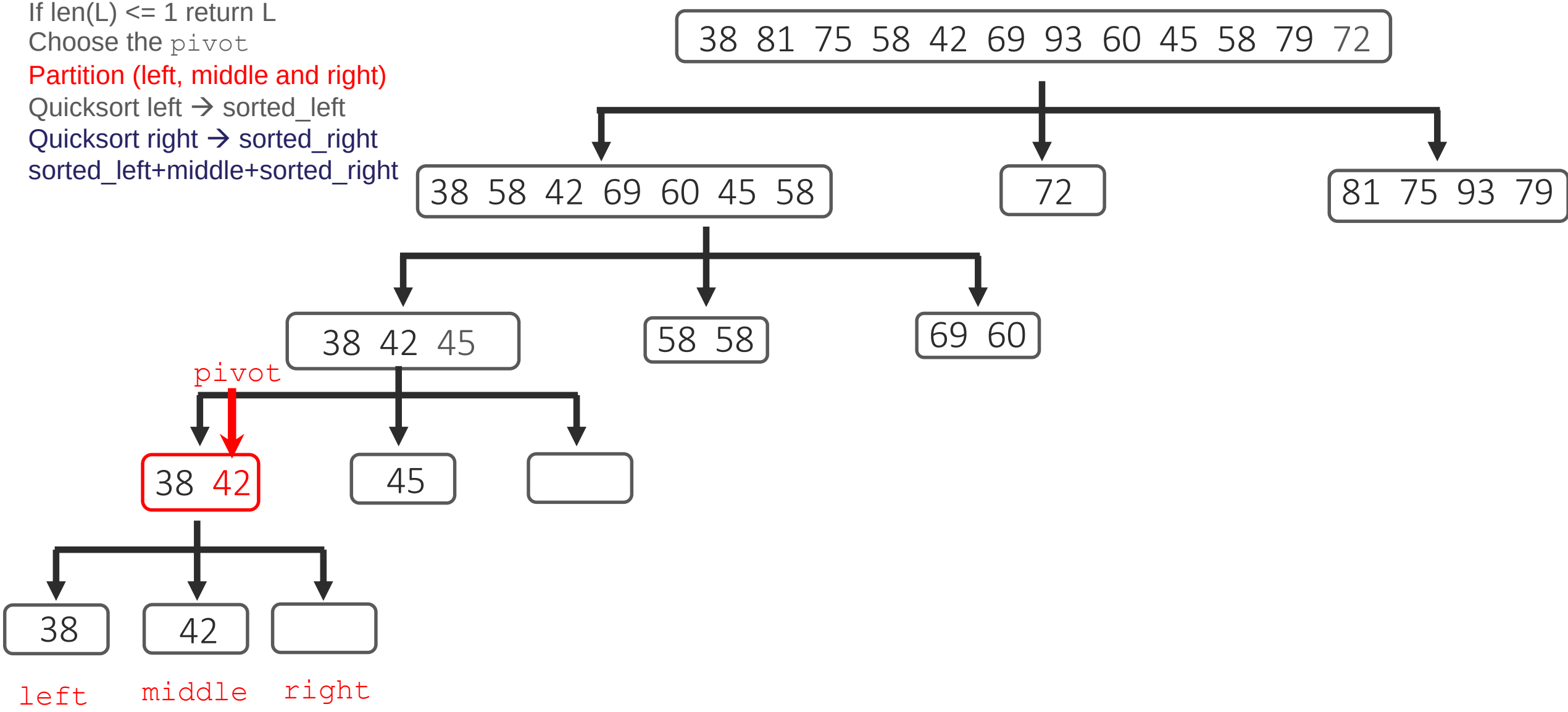
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



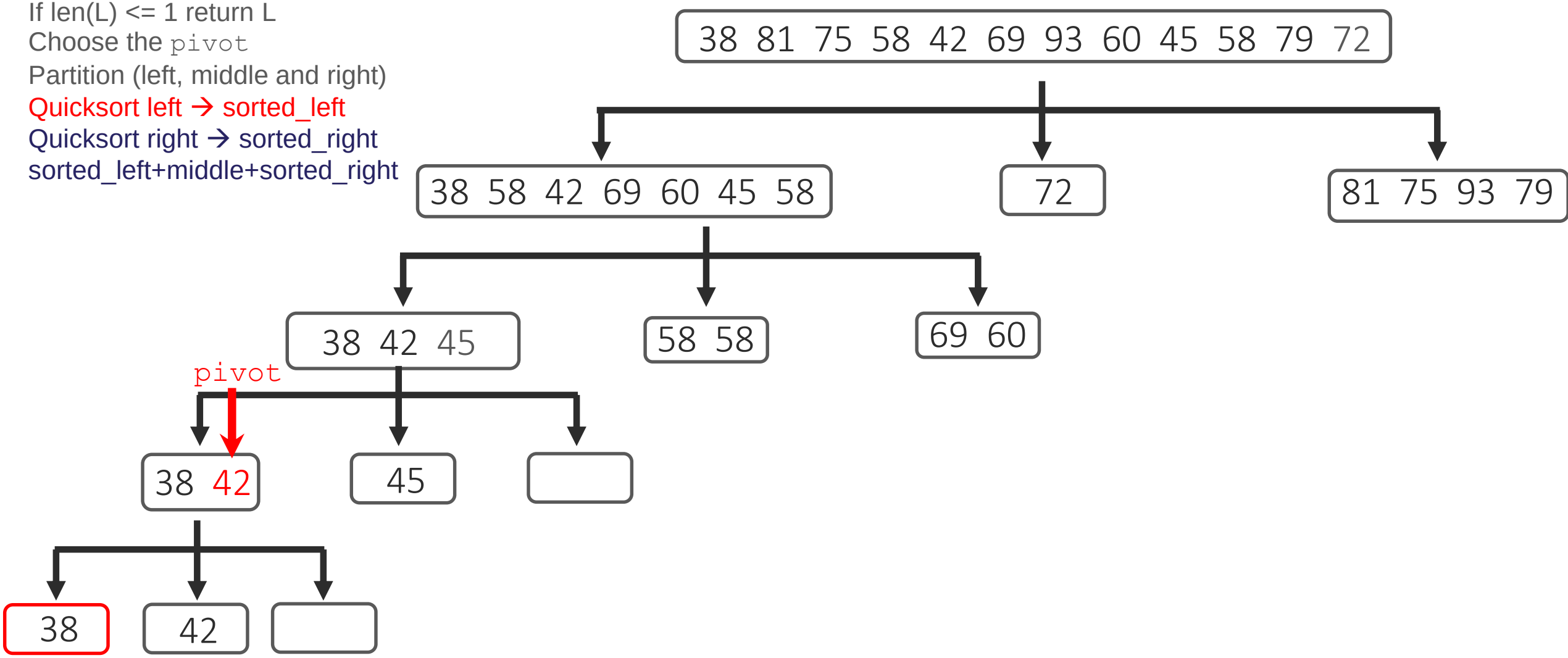
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



If len(L) <= 1 return L

Choose the `pivot`

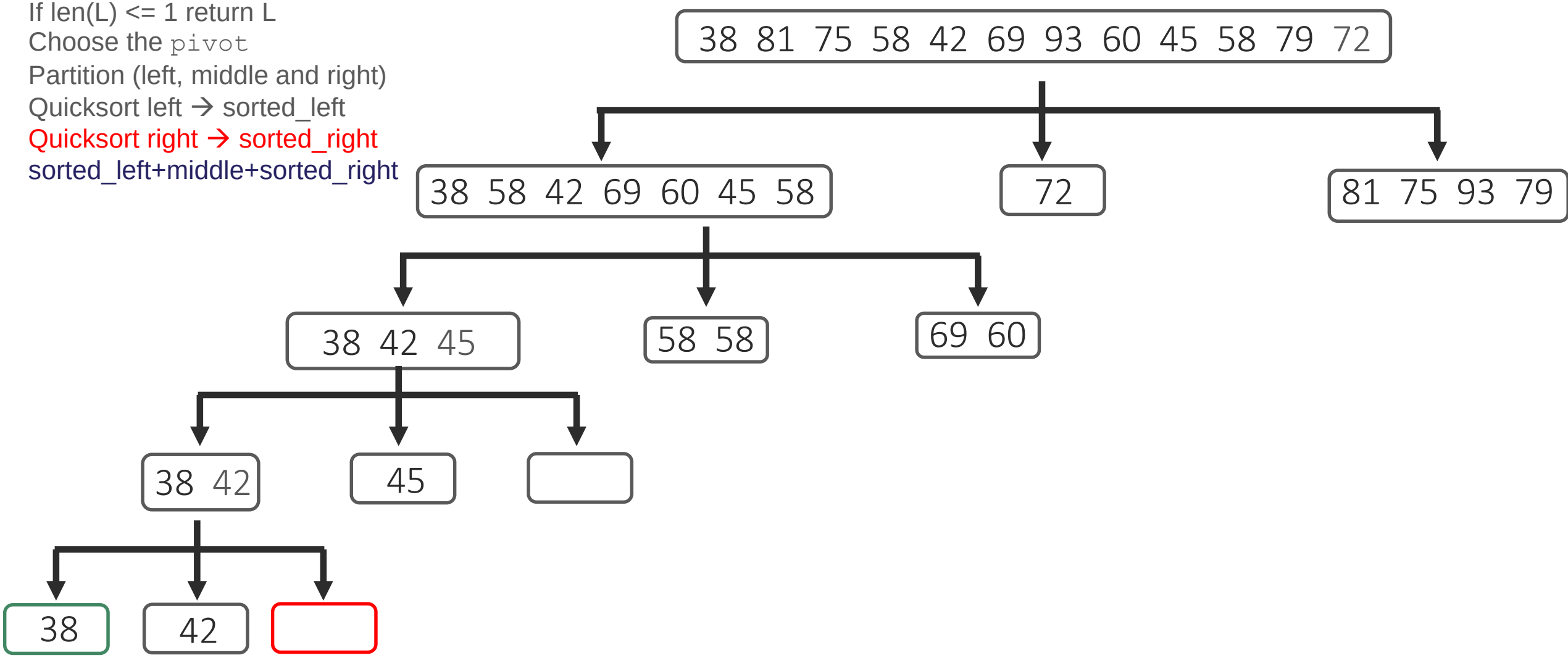
Partition (left, middle and right)

Quicksort left → sorted_left

Quicksort right → sorted_right

sorted_left+middle+sorted_right

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

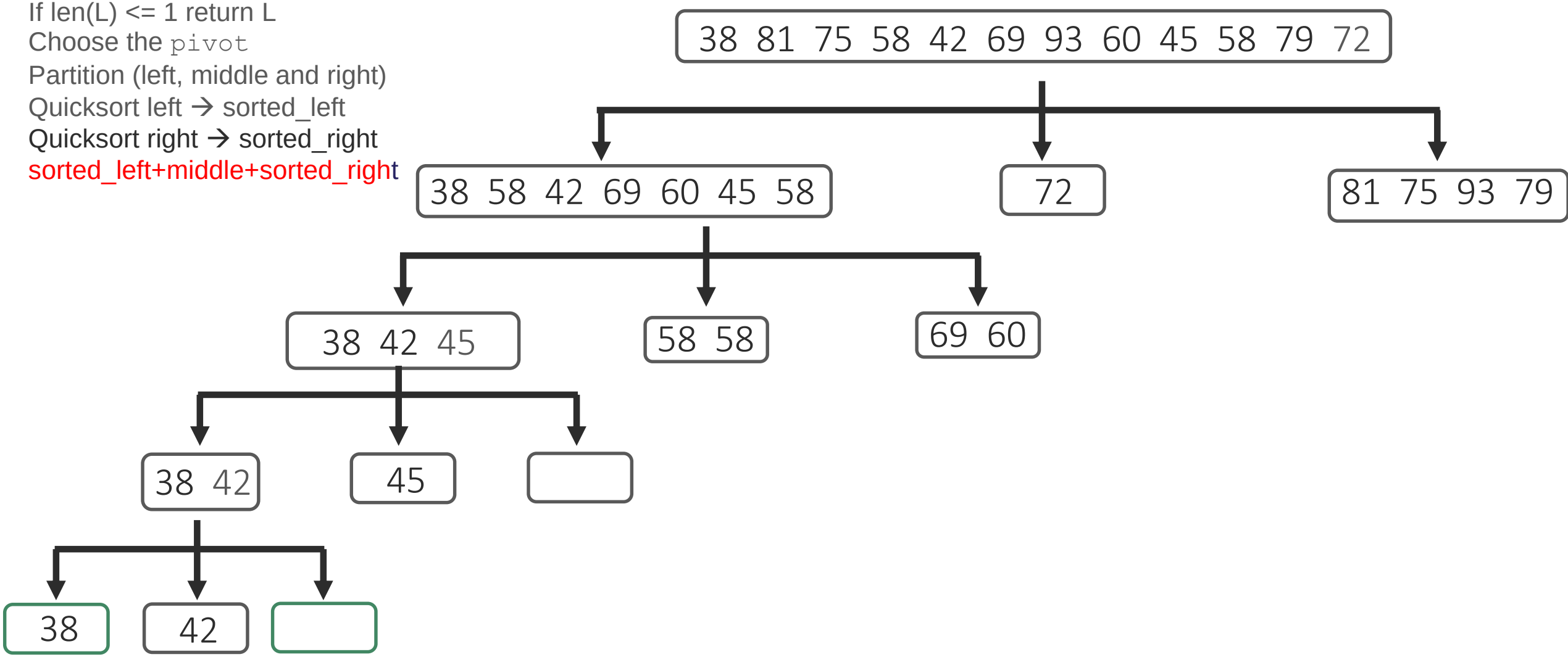


`sorted_left`: 38

`middle`: 42

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

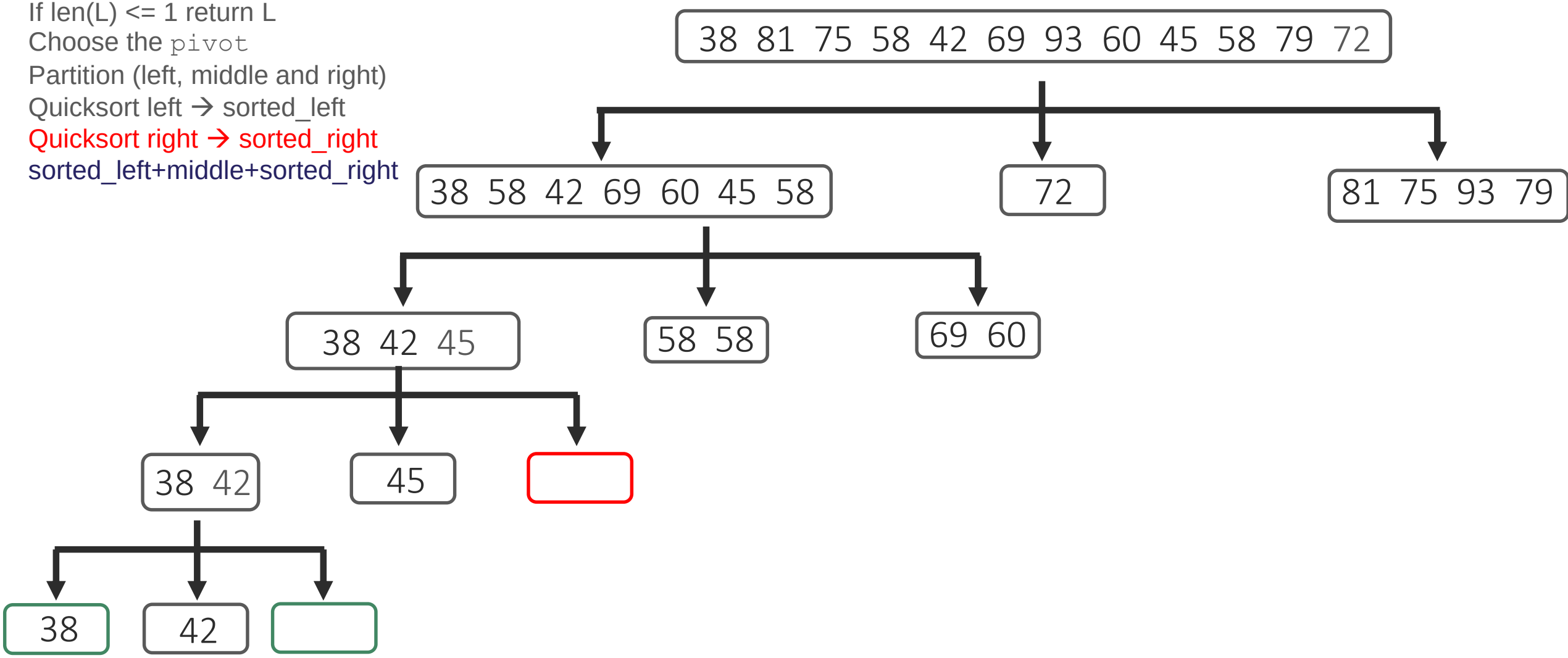


`sorted_left`: 38

`middle`: 42

`sorted_right`: []

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



`sorted_left`: 38 42

`middle`: 45

`sorted_right`:

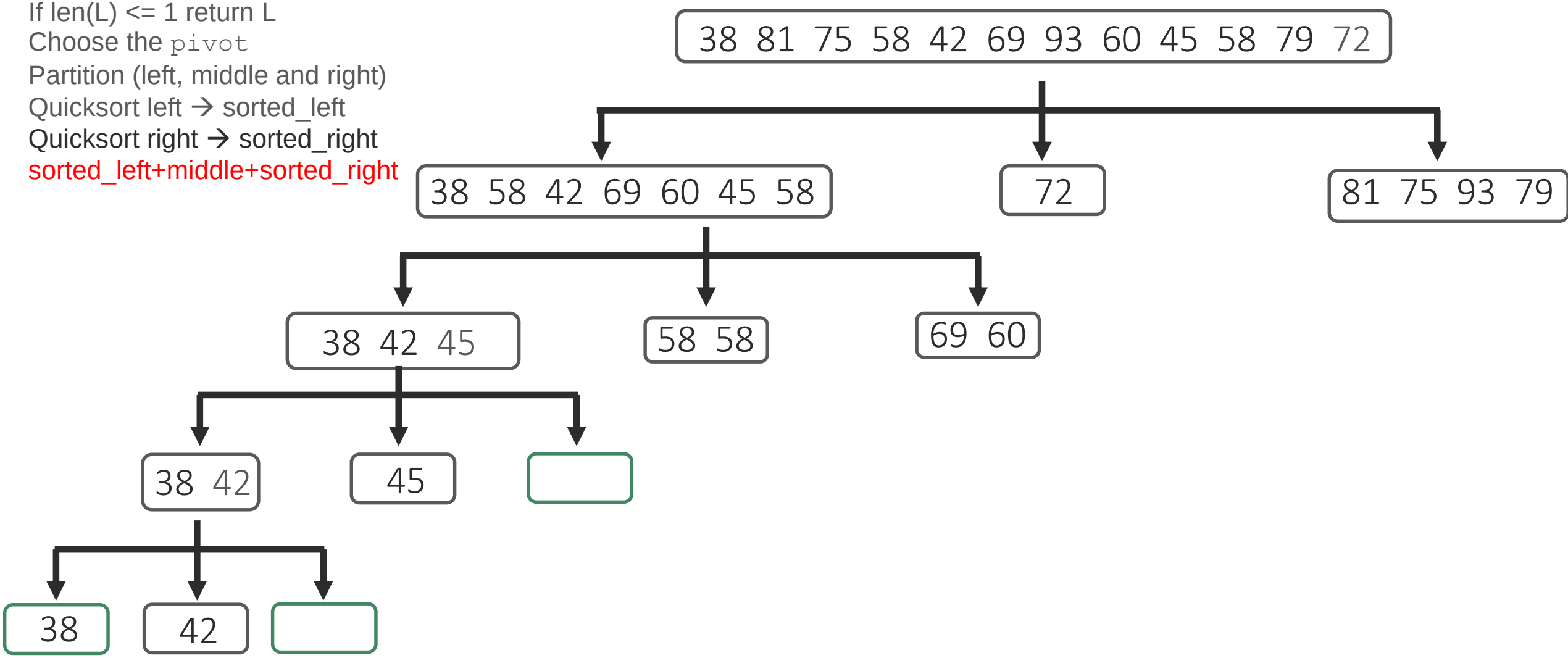
If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

`sorted_left`: 38 42

`middle`: 45

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

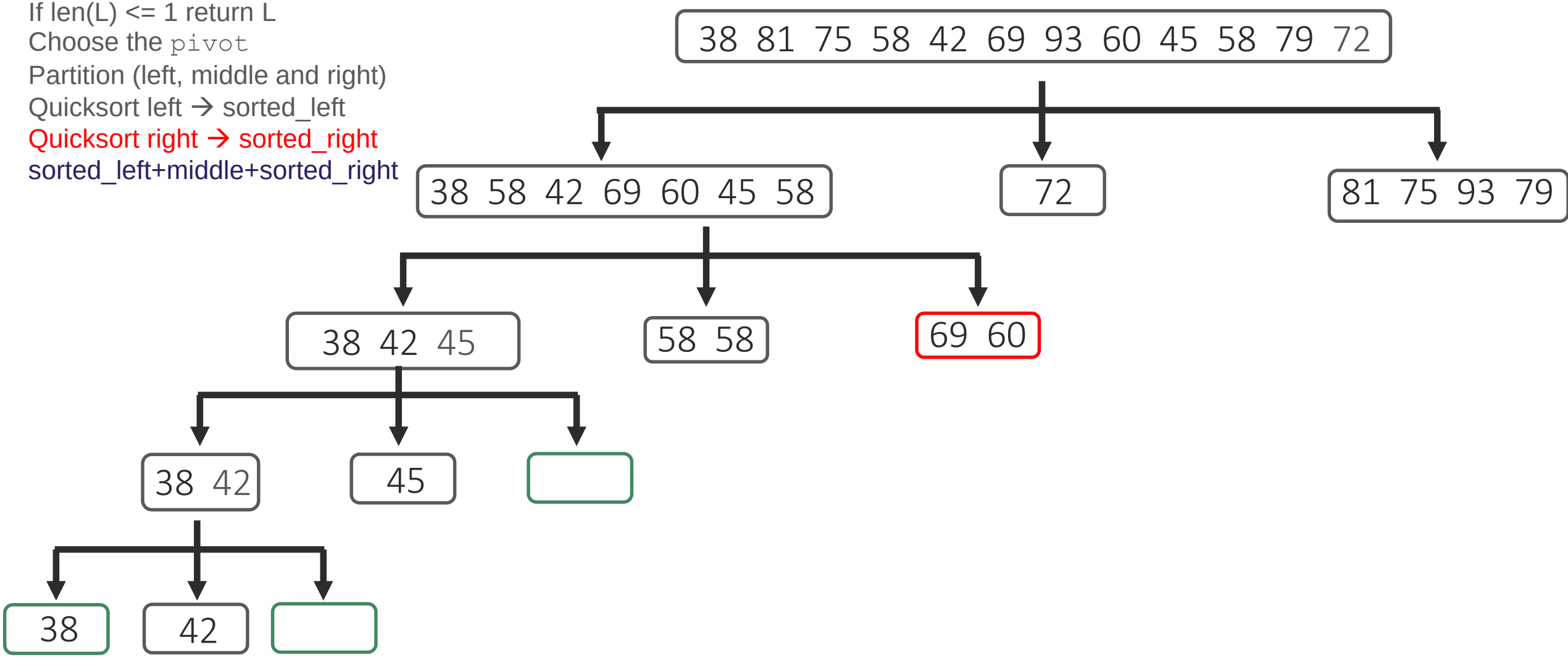


`sorted_left`: 38 42

`middle`: 45

`sorted_right`: []

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

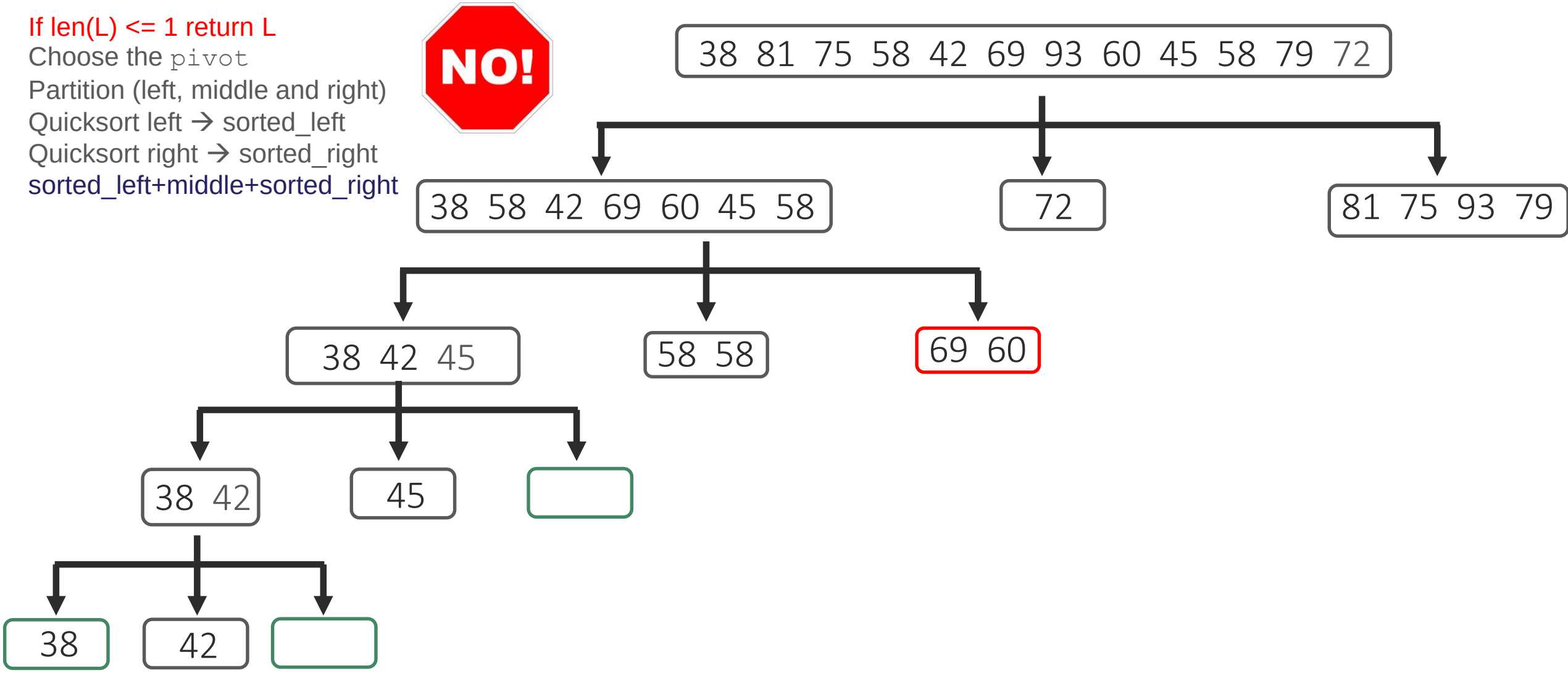


`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

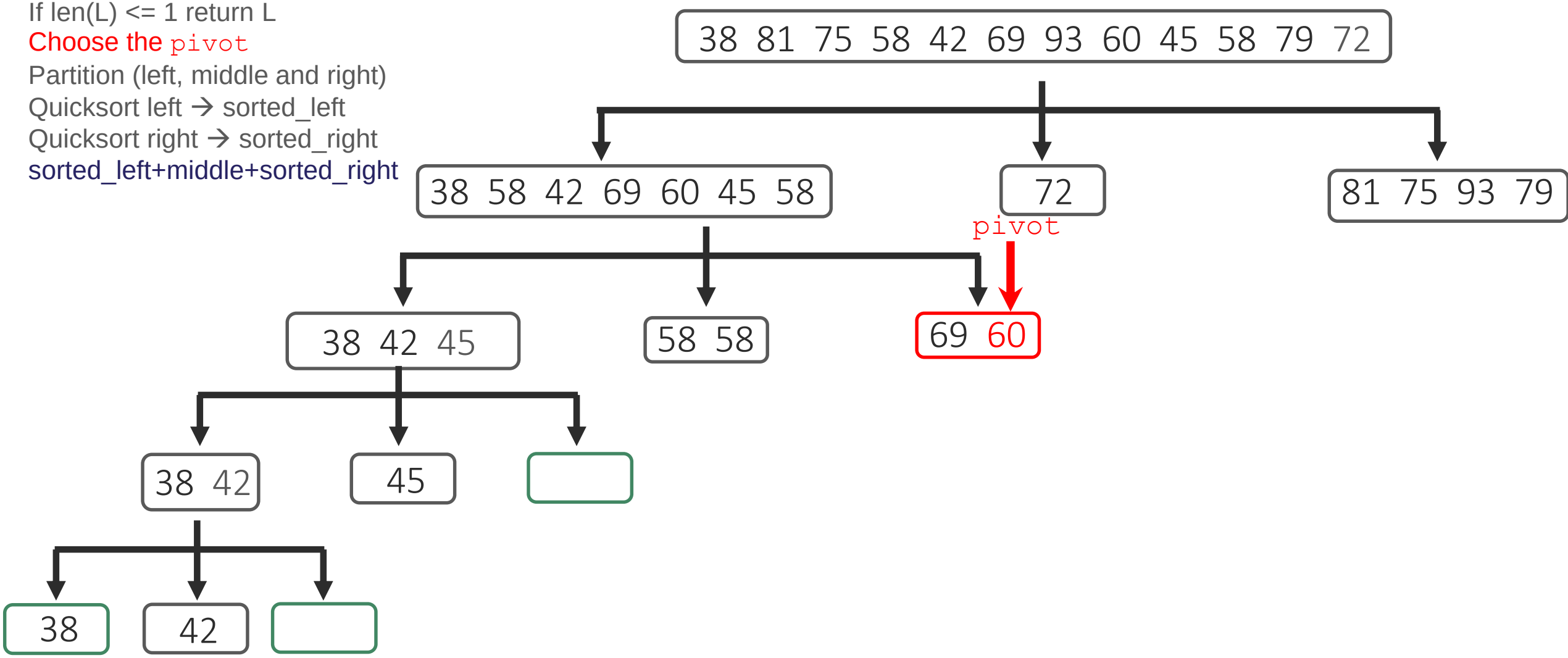


`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

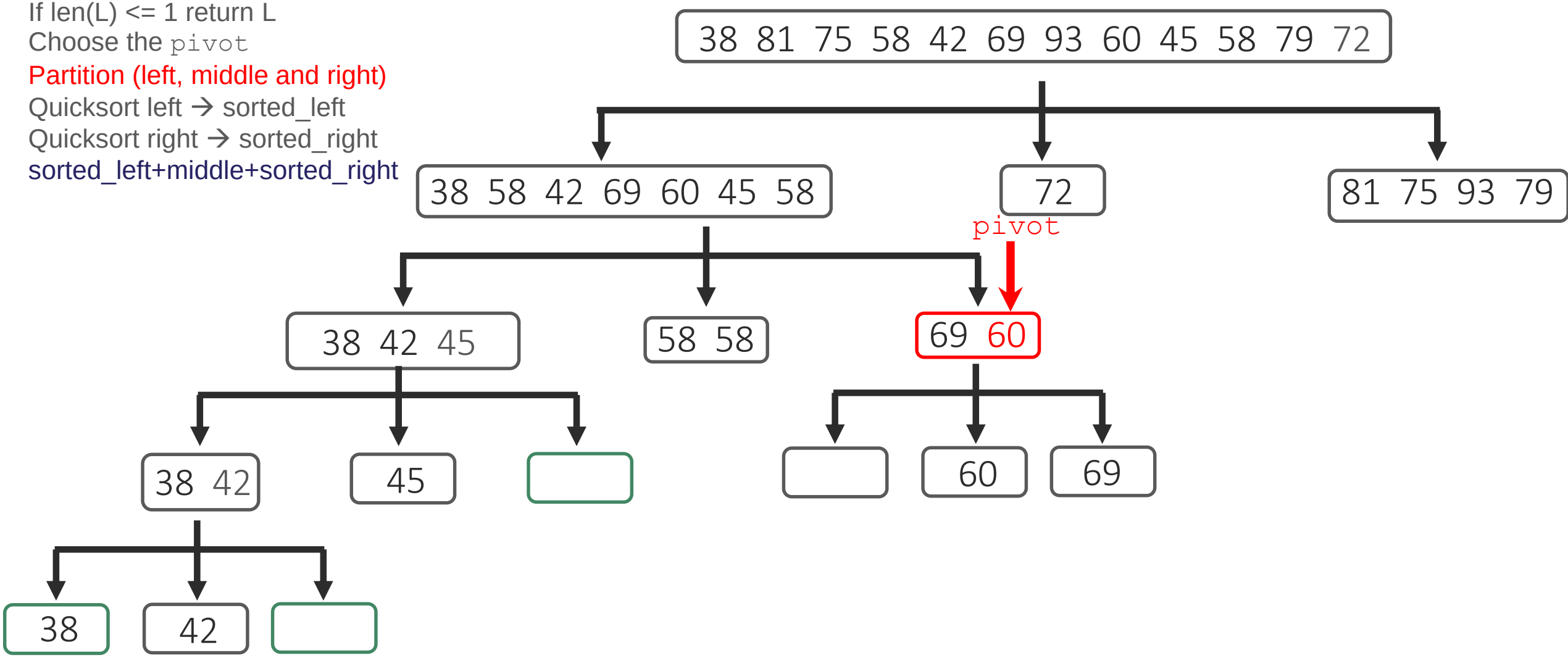


`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

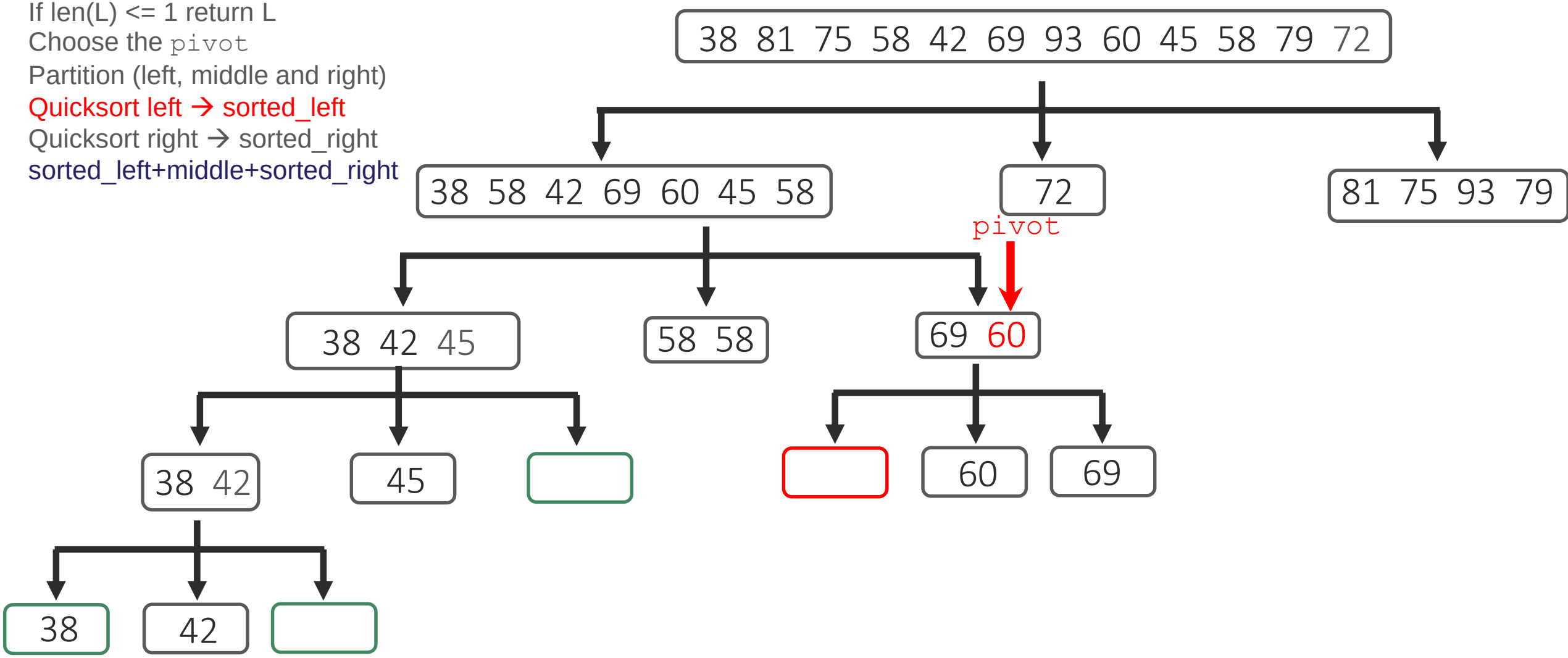


`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L

Choose the `pivot`

Partition (left, middle and right)

Quicksort left → sorted_left

Quicksort right → sorted_right

sorted_left+middle+sorted_right

`sorted_left:` []

`middle:` 60

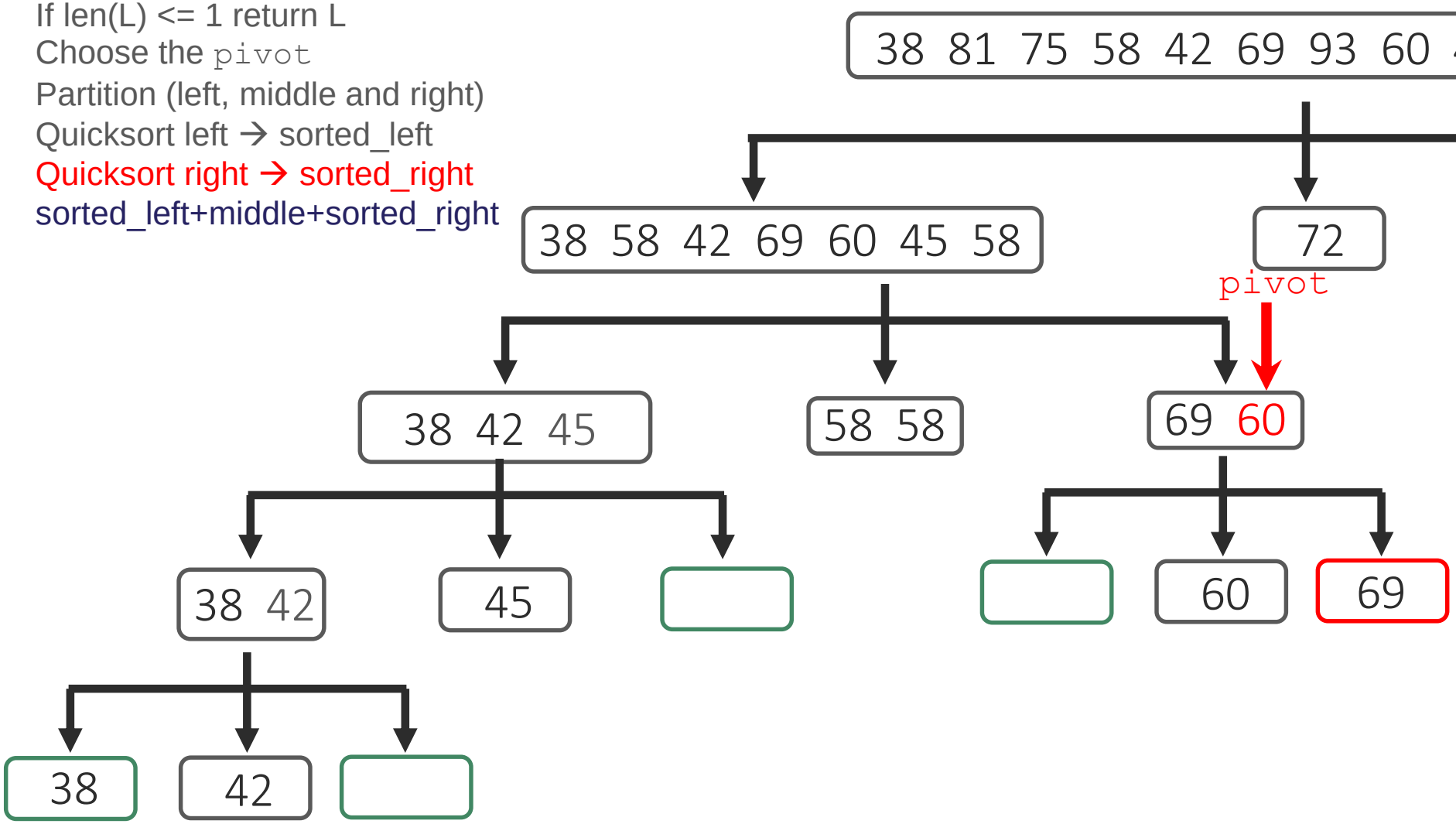
`sorted_right:`

`sorted_left:` 38 42 45

`middle:` 58 58

`sorted_right:`

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



`sorted_left:` []
`middle:` 60
`sorted_right:` 69

`sorted_left:` 38 42 45

`middle:` 58 58

`sorted_right:`

If len(L) <= 1 return L

Choose the `pivot`

Partition (left, middle and right)

Quicksort left → sorted_left

Quicksort right → sorted_right

sorted_left+middle+sorted_right

`sorted_left`: []

`middle`: 60

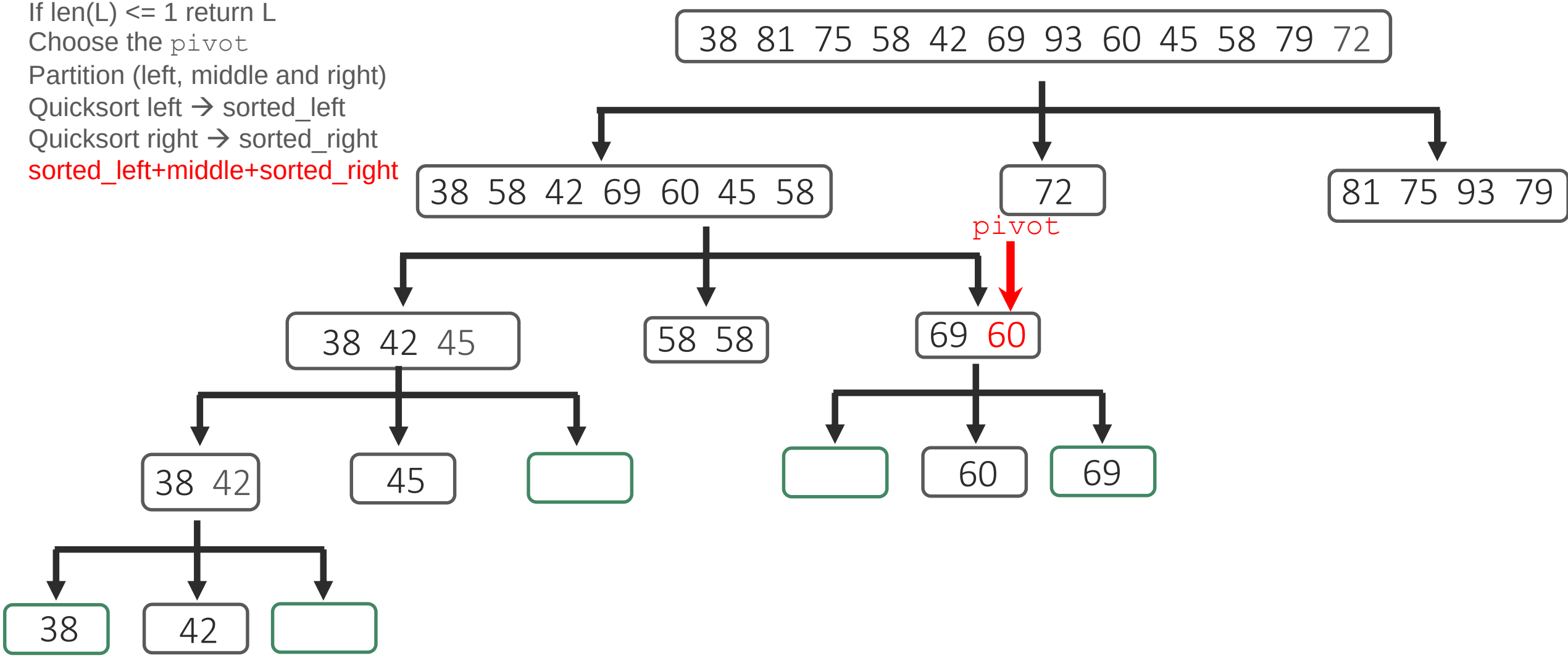
`sorted_right`: 69

`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

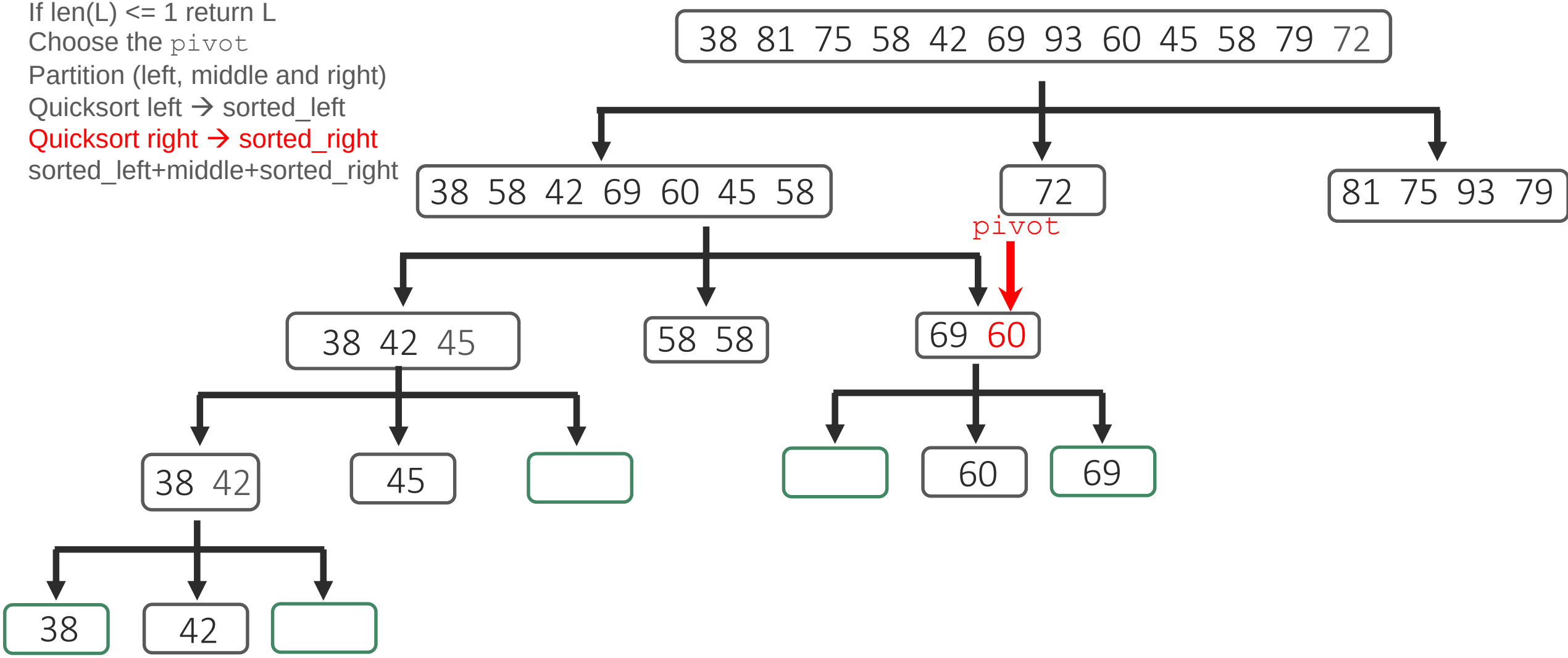


`sorted_left`: 38 42 45

`middle`: 58 58

`sorted_right`: 60 69

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right

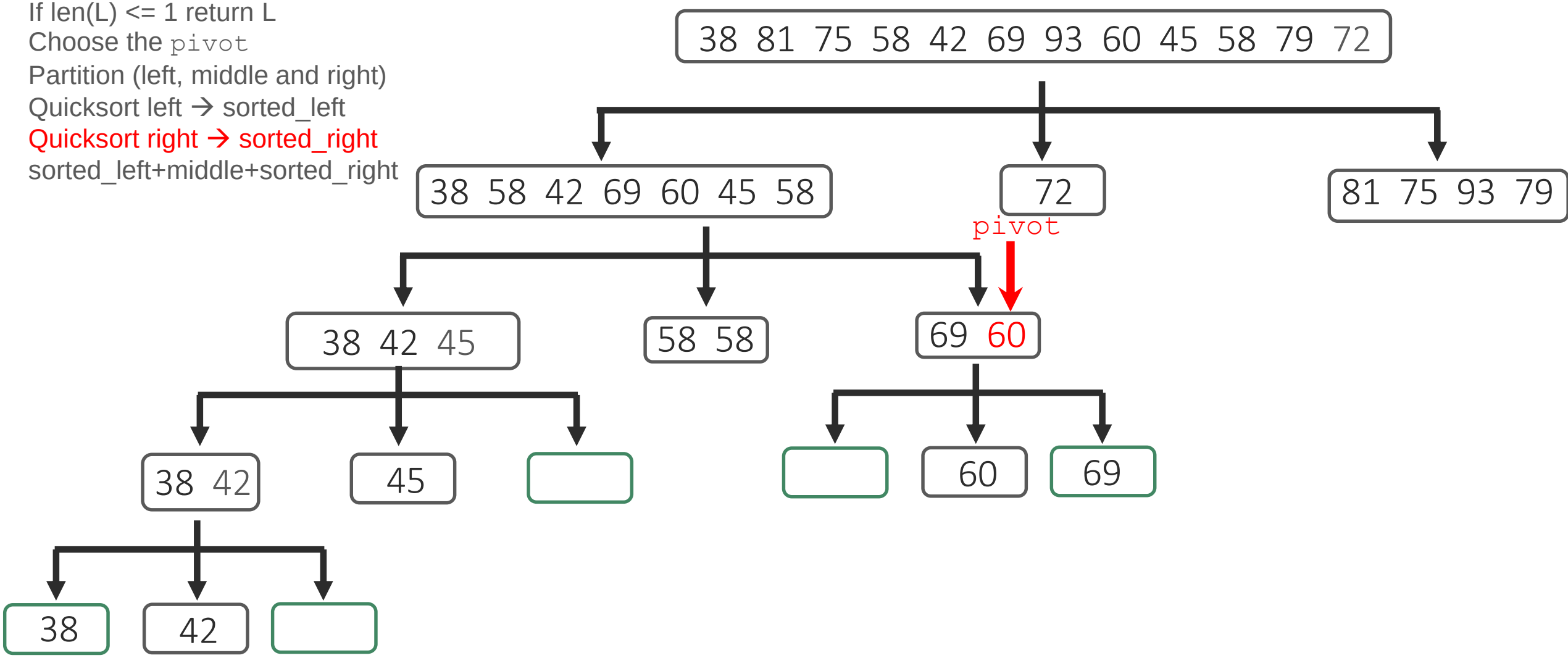


`sorted_left`: 38 42 45 58 58 60 69

`middle`: 72

`sorted_right`:

If len(L) <= 1 return L
Choose the `pivot`
Partition (left, middle and right)
Quicksort left → sorted_left
Quicksort right → sorted_right
sorted_left+middle+sorted_right



`sorted_left`: 38 42 45 58 58 60 69

`middle`: 72

`sorted_right`: