

## Sorting

*David Croft*

Introduction

Bubblesort

Stable sort

In-place

Selection sort

Other  
algorithms

Quicksort

Divide & Conquer

Comparing

Recap

# Sorting algorithms

David Croft

Coventry University

david.croft@coventry.ac.uk

November 25, 2015

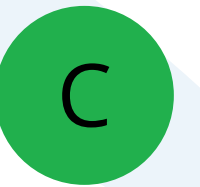
# Overview

- 1 Introduction
- 2 Bubblesort
  - Stable sort
  - In-place
- 3 Selection sort
- 4 Other algorithms
- 5 Quicksort
  - Divide & Conquer
- 6 Comparing
- 7 Recap

Sorting is one of the classic problems for learning algorithms.

- Requirement for everything.
- Obvious applications like sorting text, statistics (median calculations).
- Less obvious, sorting objects in games for FOV calculations.
- Route planning.

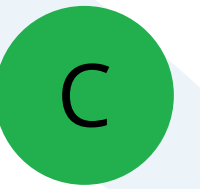
# Bubblesort



Very simple sort.

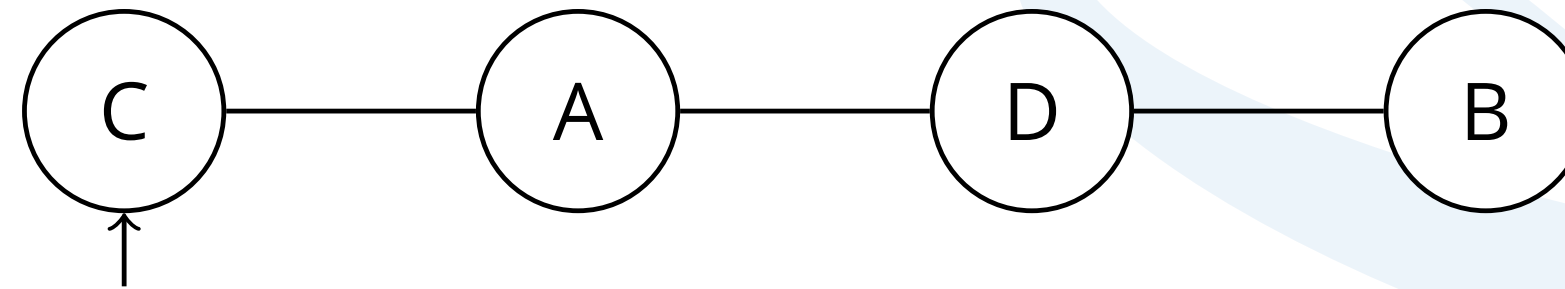
- Compares each item to the next in the sequence.
  - Swap items if in wrong order.

## Bubblesort

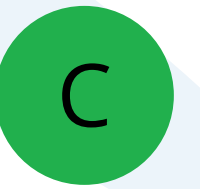


Very simple sort.

- Compares each item to the next in the sequence.
- Swap items if in wrong order.

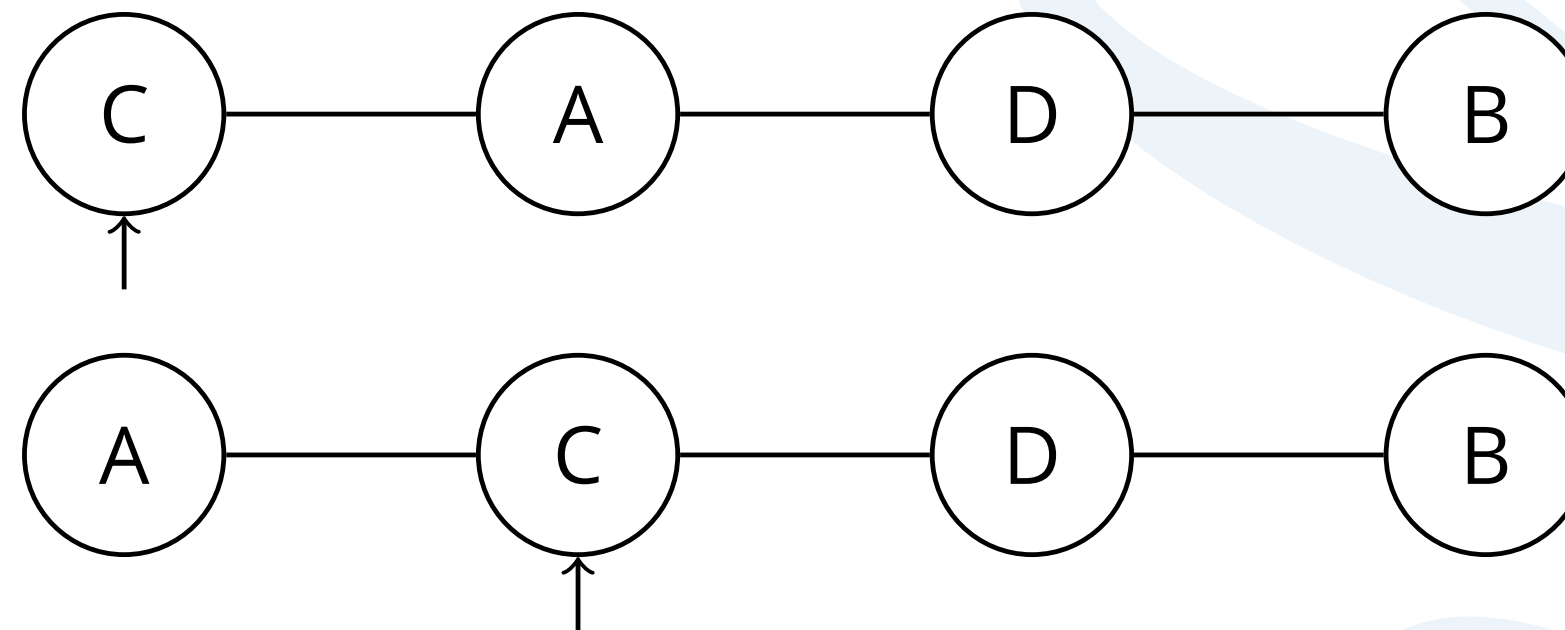


## Bubblesort

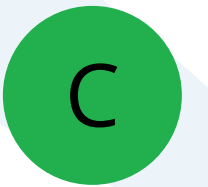


Very simple sort.

- Compares each item to the next in the sequence.
- Swap items if in wrong order.

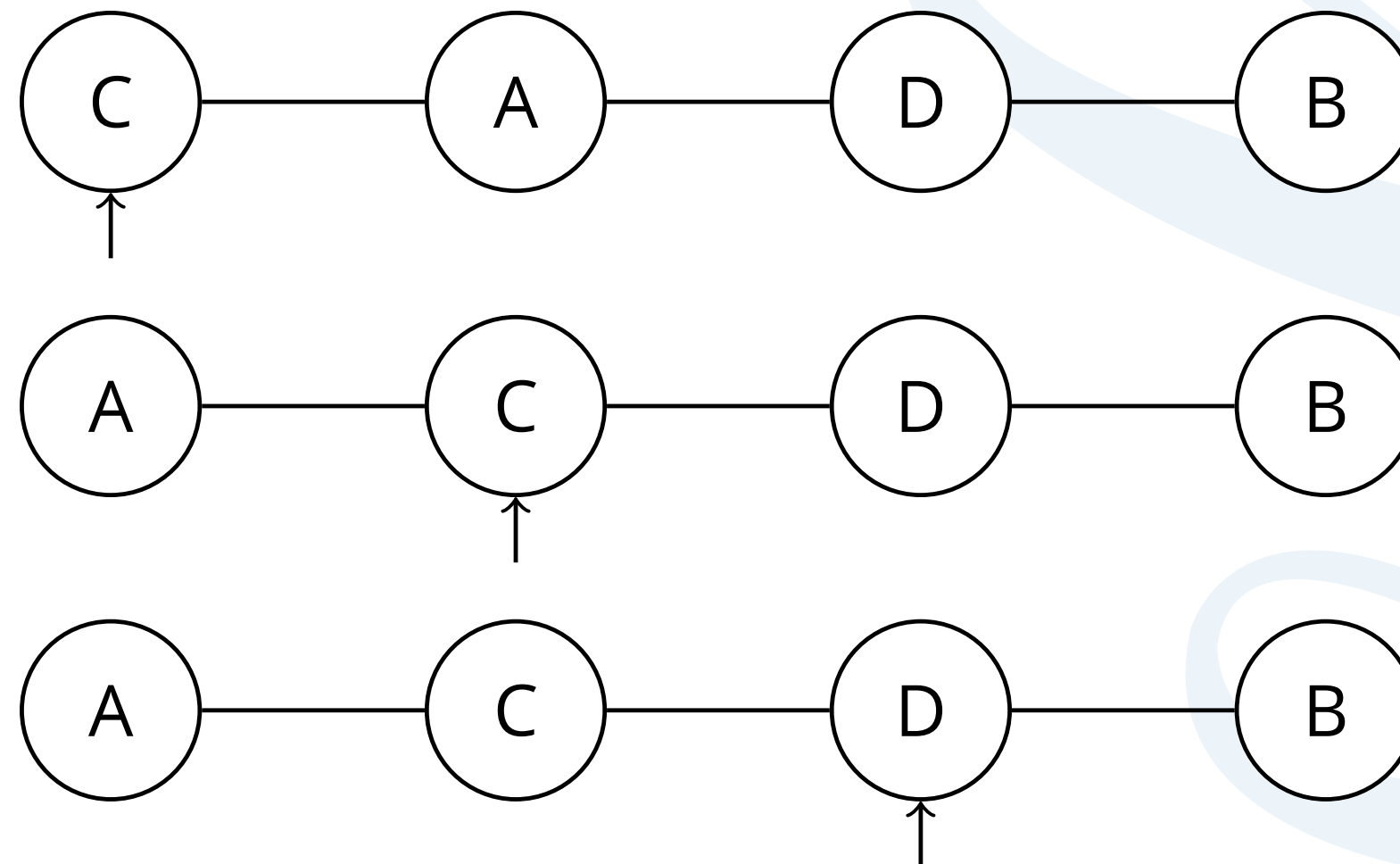


## Bubblesort



Very simple sort.

- Compares each item to the next in the sequence.
- Swap items if in wrong order.

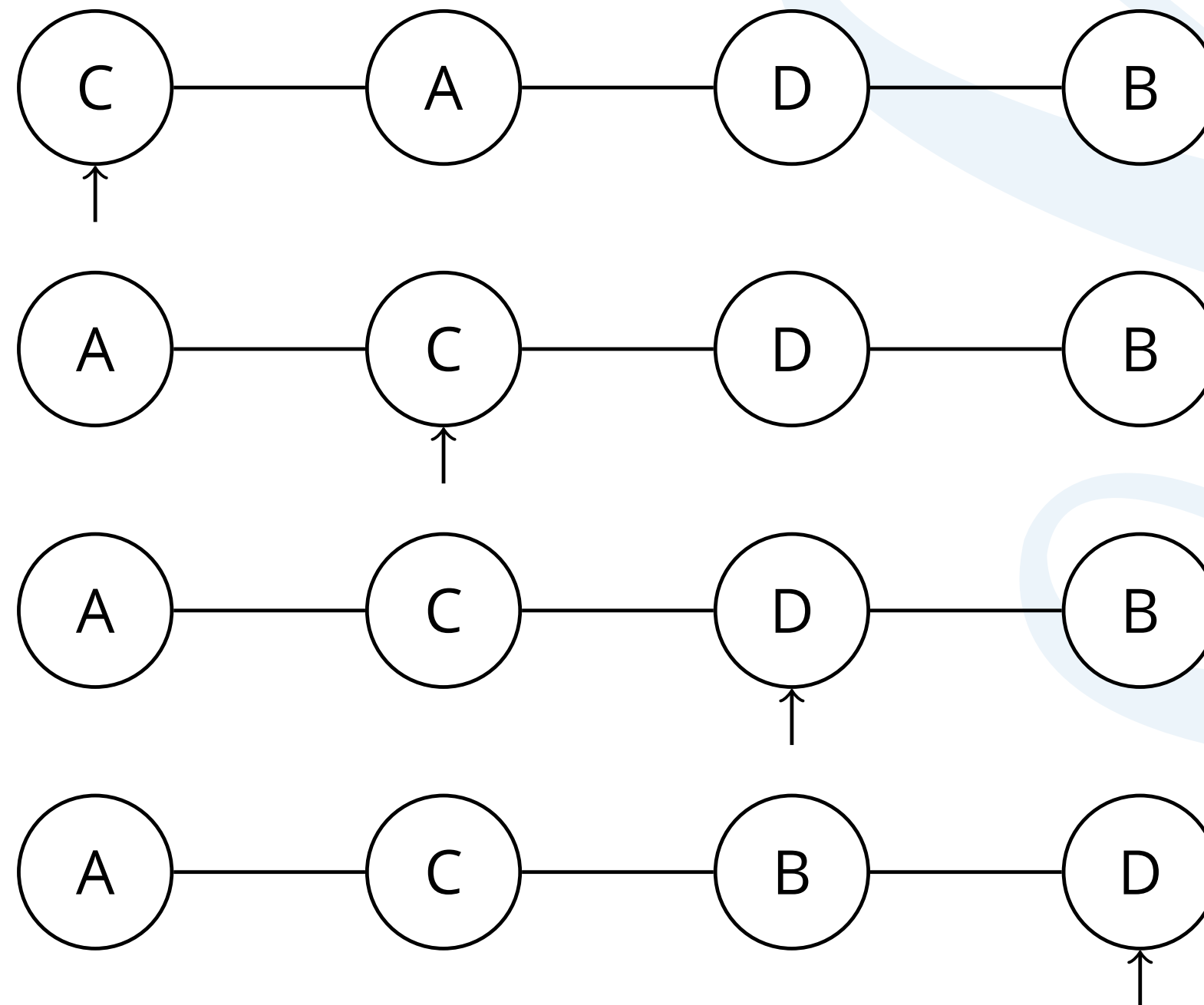


## Bubblesort

C

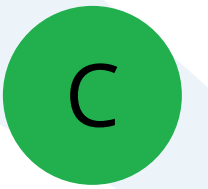
Very simple sort.

- Compares each item to the next in the sequence.
- Swap items if in wrong order.



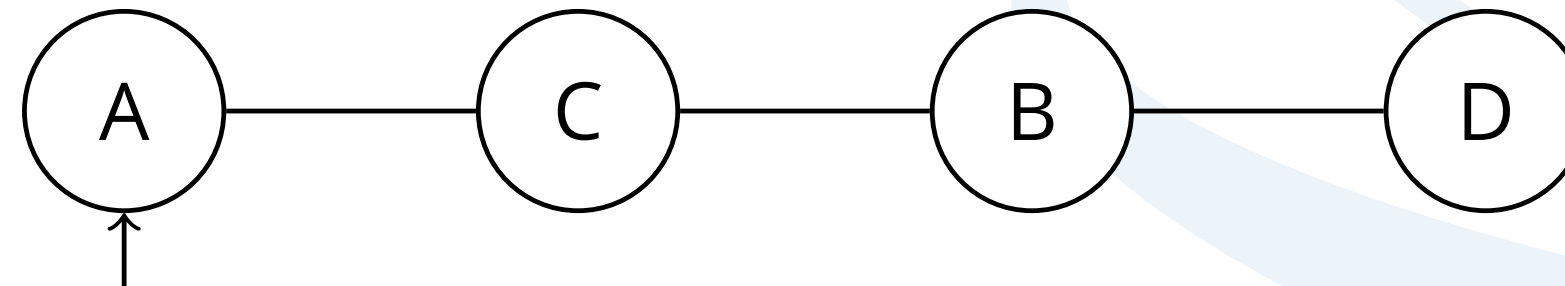


## Bubblesort

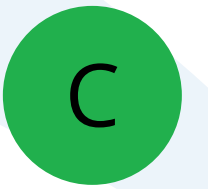


Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

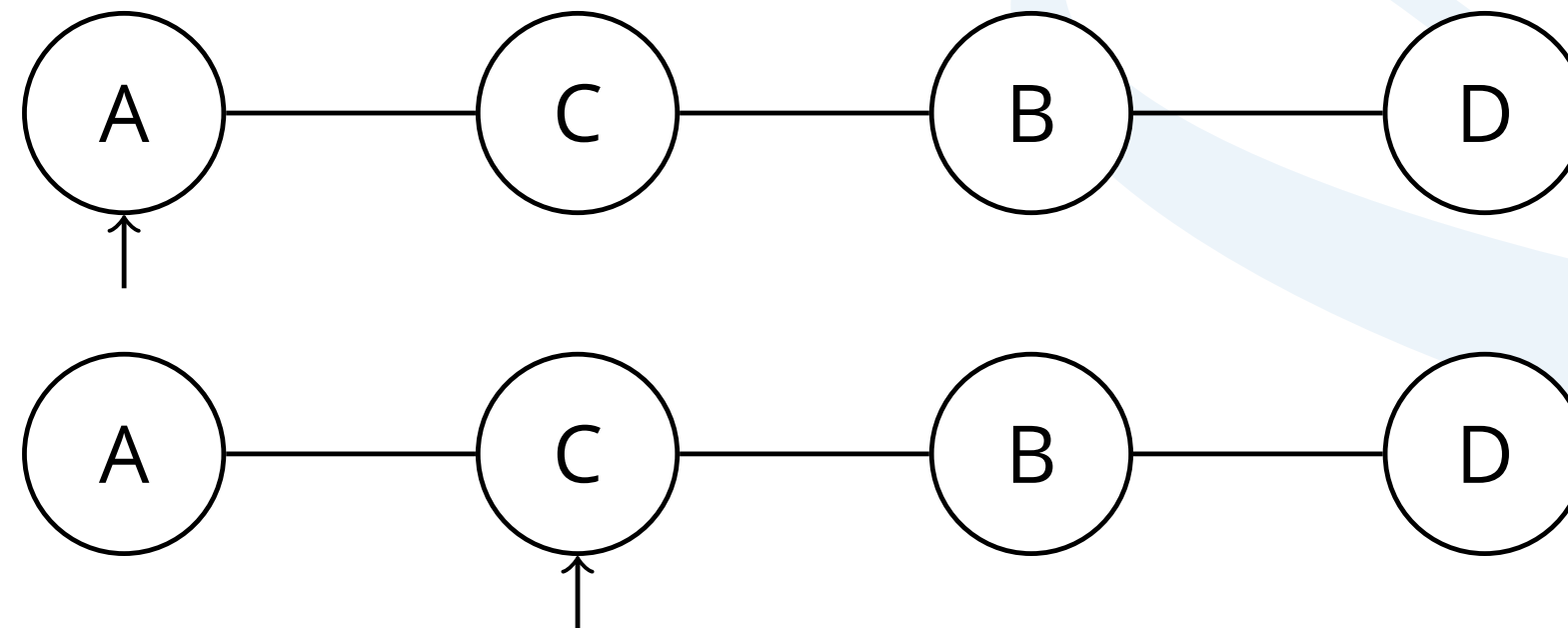


## Bubblesort

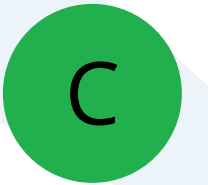


Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

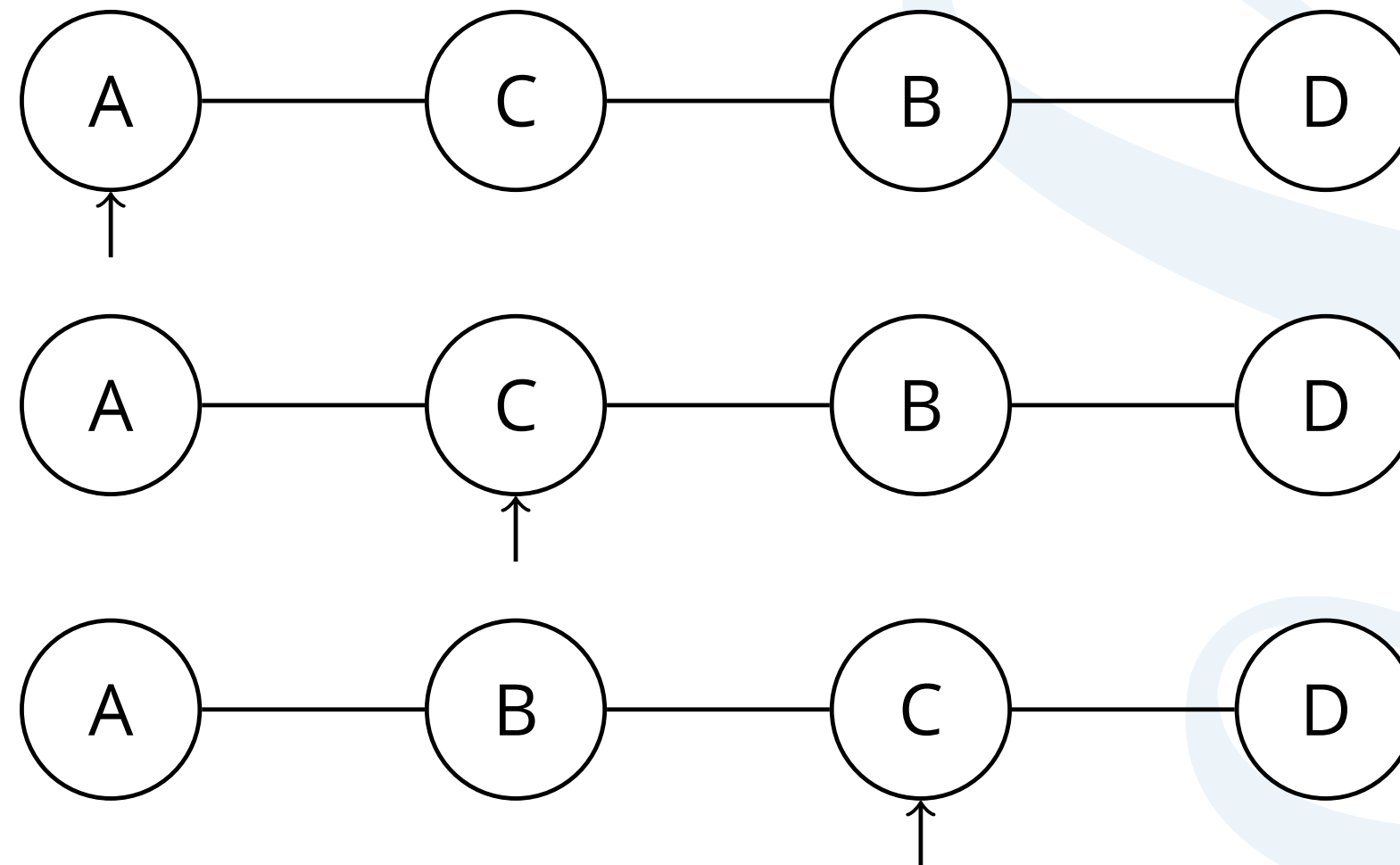


## Bubblesort

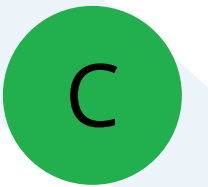


Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

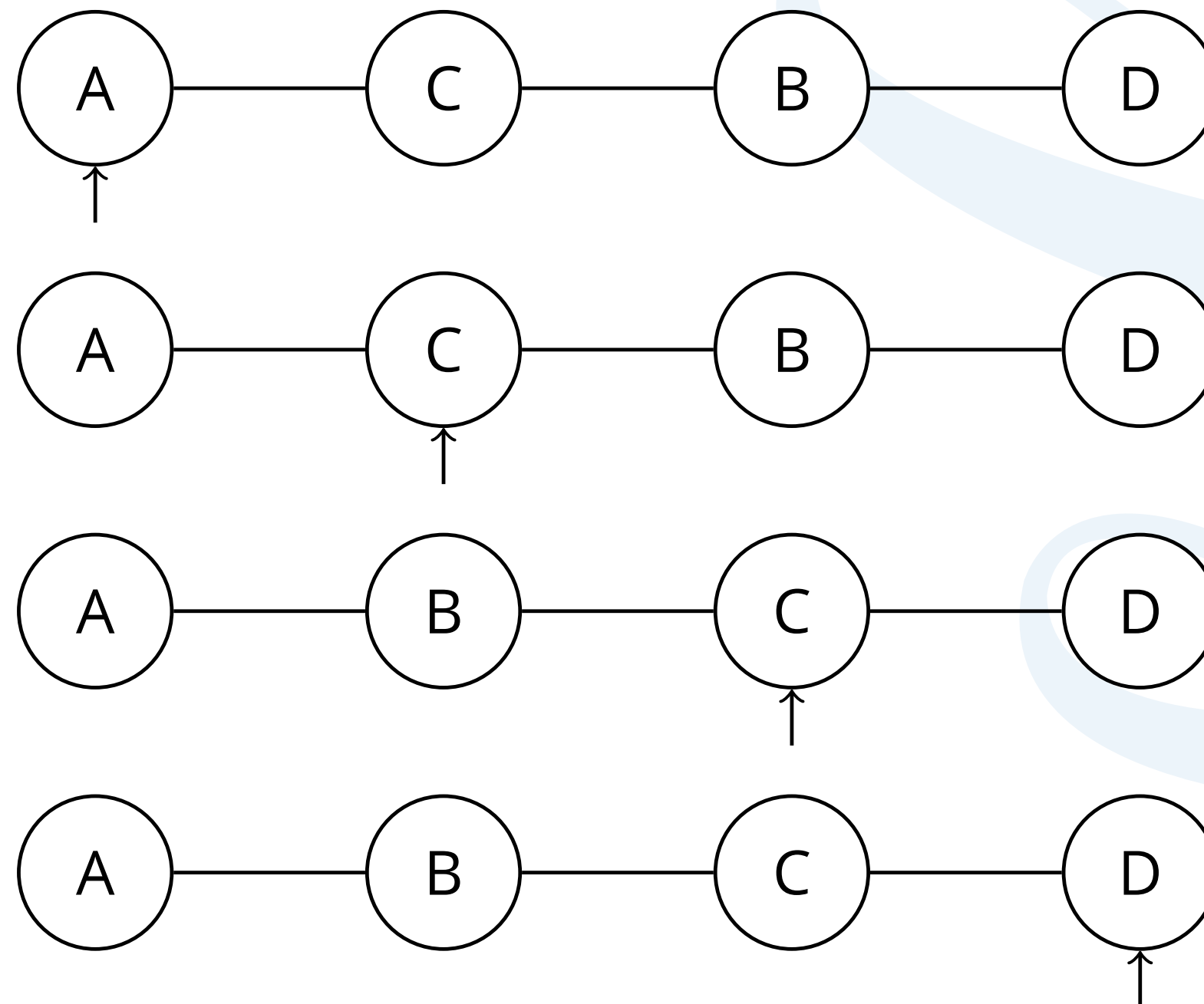


## Bubblesort



Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.



# Stable sort



Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

# Stable sort

1

Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

- Not important if e.g. sorting people by height.

# Stable sort

I

Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

- Not important if e.g. sorting people by height.
- Important if e.g. priority queues.
  - Imagine a queue in an emergency room.

# Stable sort

I

Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

- Not important if e.g. sorting people by height.
- Important if e.g. priority queues.
  - Imagine a queue in an emergency room.
  - Treat the most serious conditions first, sort people on how bad injury is.



Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

- Not important if e.g. sorting people by height.
- Important if e.g. priority queues.
  - Imagine a queue in an emergency room.
  - Treat the most serious conditions first, sort people on how bad injury is.
  - If many people have same injury then should be seen based on when entered queue.

Bubble sort is what's known as an stable in-place sort.

Stable meaning that equivalent elements do not change their relative orders.

- Not important if e.g. sorting people by height.
- Important if e.g. priority queues.
  - Imagine a queue in an emergency room.
  - Treat the most serious conditions first, sort people on how bad injury is.
  - If many people have same injury then should be seen based on when entered queue.

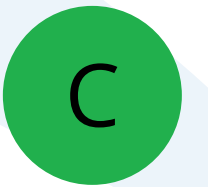
With unstable sorting algorithm the relative orders of equivalent elements can be changed.



In-place meaning that it only needs a small amount of additional memory in order to work.

- More memory efficient than the alternative.
- Can be important if...
  - ...dealing with large amounts of data.
  - ...have limited resources (i.e. embedded systems).
- Bubble sort only needs a few extra variables to swap the elements and to step through the sequence.

# Bubblesort



One of the simplest sorting algorithms.

- Explained here to introduce you to sorting concepts.
  - In-place, stable.

# Bubblesort

C

One of the simplest sorting algorithms.

- Explained here to introduce you to sorting concepts.
  - In-place, stable.
- Is rubbish.

# Bubblesort

C

One of the simplest sorting algorithms.

- Explained here to introduce you to sorting concepts.
  - In-place, stable.
- Is rubbish.
  - Horrible performance, average is  $O(n^2)$ .

One of the simplest sorting algorithms.

- Explained here to introduce you to sorting concepts.
  - In-place, stable.
- Is rubbish.
  - Horrible performance, average is  $O(n^2)$ .
  - But best case is only  $O(n)$ .



The time taken to sort a sequence depends on:

- The starting order of the sequence.

For example, Bubblesorting a 100 elements:

So sorting algorithms have 3  $O()$  values.





The time taken to sort a sequence depends on:

- The starting order of the sequence.

For example, Bubblesorting a 100 elements:

- Best case, are already sorted.
  - Iterate over sequence once.
  - 100 comparisons.

So sorting algorithms have 3  $O()$  values.

The time taken to sort a sequence depends on:

- The starting order of the sequence.

For example, Bubblesorting a 100 elements:

- Best case, are already sorted.
  - Iterate over sequence once.
  - 100 comparisons.
- Worst case, in reverse order.
  - Iterate over sequence 100 times.
  - 10,000 comparisons.

So sorting algorithms have 3  $O()$  values.



The time taken to sort a sequence depends on:

- The starting order of the sequence.

For example, Bubblesorting a 100 elements:

- Best case, are already sorted.
  - Iterate over sequence once.
  - 100 comparisons.
- Worst case, in reverse order.
  - Iterate over sequence 100 times.
  - 10,000 comparisons.
- Average case, random order.
  - Somewhere in between.

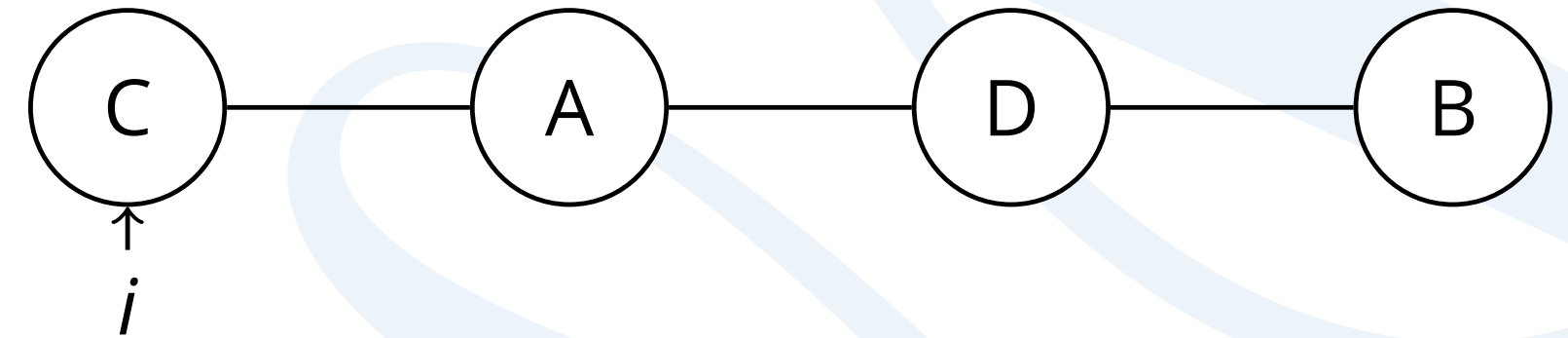
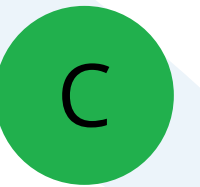
So sorting algorithms have 3  $O()$  values.

# Selection sort

C

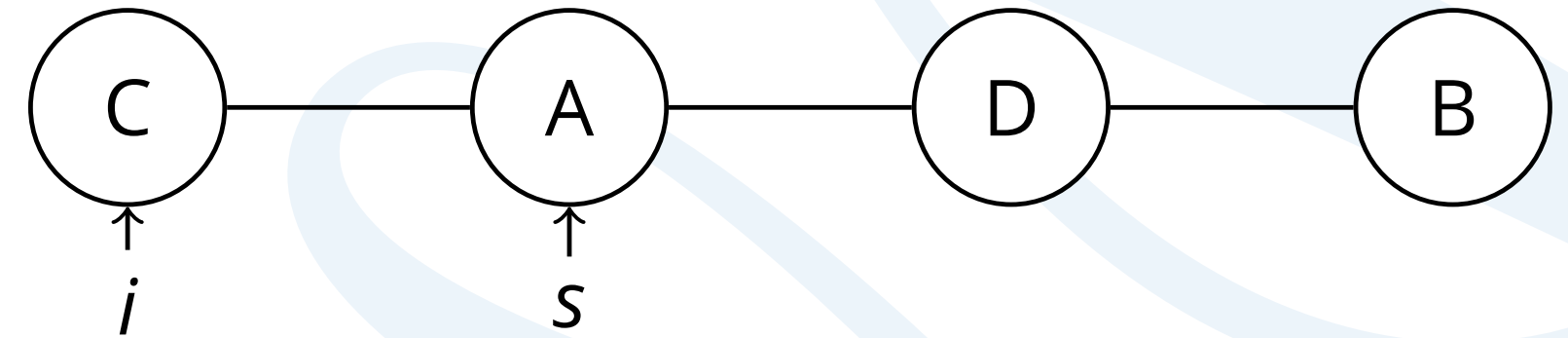
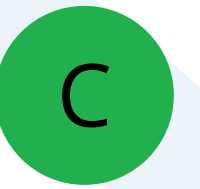
- Divides sequence into sorted and unsorted regions.
  - Not stable.
  - In place.
- 1 Iterate over sequence.
  - 2 For each element search the remaining elements on its right for the smallest value.
  - 3 Swap smallest element with current element.

## Selection sort II



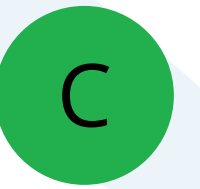
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

## Selection sort II

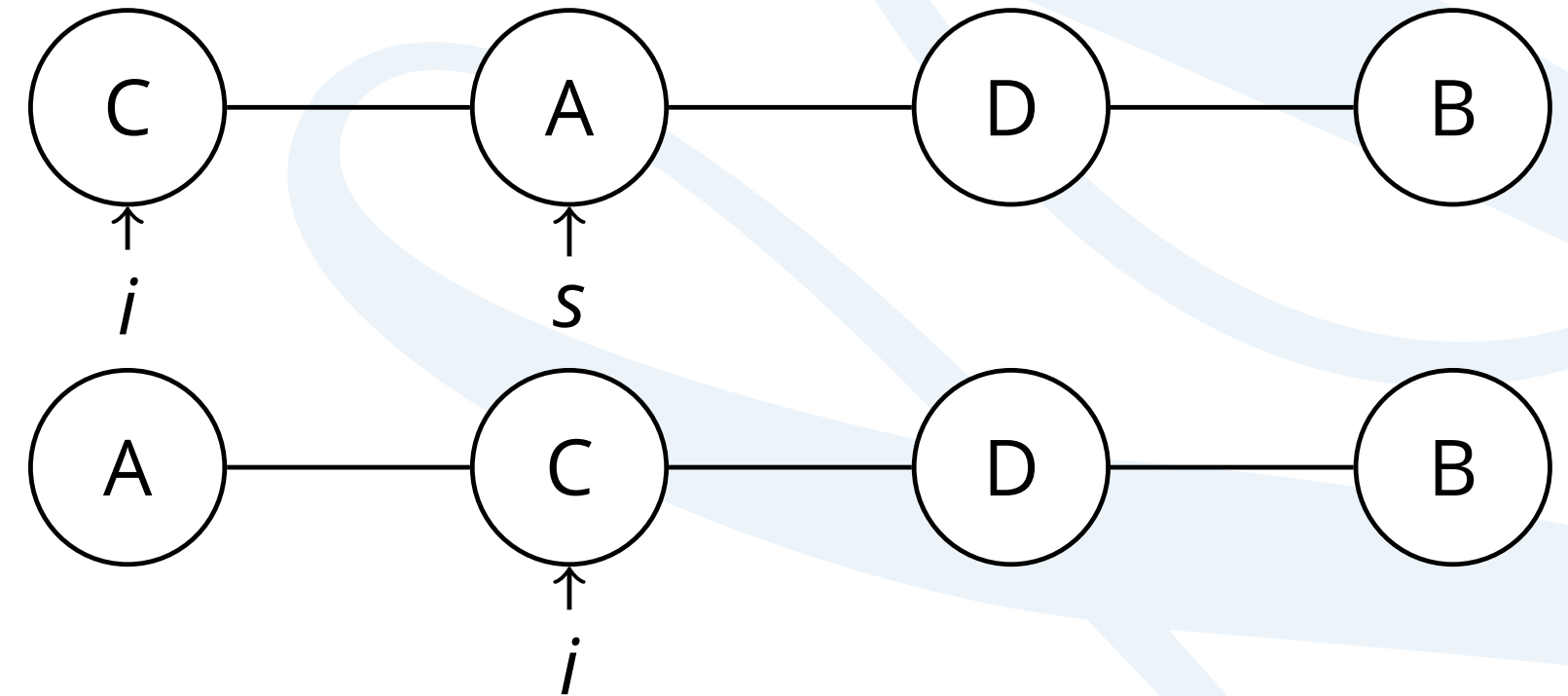


- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

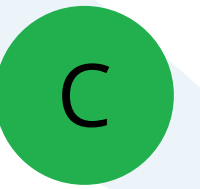
## Selection sort II



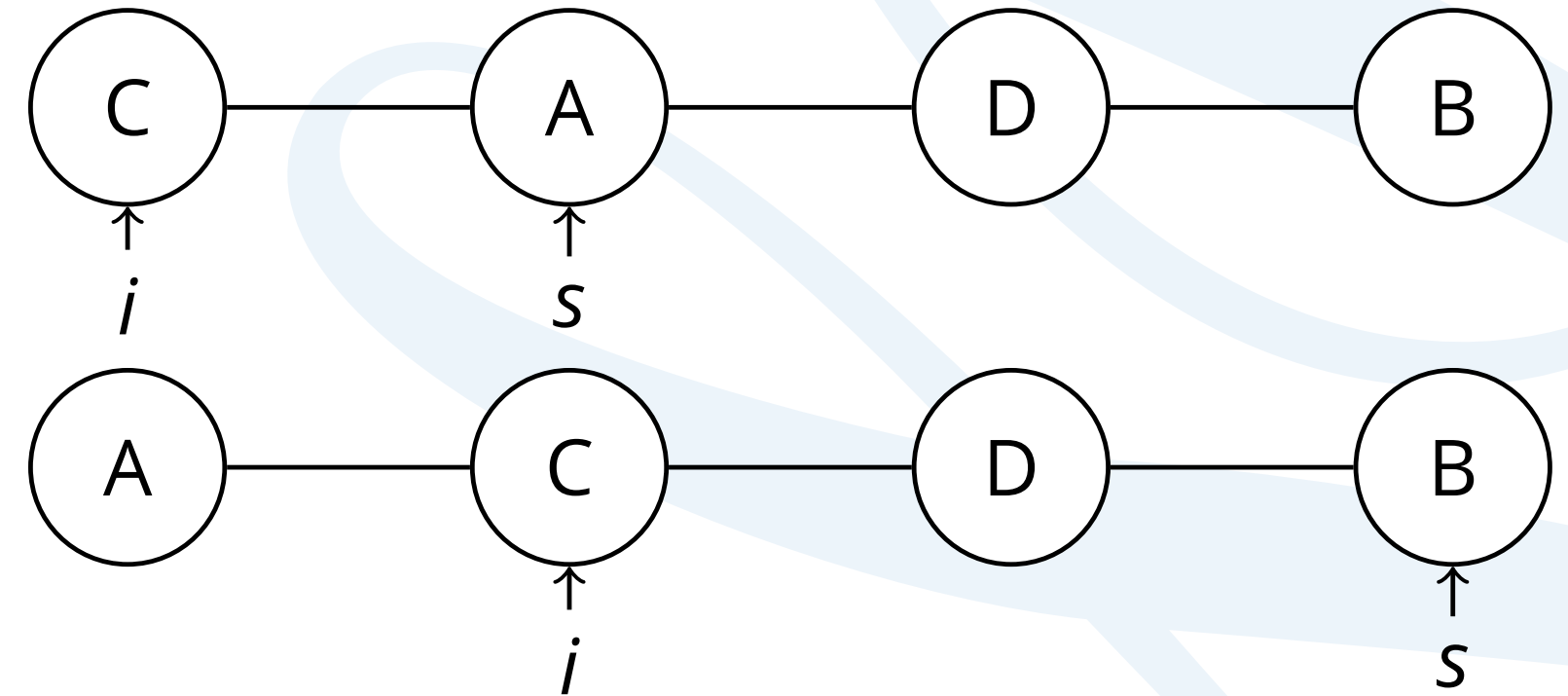
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



## Selection sort II

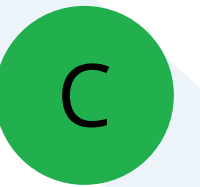


- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

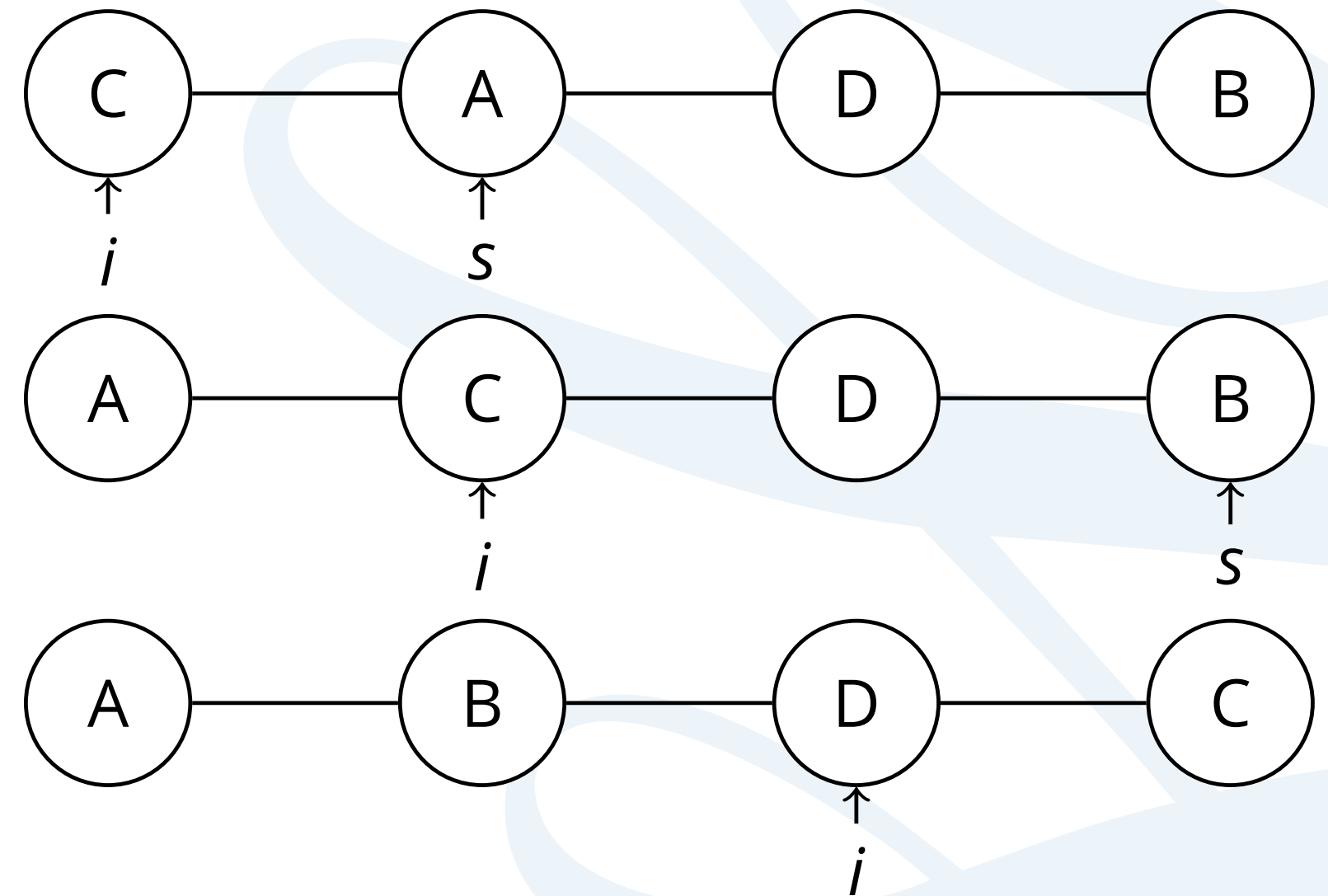




## Selection sort II

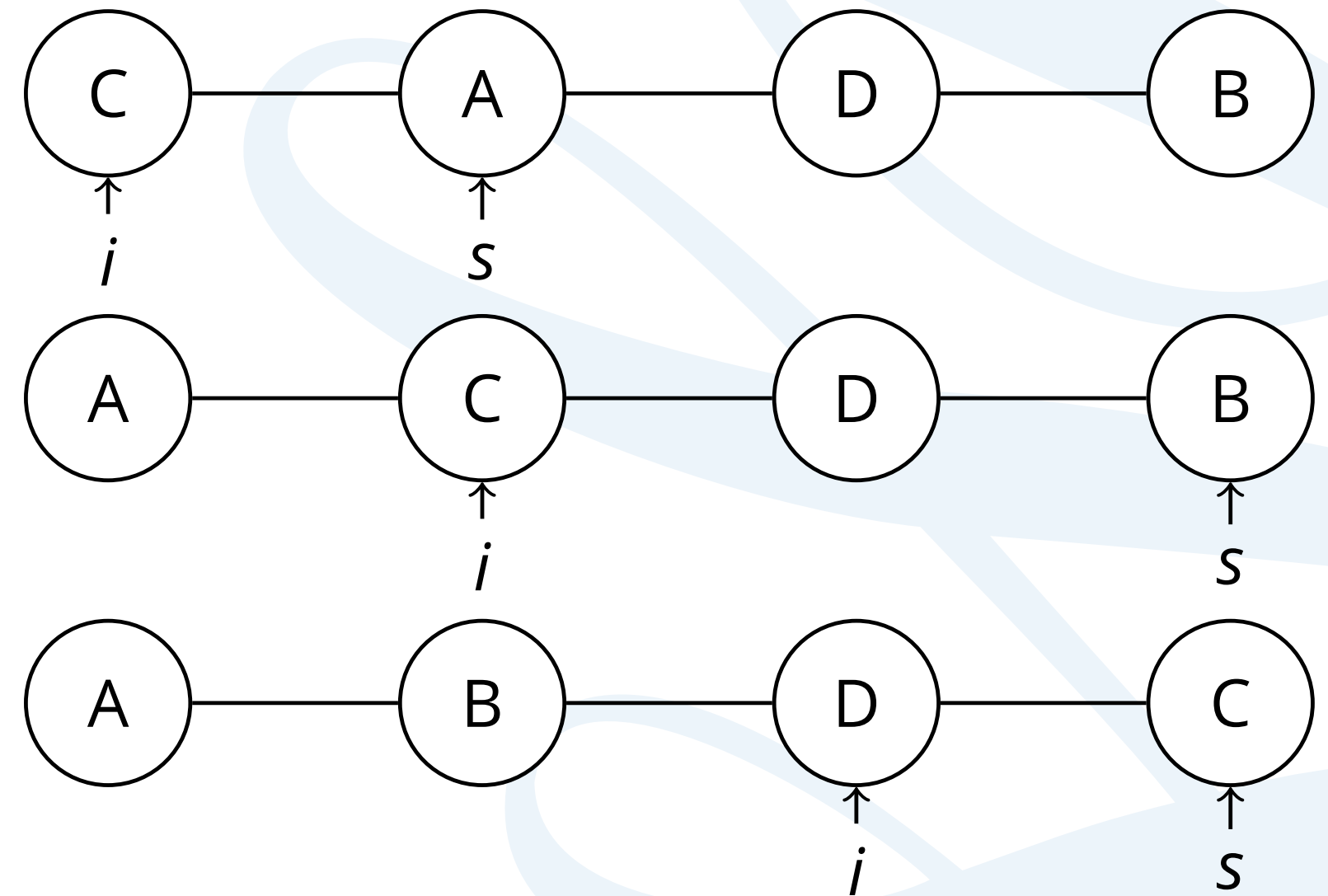
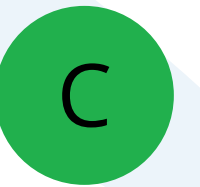


- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



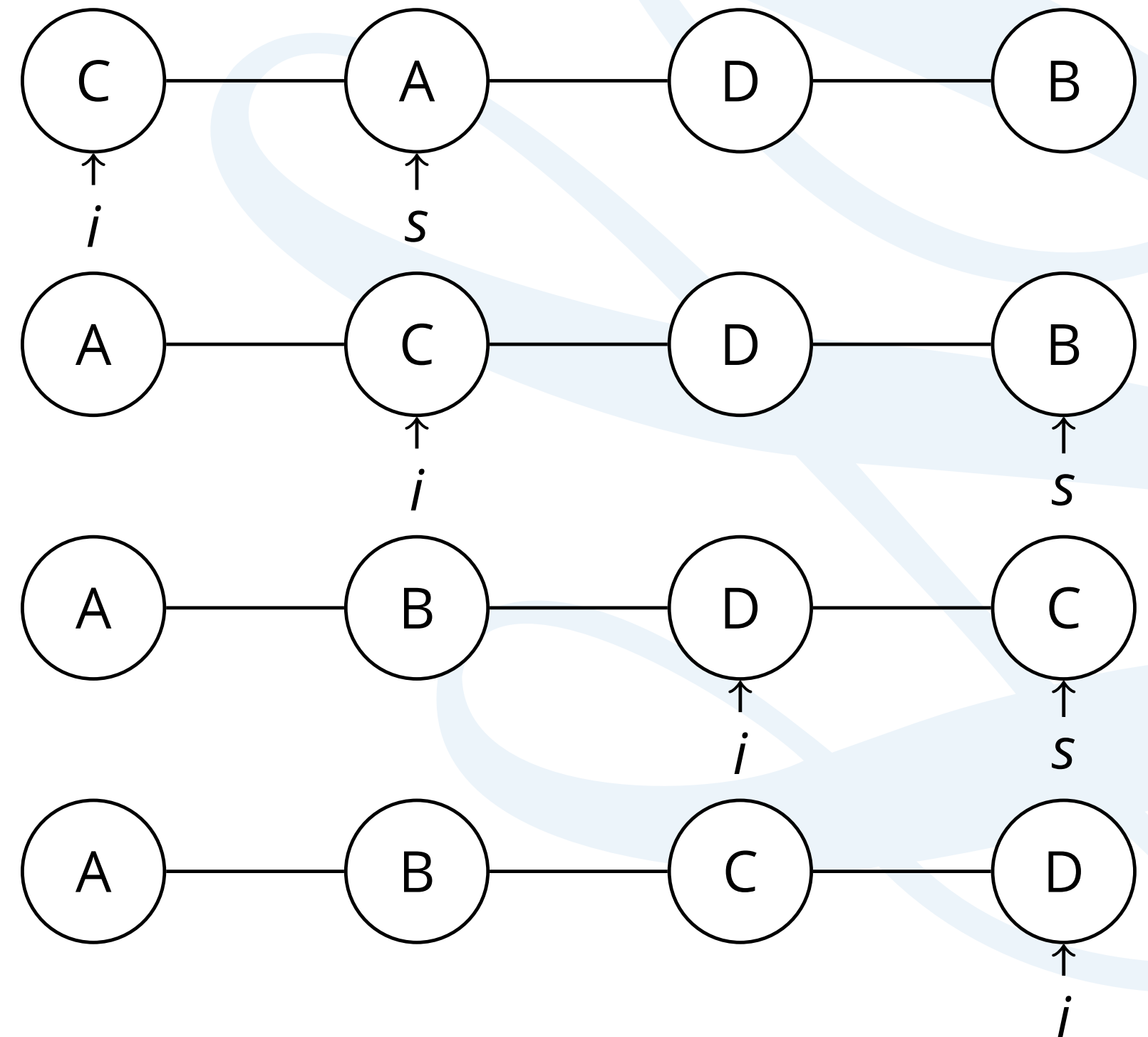
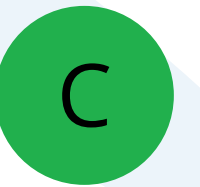
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

## Selection sort II



- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

## Selection sort II



$O()$ 

A

Bubblesort is  $O(n^2)$ .

Selection sort is  $O(n^2)$ .

- Selection sort is generally faster than bubble.
  - But have same  $O()$  complexity.
  - WTF?
- $O()$  notation describes how an algorithm will grow.
- Not good at absolute performances.
- Selection sort typically does fewer comparisons and swaps than bubblesort.
  - Therefore faster.

# Sorting Algorithms

Many sorting algorithms

- Different trade-offs, performances. <https://www.youtube.com/watch?v=ZZuD6iUe3Pc>
- Some are just jokes.

1 Bead

2 Bogo

3 Bubble

4 Circle

5 Cocktail

6 Comb

7 Counting

8 Cycle

9 Gnome

10 Heap

11 Insert

12 Merge

13 Pancake

14 Patience

15 Permutation

16 Quick

17 Radix

18 Selection

19 Shell

20 Sleep

21 Stooge

22 Strand

23 Tree

# Quicksort

C

Neither bubble or selection sort are very good.

- Simple algorithms but slow.
- Not used in real life.

One of the fastest sorting algorithms.

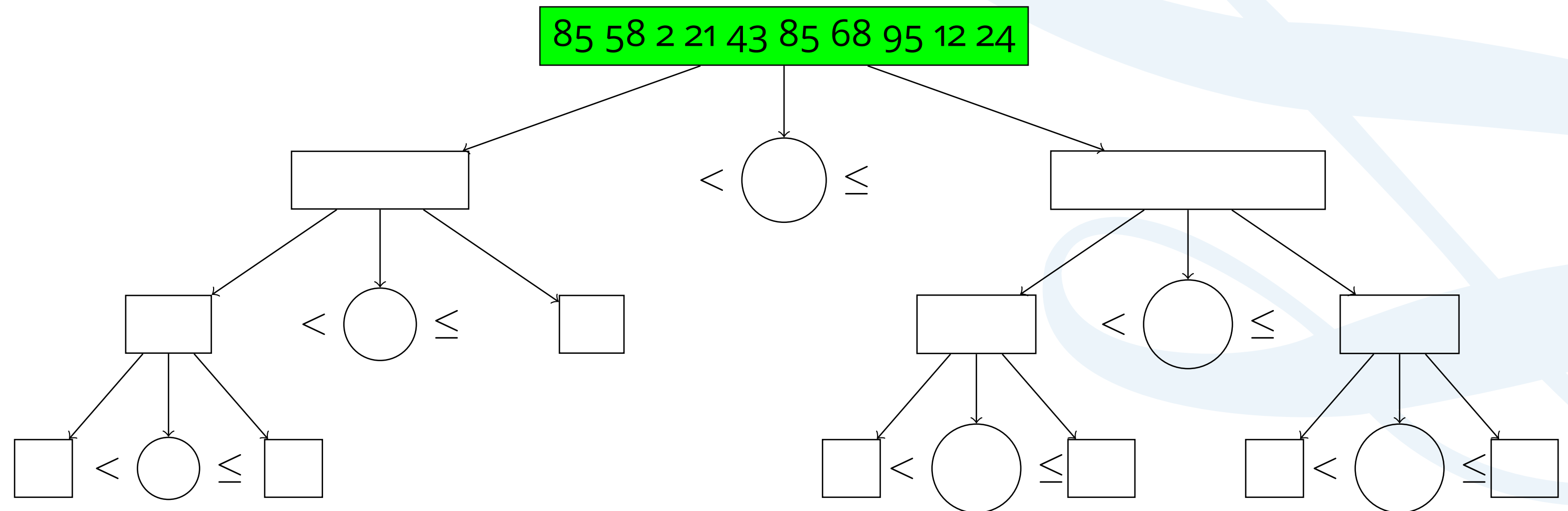
- Used in real life.
- Recursively breaks the sequence in half.
  - Divide & Conquer.

# Quicksort II

I

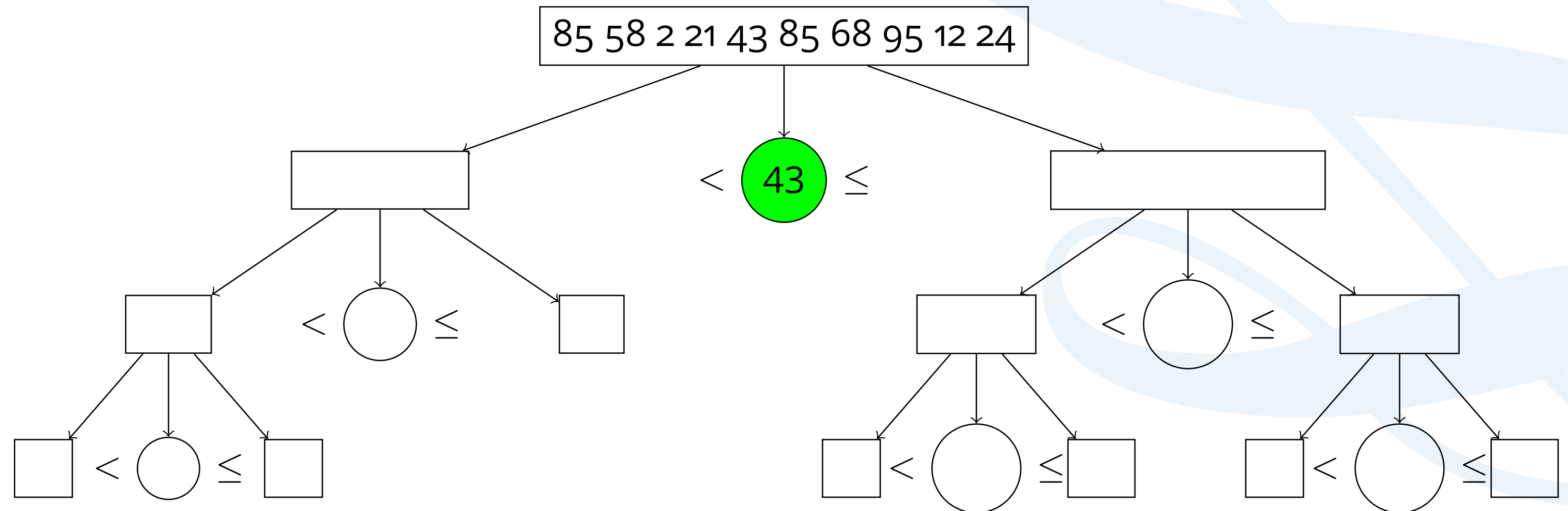
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

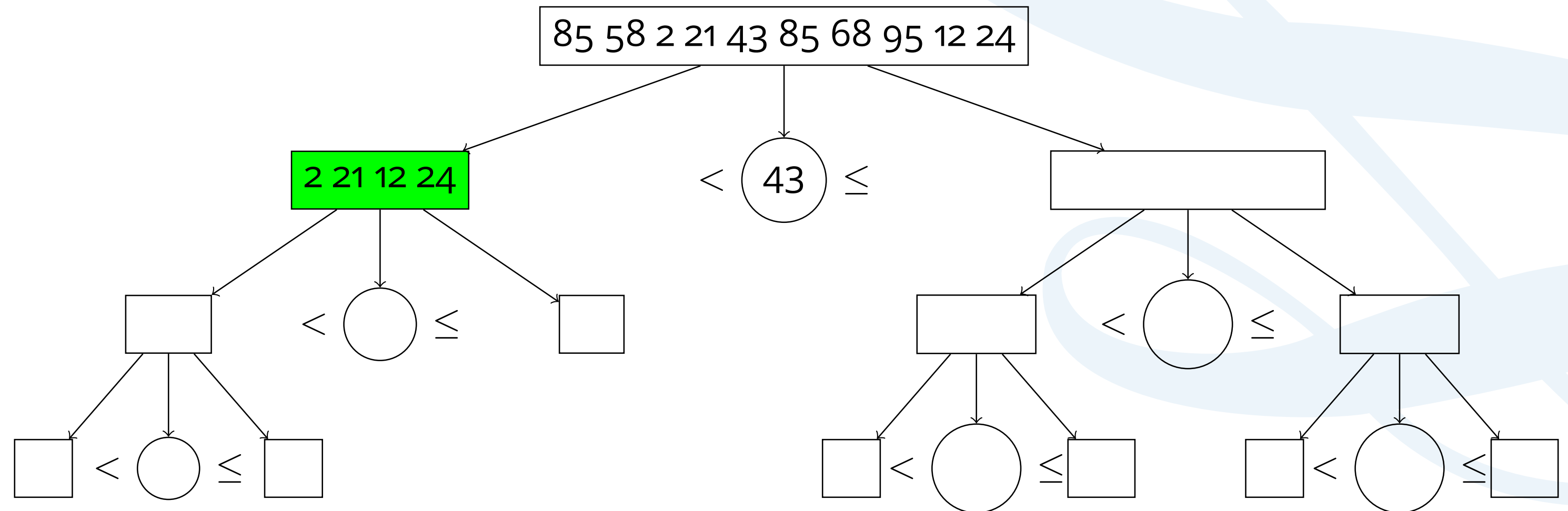




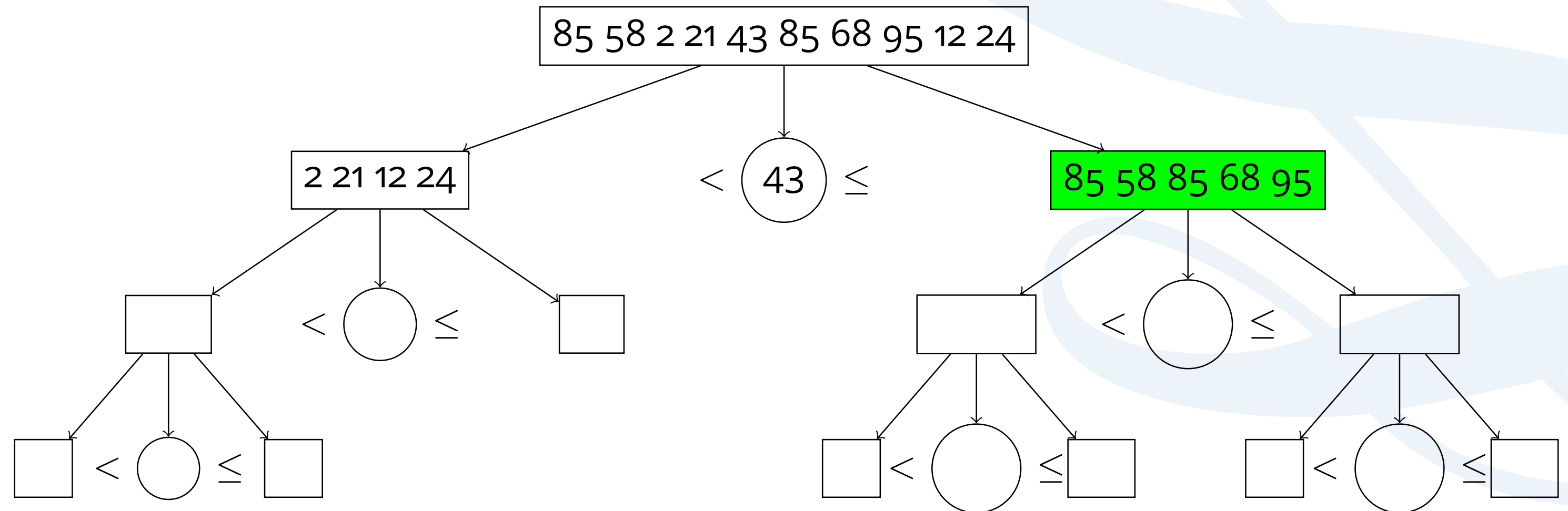
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



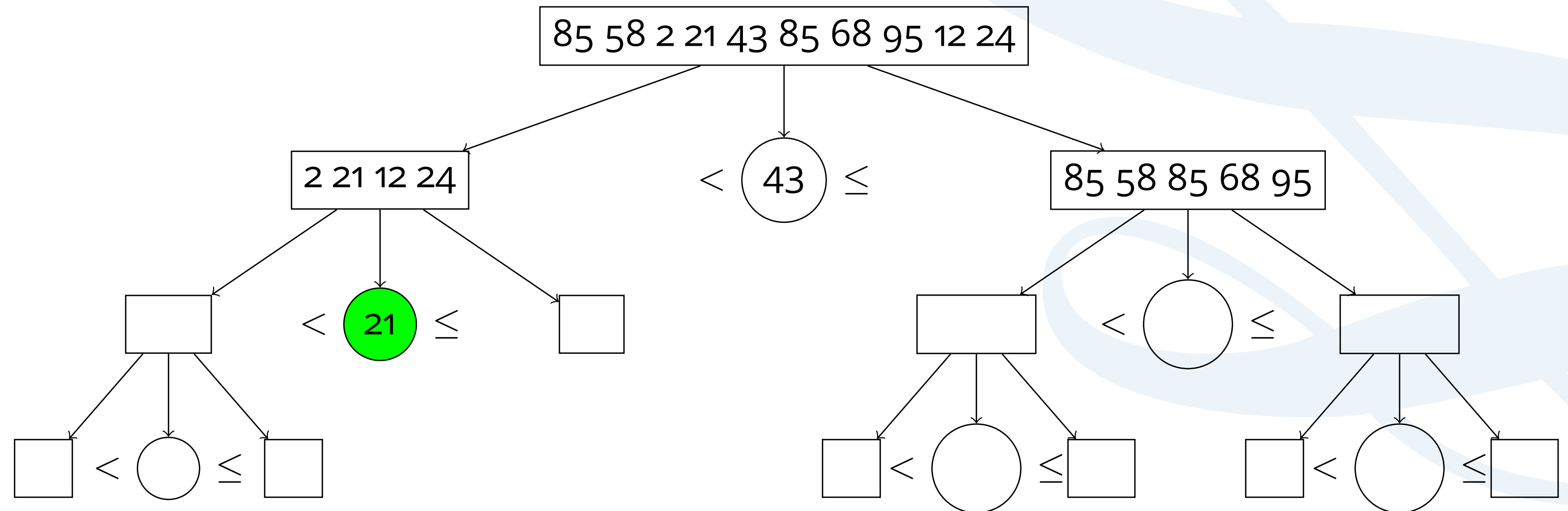
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



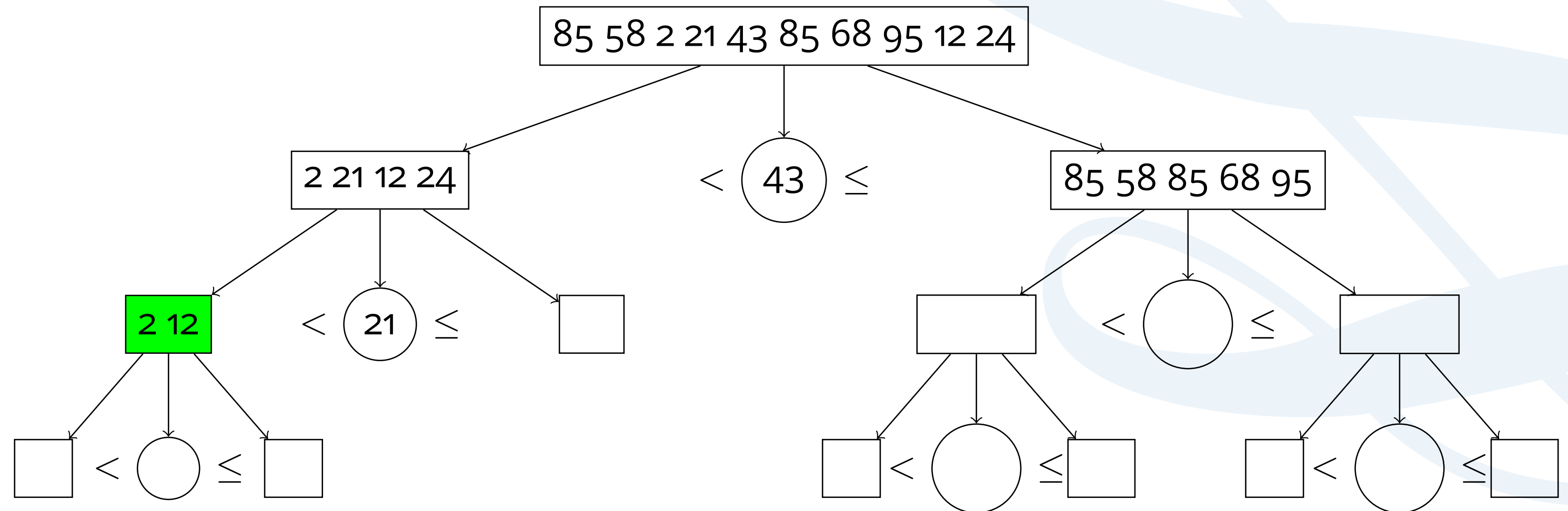
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



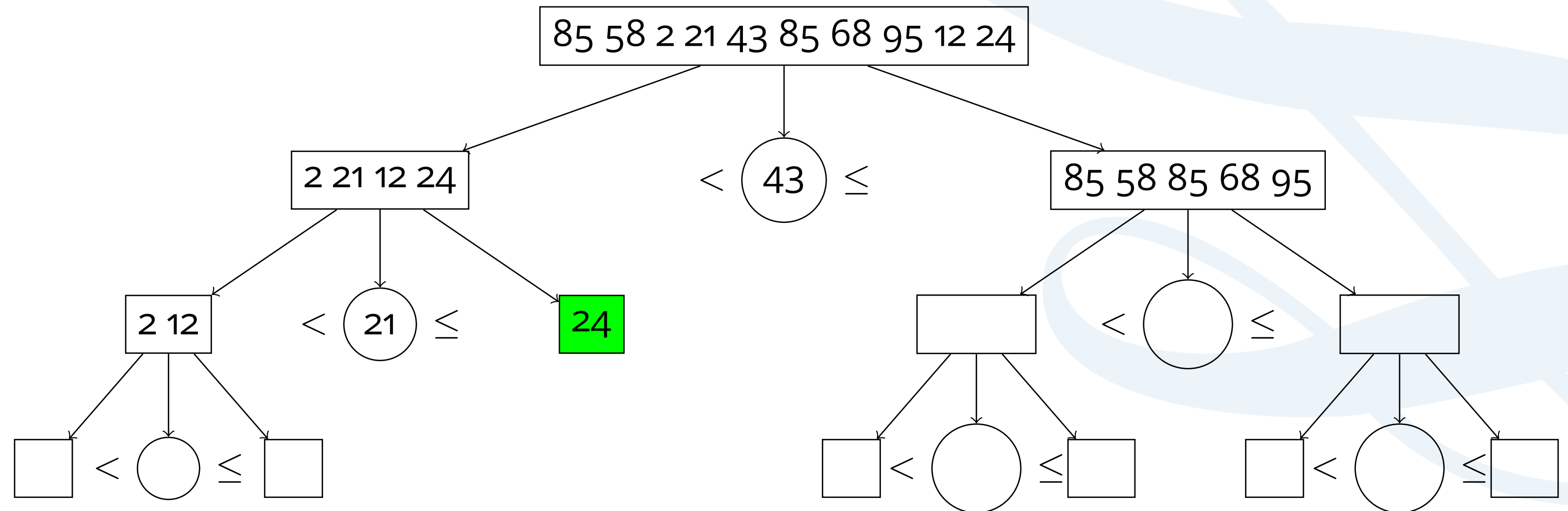
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



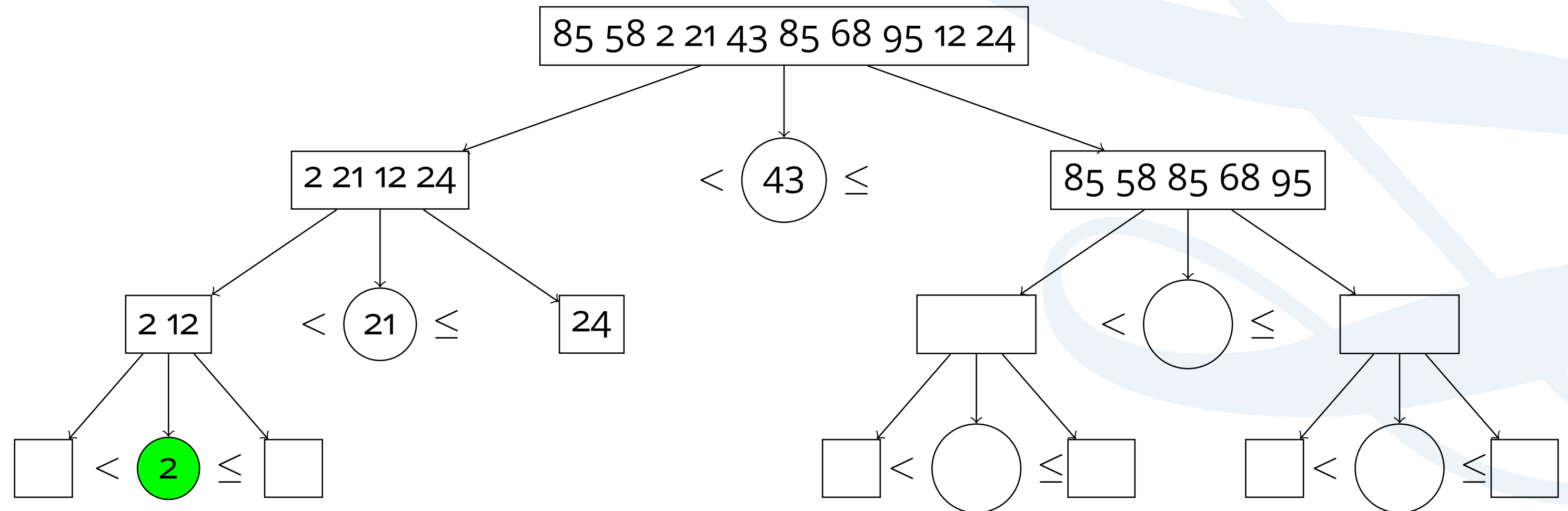
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

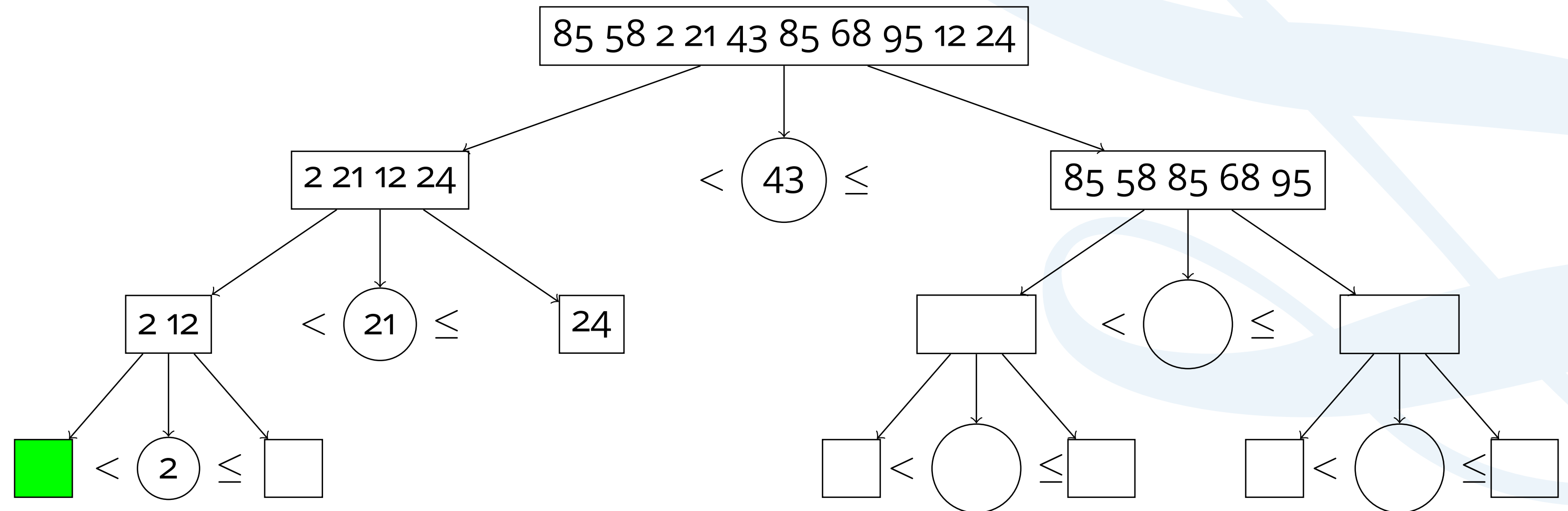


- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



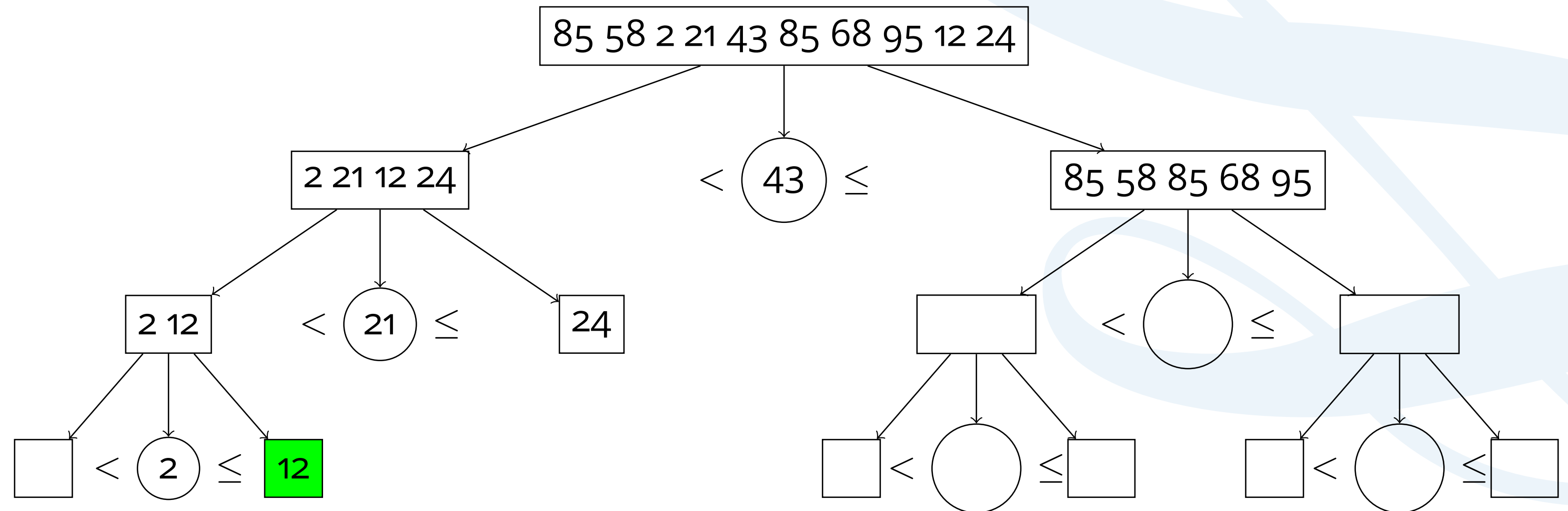


- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

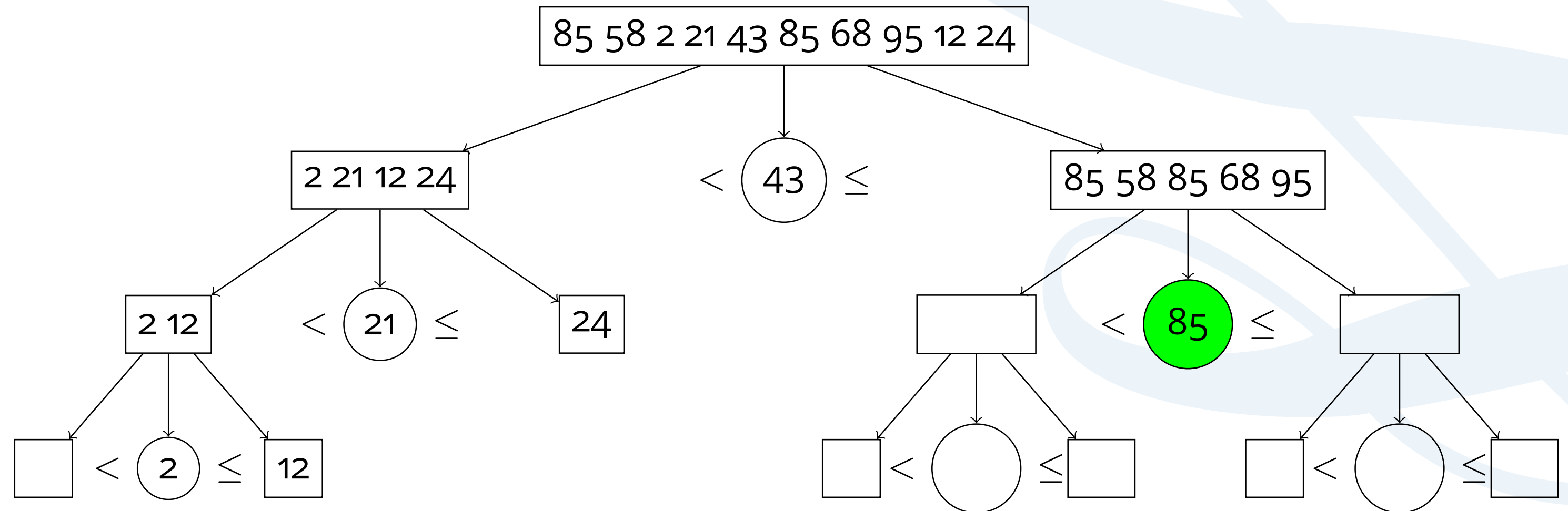




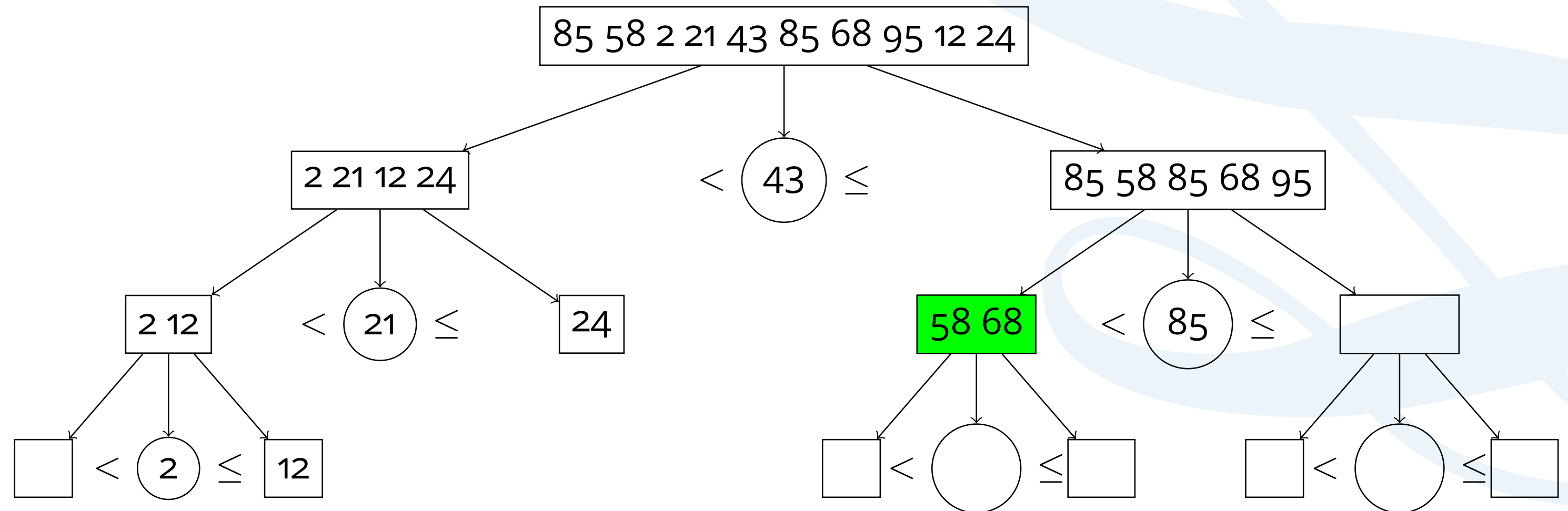
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



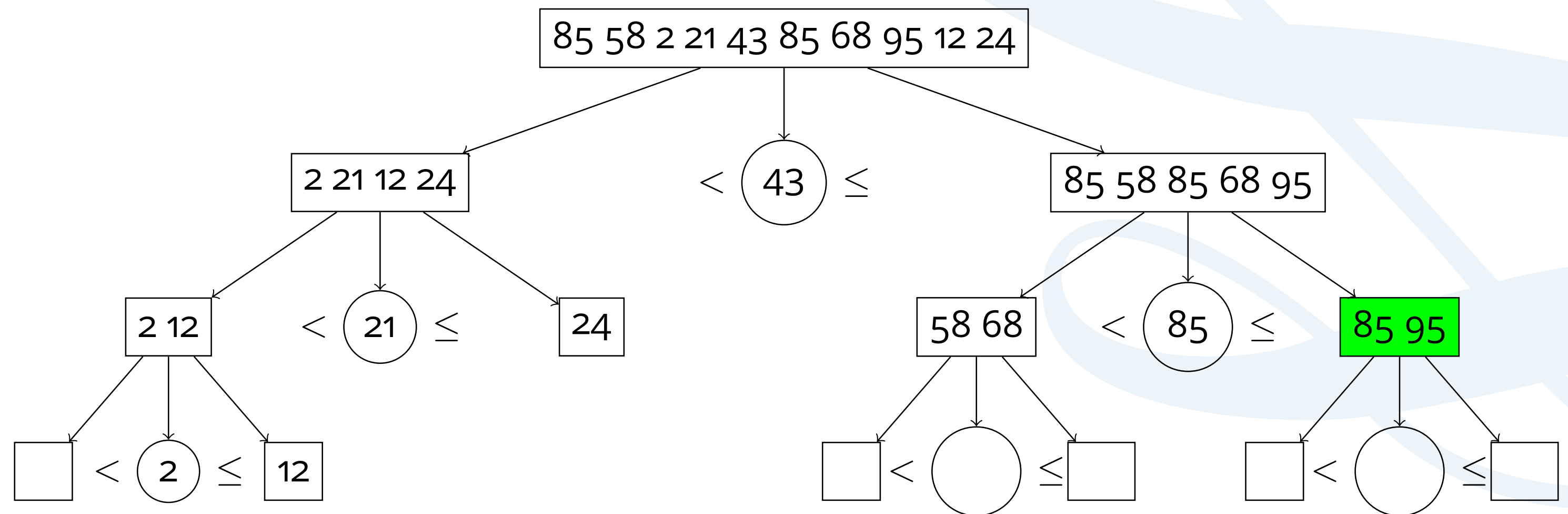
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



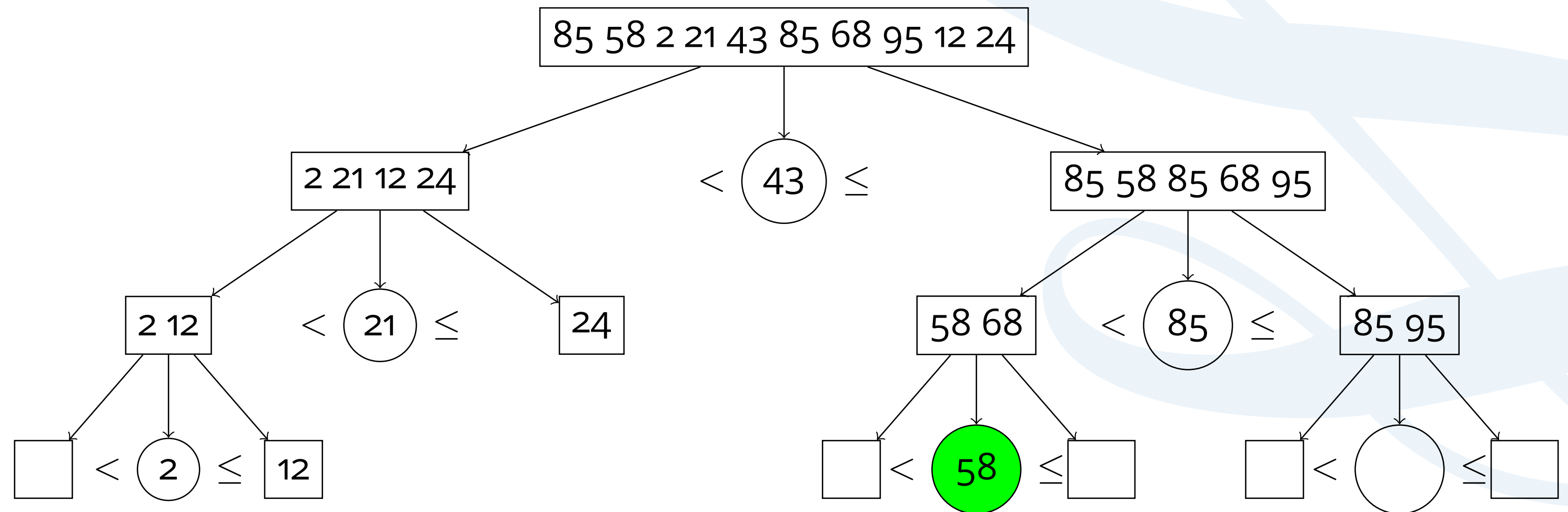
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



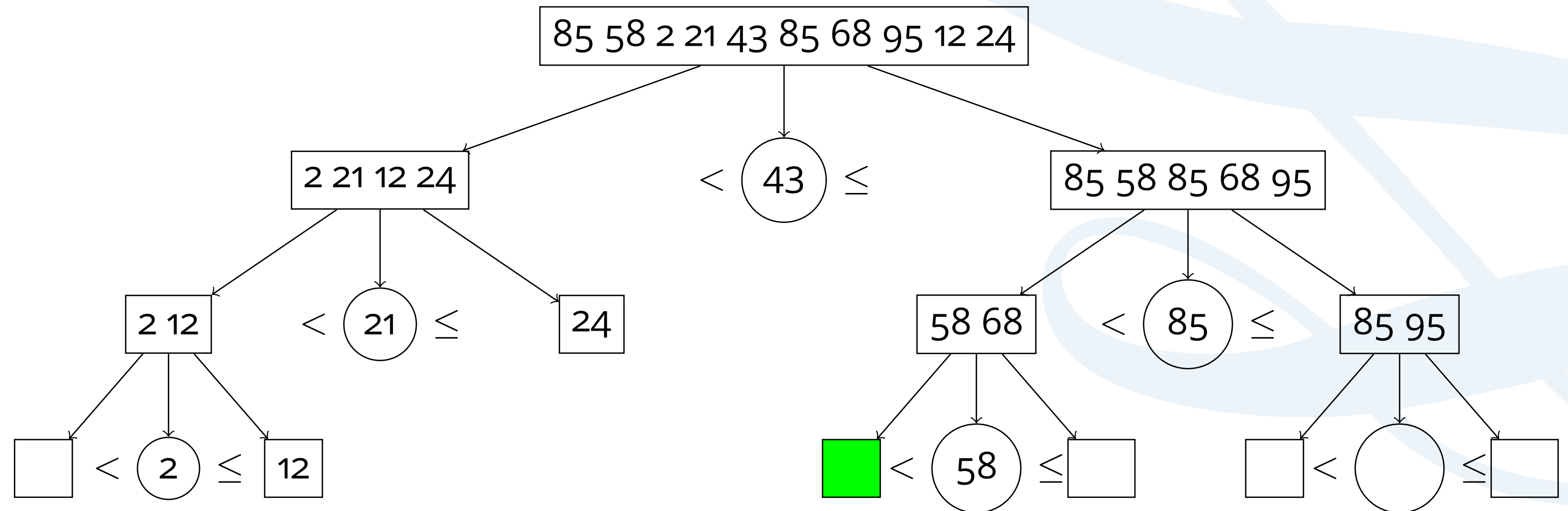
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



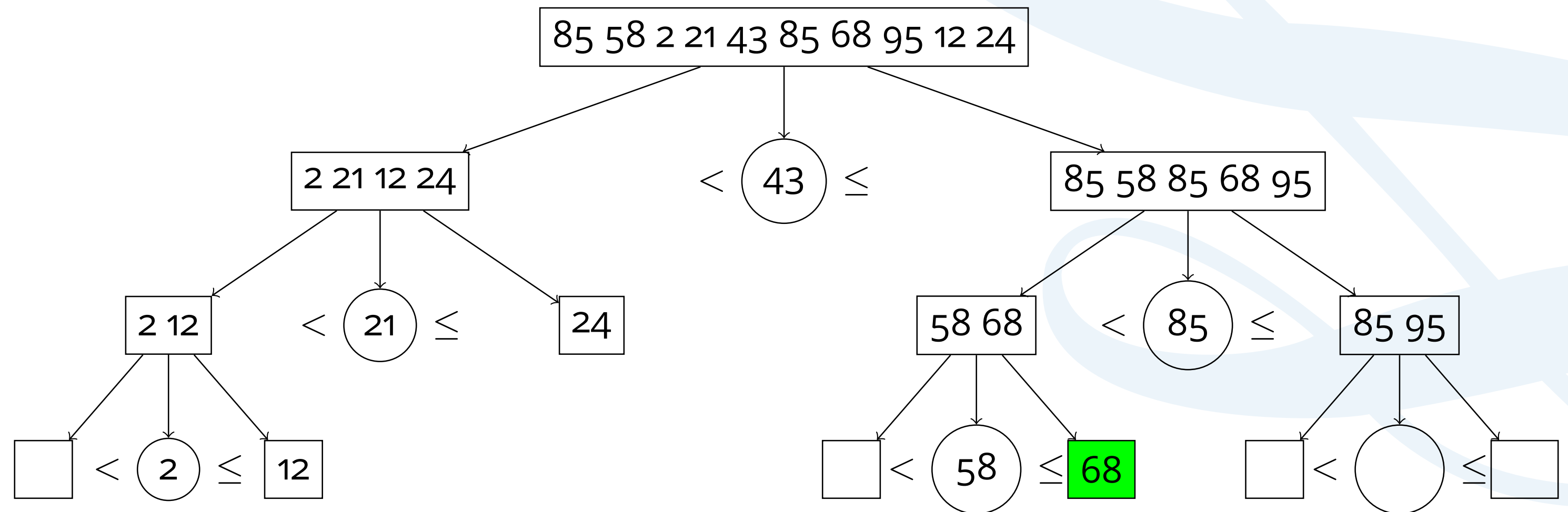
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

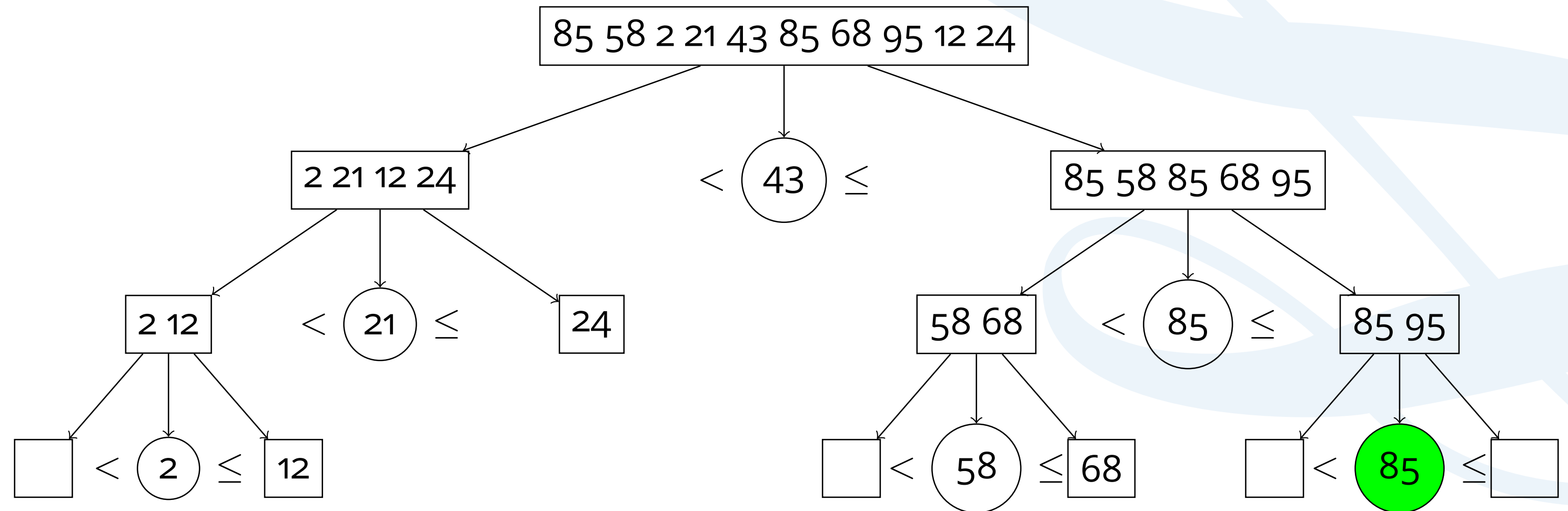


- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



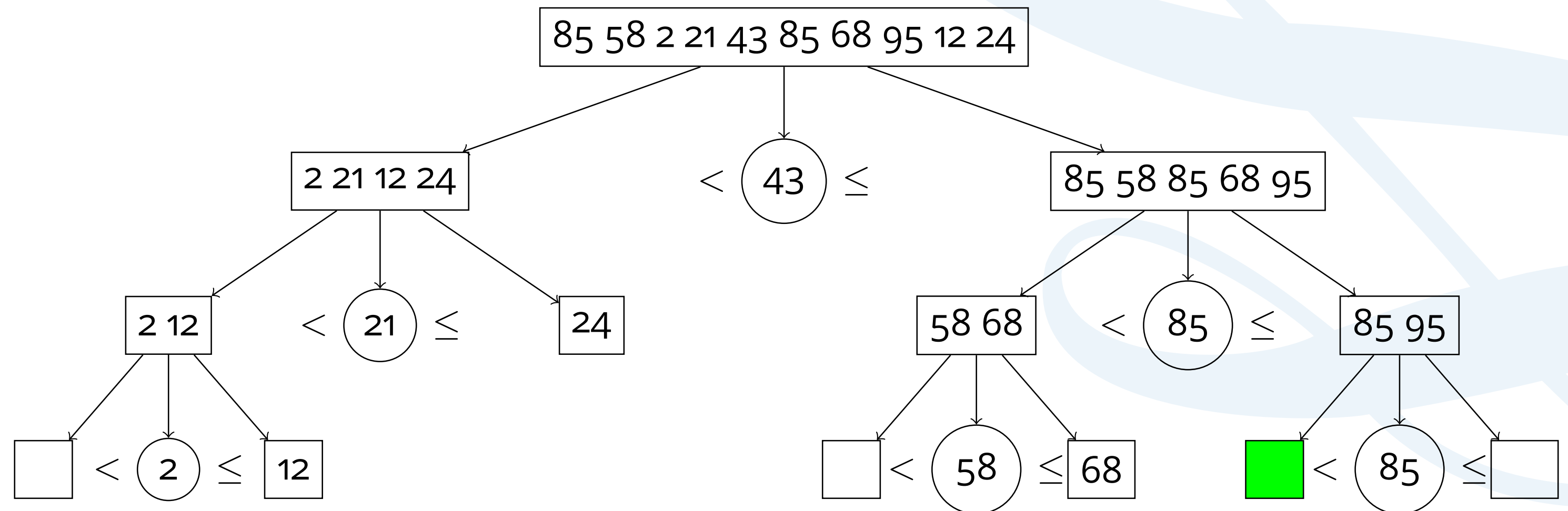


- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

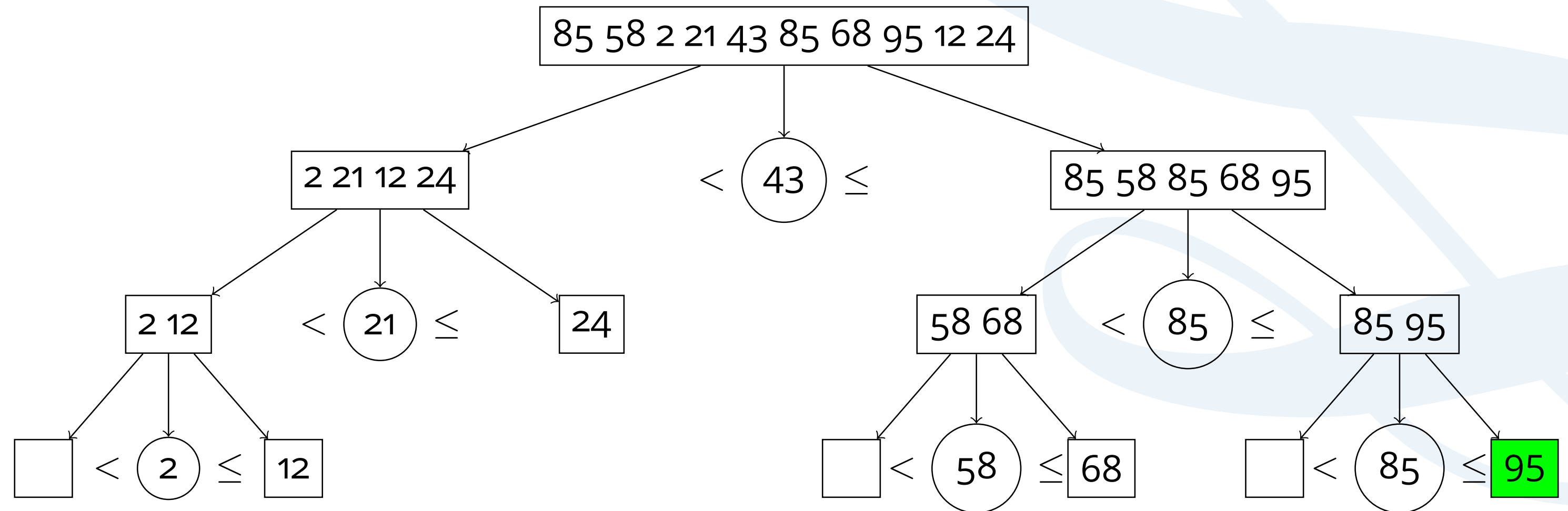




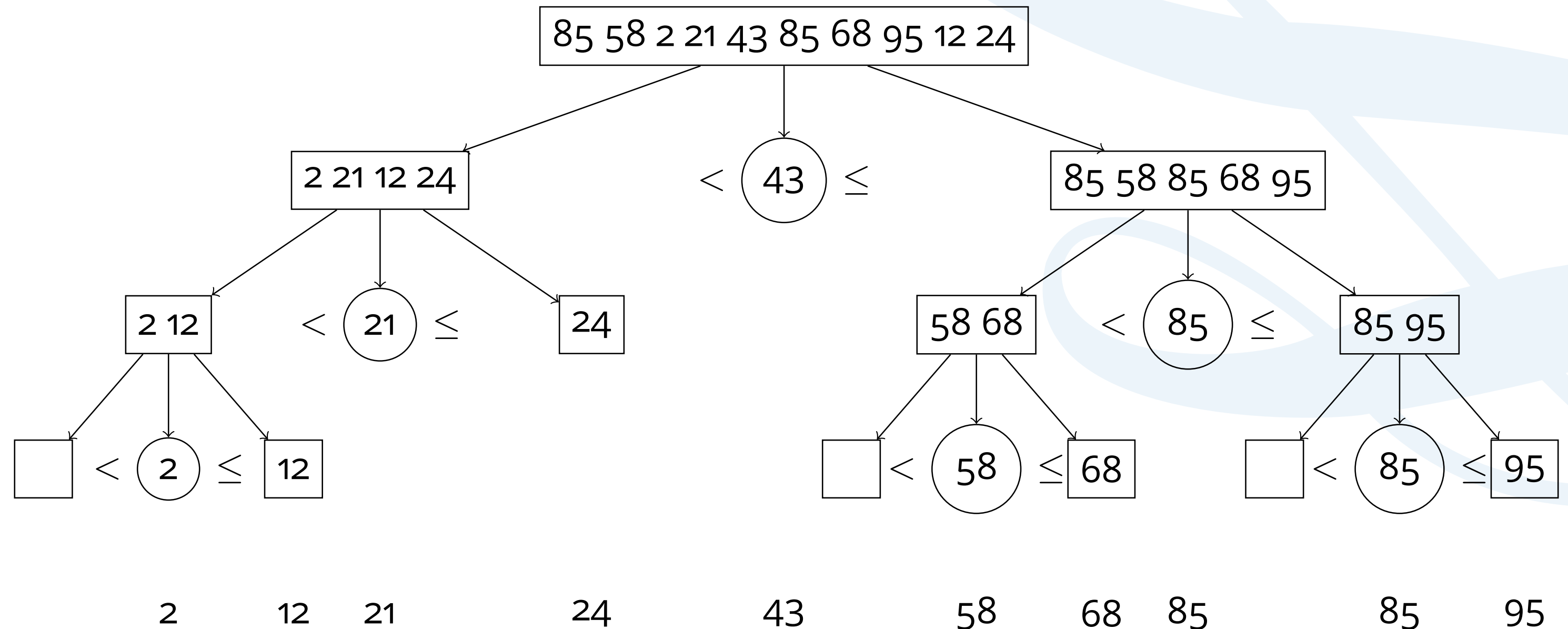
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values  $<$  pivot in one group.
- 3 Put all values  $>$  pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



Quicksort is...

- ...sometimes in-place.
  - Depends on implementation.
- ...sometimes stable.
  - Depends on implementation.

Some issues with the original algorithms (1959).

- Choosing the pivot.
  - First element.
  - Middle element.
  - Average of first, middle and last.
- Repeated elements.
  - Fat partition.

# Divide and Conquer

C

Quicksort is a divide and conquer algorithm.

- Too hard to sort the whole sequence?
- Divide the problem.
  - Still too hard?
  - Divide the problem.
    - Still too hard?
    - Divide the problem.
    - Etc, etc, etc.

Naturally suited for parallelism.

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?
- What are you sorting?
  - Linked lists?
  - Sequential memory (arrays)?



# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?
- What are you sorting?
  - Linked lists?
  - Sequential memory (arrays)?
- Where are you sorting?
  - RAM?
  - EEPROM? cheap to read, expensive to write.

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?
- What are you sorting?
  - Linked lists?
  - Sequential memory (arrays)?
- Where are you sorting?
  - RAM?
  - EEPROM? cheap to read, expensive to write.
- Size of  $n$ .
  - Insertion sort with small  $n$ .

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?
- What are you sorting?
  - Linked lists?
  - Sequential memory (arrays)?
- Where are you sorting?
  - RAM?
  - EEPROM? cheap to read, expensive to write.
- Size of  $n$ .
  - Insertion sort with small  $n$ .
- Consistent performance.
  - Selection sort.

# Comparing algorithms

I

Have seen there are many ways to sort.

- Best sorting algorithm depends on multiple factors.
- Good in one situation is bad in another.
- Stability? In place?
- What are you sorting?
  - Linked lists?
  - Sequential memory (arrays)?
- Where are you sorting?
  - RAM?
  - EEPROM? cheap to read, expensive to write.
- Size of  $n$ .
  - Insertion sort with small  $n$ .
- Consistent performance.
  - Selection sort.

# Quiz

# Recap

- Many sorting algorithms.
- Bubblesort.
- Selection sort.
- Quicksort
- Advantages/disadvantages.
  - In place.
  - In order.
  - Divide and Conquer.
- Performance
  - $O()$
  - Sequence type.
  - Read/writes.
  - Size of  $n$ .

# The End