Sorting algorithms

David Croft

Coventry University

david.croft@coventry.ac.uk

November 20, 2015

Overview

Coventry University

Sorting

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

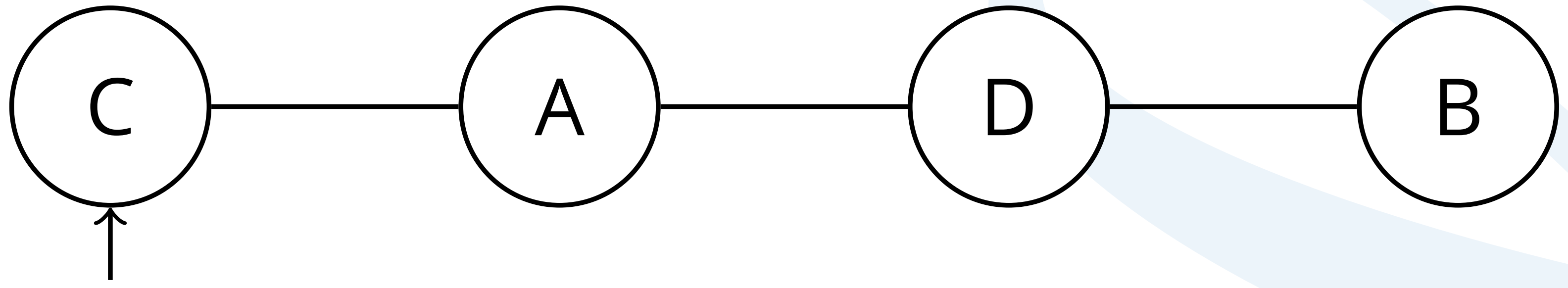
Very simple sort.

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

Bubblesort

Very simple sort.

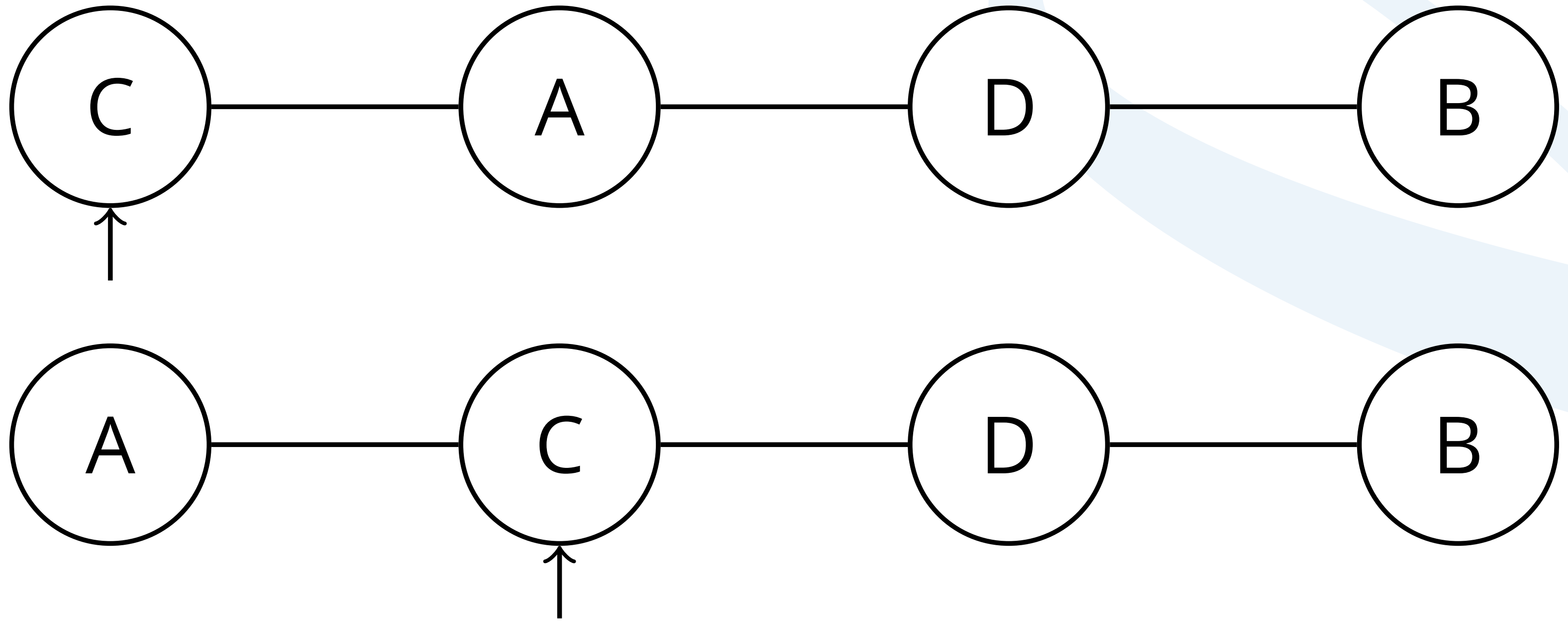
- Compares each item to the next in the sequence.
  - Swap items if in wrong order.



Bubblesort

Very simple sort.

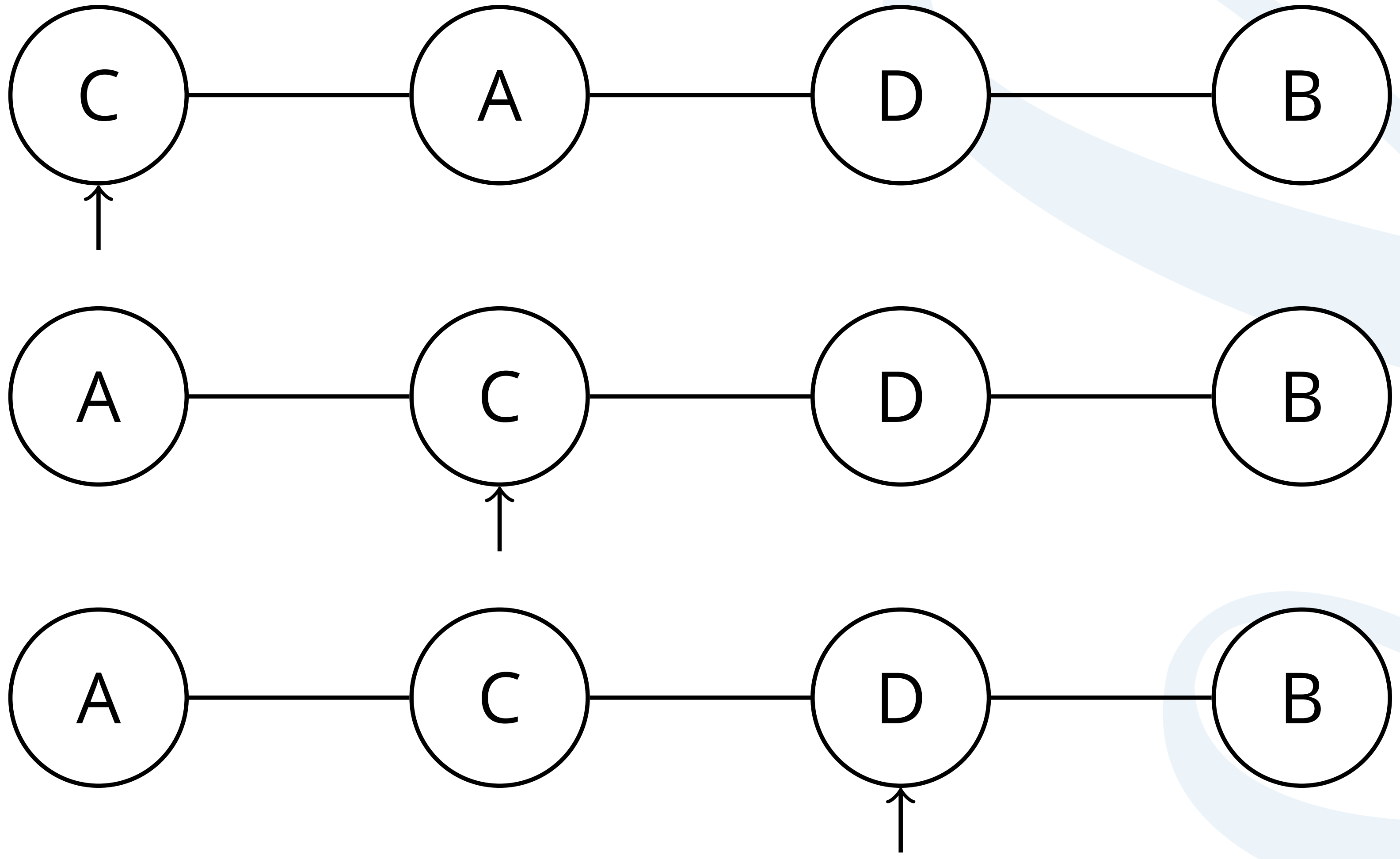
- Compares each item to the next in the sequence.
  - Swap items if in wrong order.



Bubblesort

Very simple sort.

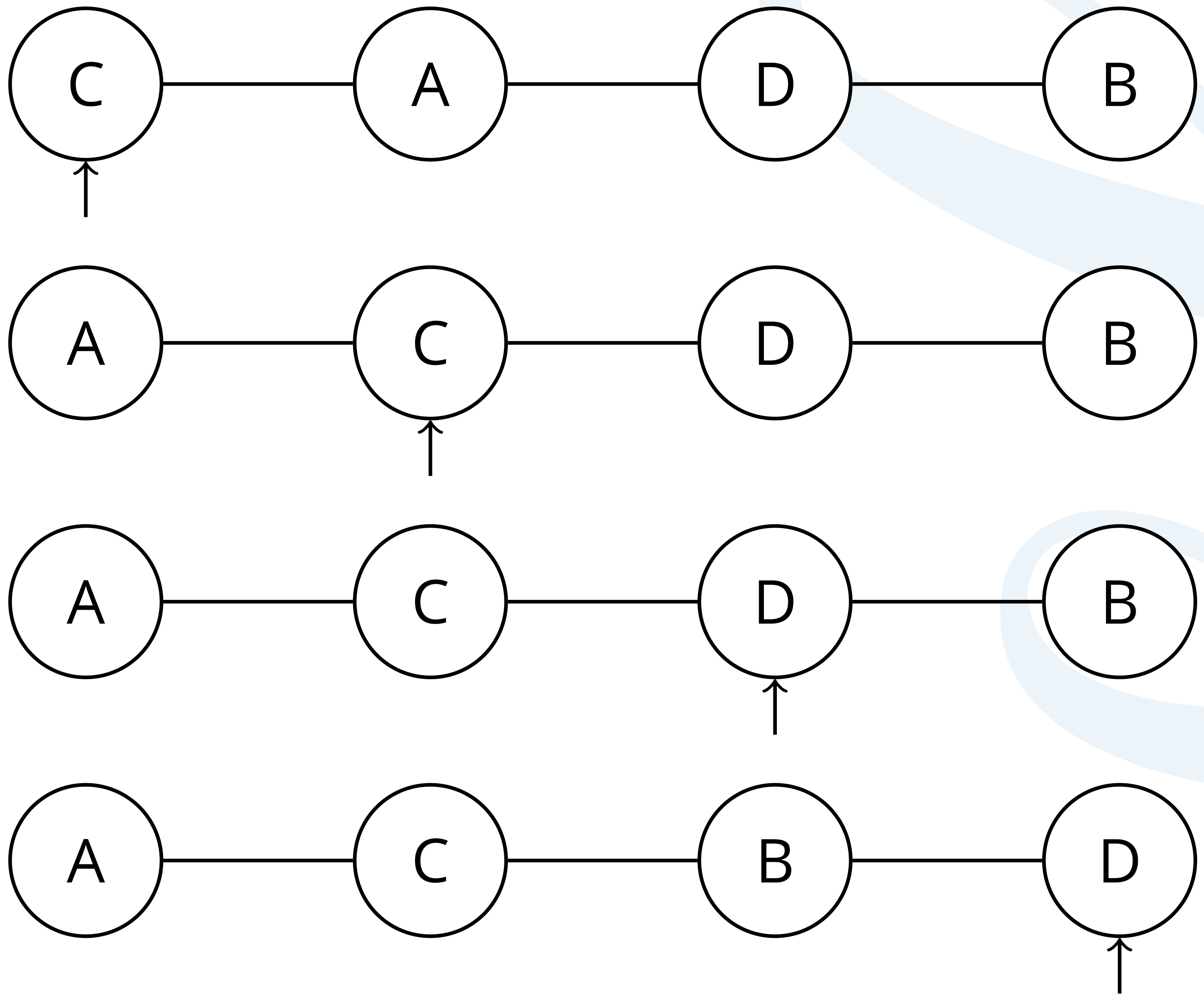
- Compares each item to the next in the sequence.
  - Swap items if in wrong order.



Bubblesort

Very simple sort.

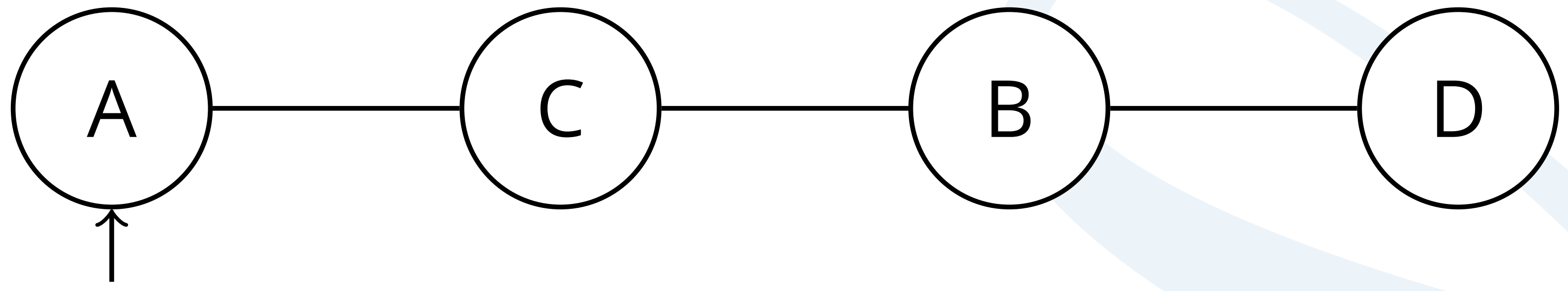
- Compares each item to the next in the sequence.
  - Swap items if in wrong order.



Bubblesort

Iterating over the sequence once isn't typically enough.

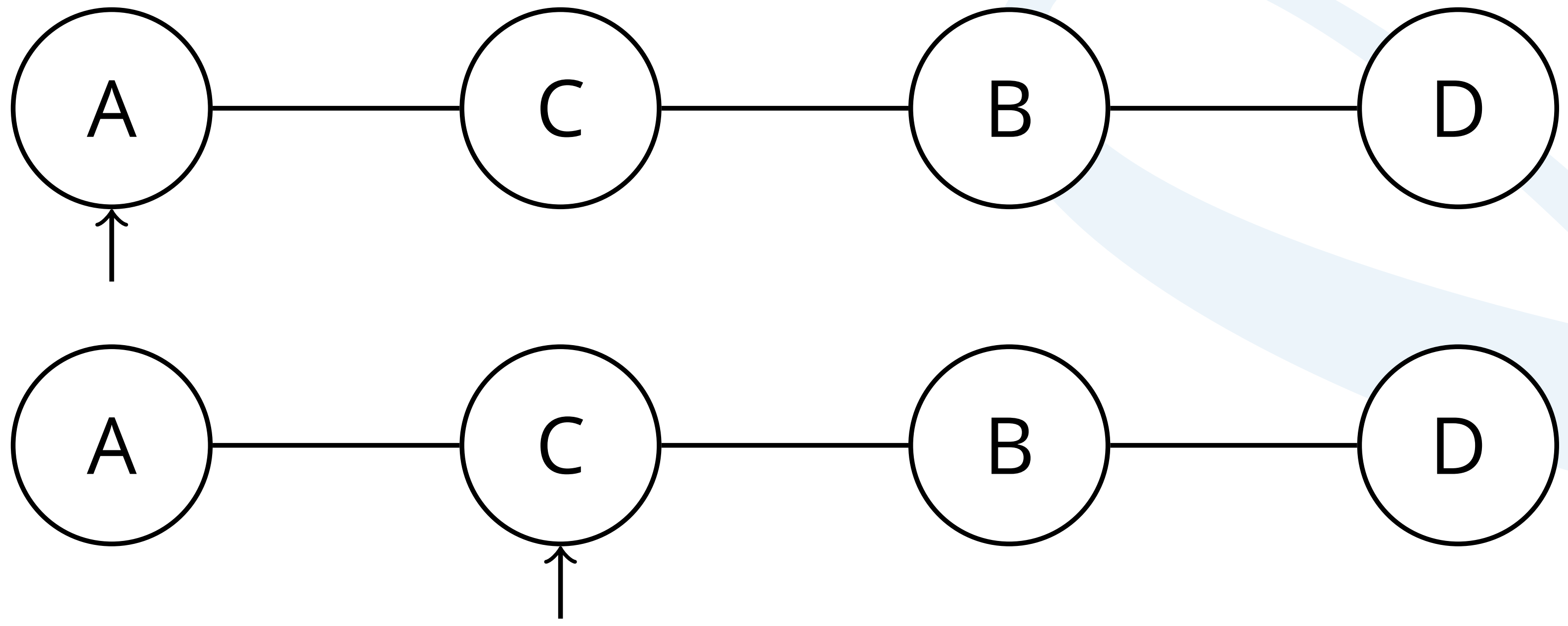
- Keep iterating over the sequence until elements are sorted.



Bubblesort

Iterating over the sequence once isn't typically enough.

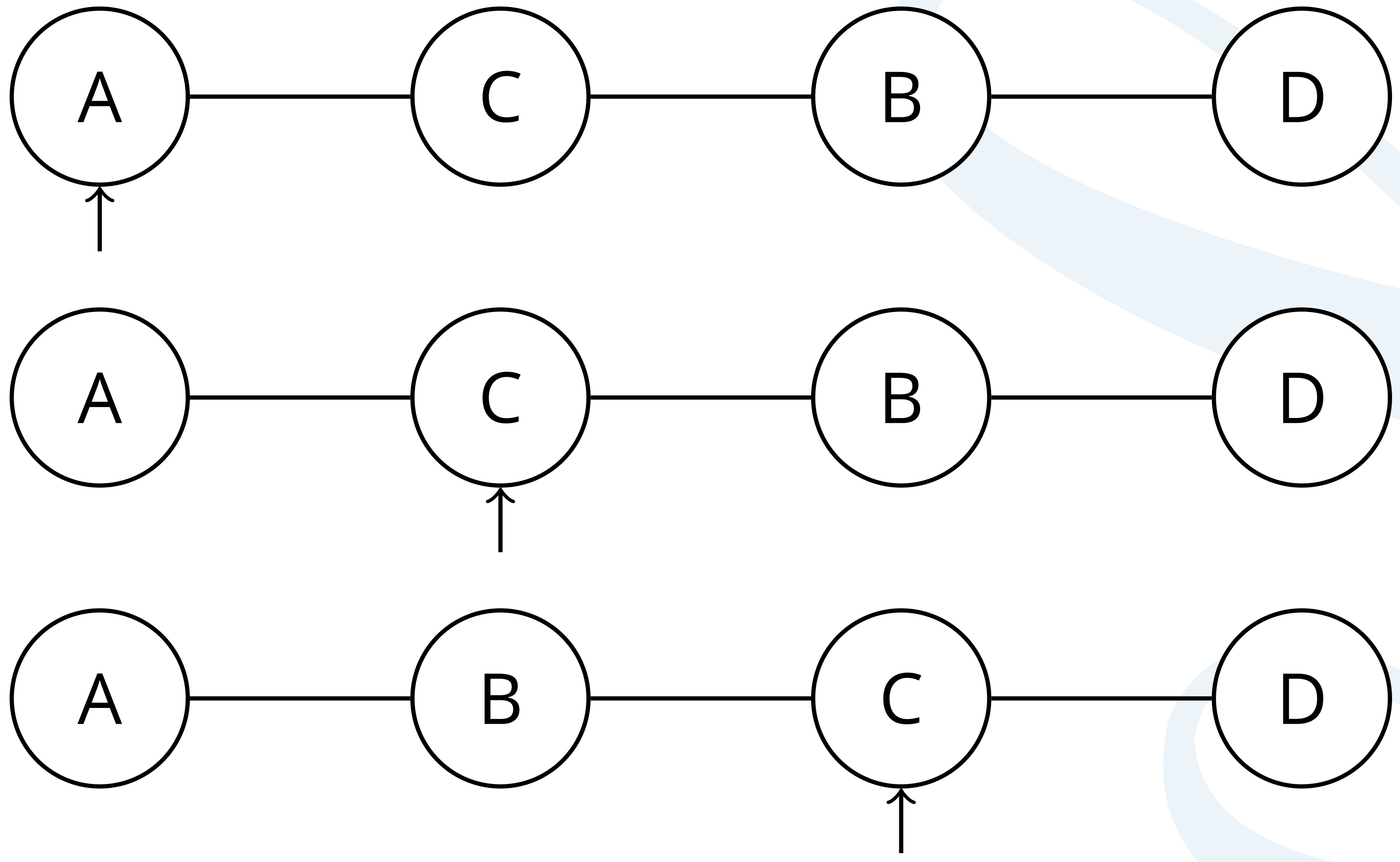
- Keep iterating over the sequence until elements are sorted.



Bubblesort

Iterating over the sequence once isn't typically enough.

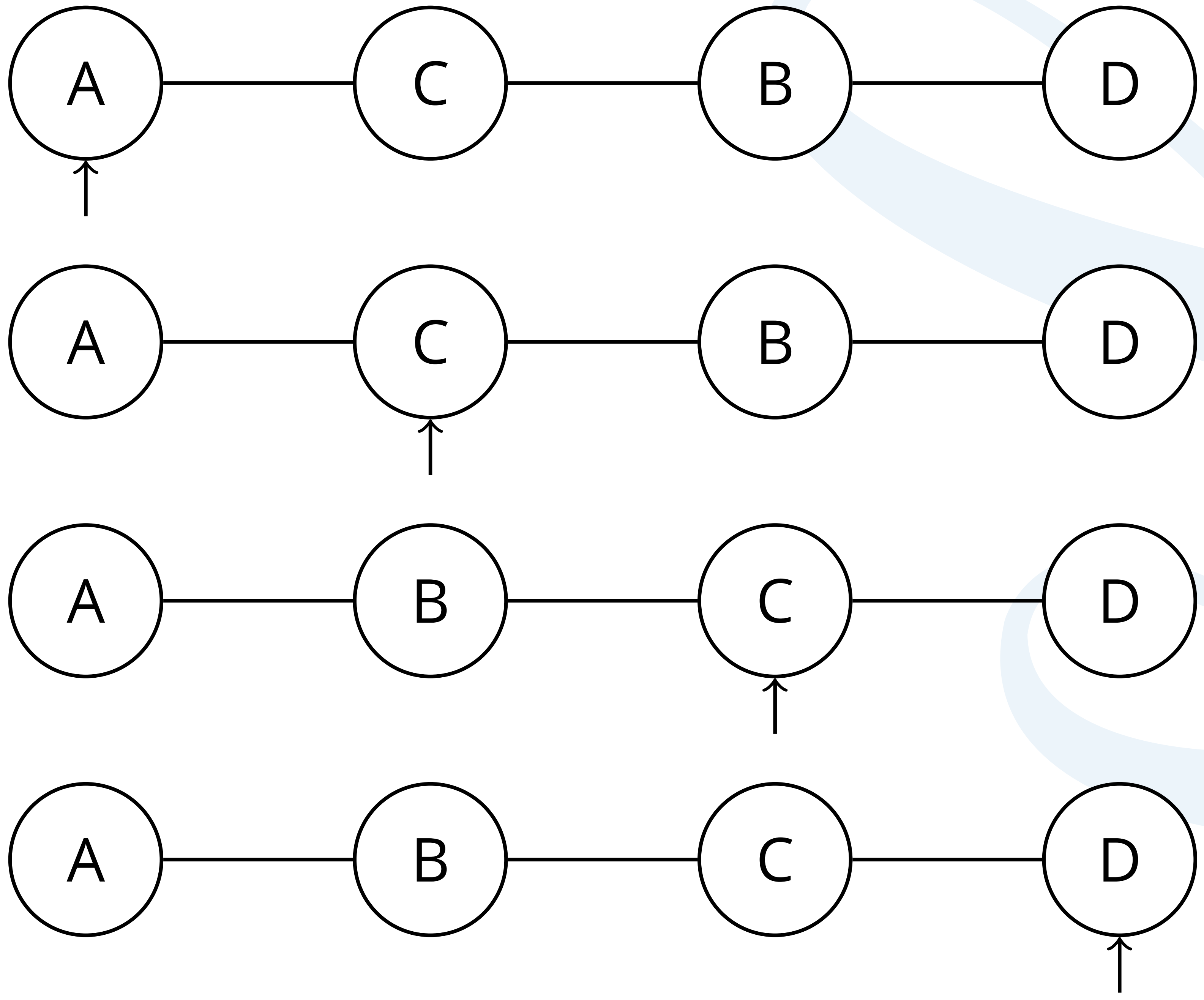
- Keep iterating over the sequence until elements are sorted.



Bubblesort

Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.



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. Stable sorting algorithms do not change the order of equivalent elements i.e. elements with the same value not have their relative orders changed after a stable sorting. With an unstable sorting algorithm the relative orders or equivalent elements can be changed. For some applications this is an important consideration. Imagine a queue in an emergency room. You want to treat the most serious conditions first, so you sort the people based on how bad their injury is. However, if two or more people have the same injury then they should get seen based on when they entered the queue.

In-place

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

One of the simplest sorting algorithms.

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

- Is rubbish.

Bubblesort

One of the simplest sorting algorithms.

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

Bubblesort

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 $O(n)$.

Order

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.

Order

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.

Order

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.

Order

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.

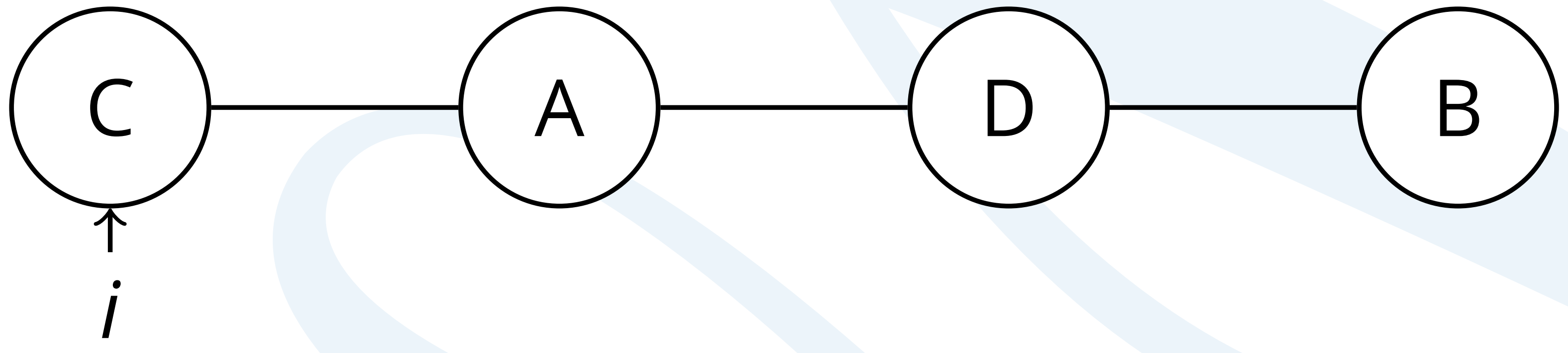
So sorting algorithms have 3 $O()$ values.

Selection sort

- Divides sequence into sorted and unsorted regions.
- Not stable.
- In place.

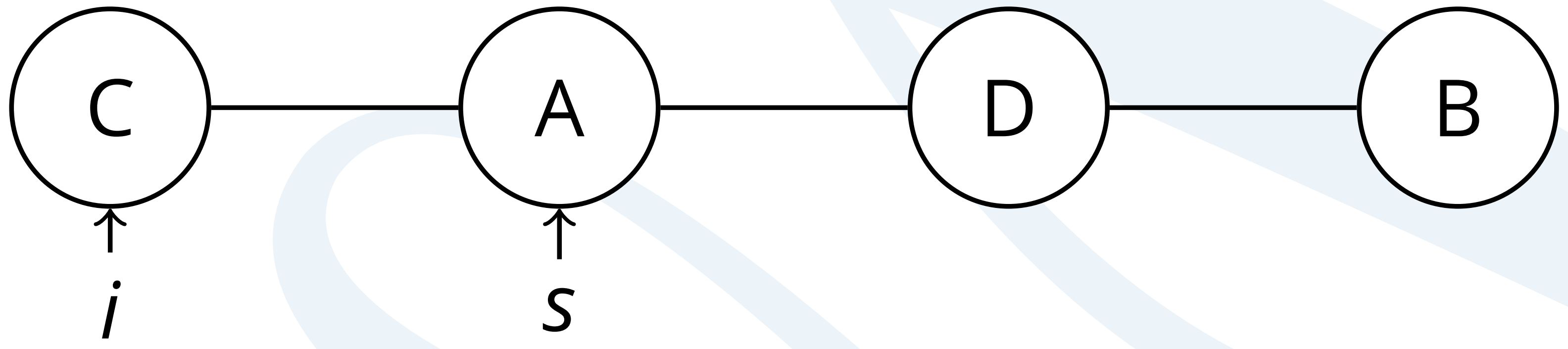
1. Iterate over sequence.
2. For each element search the remaining elements on its right for the smallest value.
3. Swap smallest element with current element.

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.

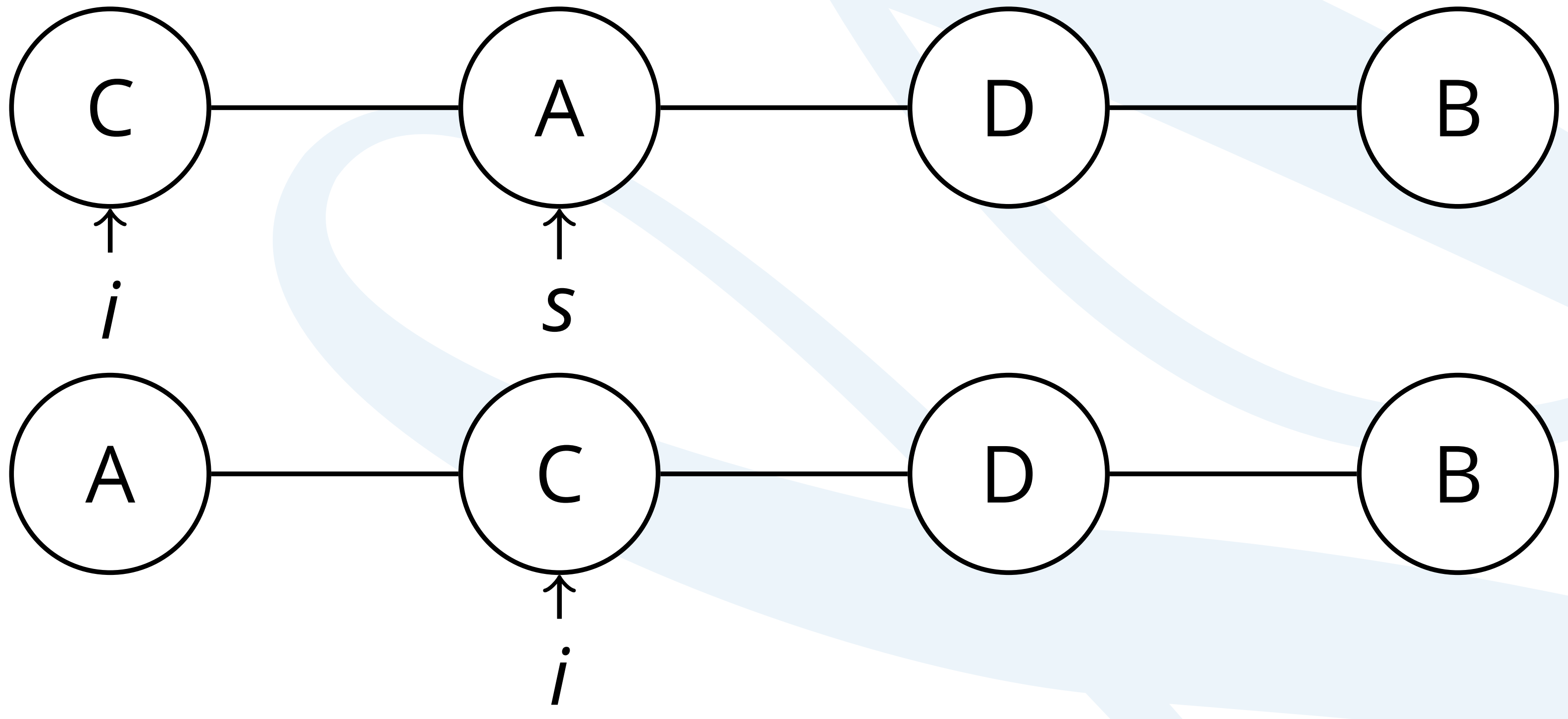
Selection sort II



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.

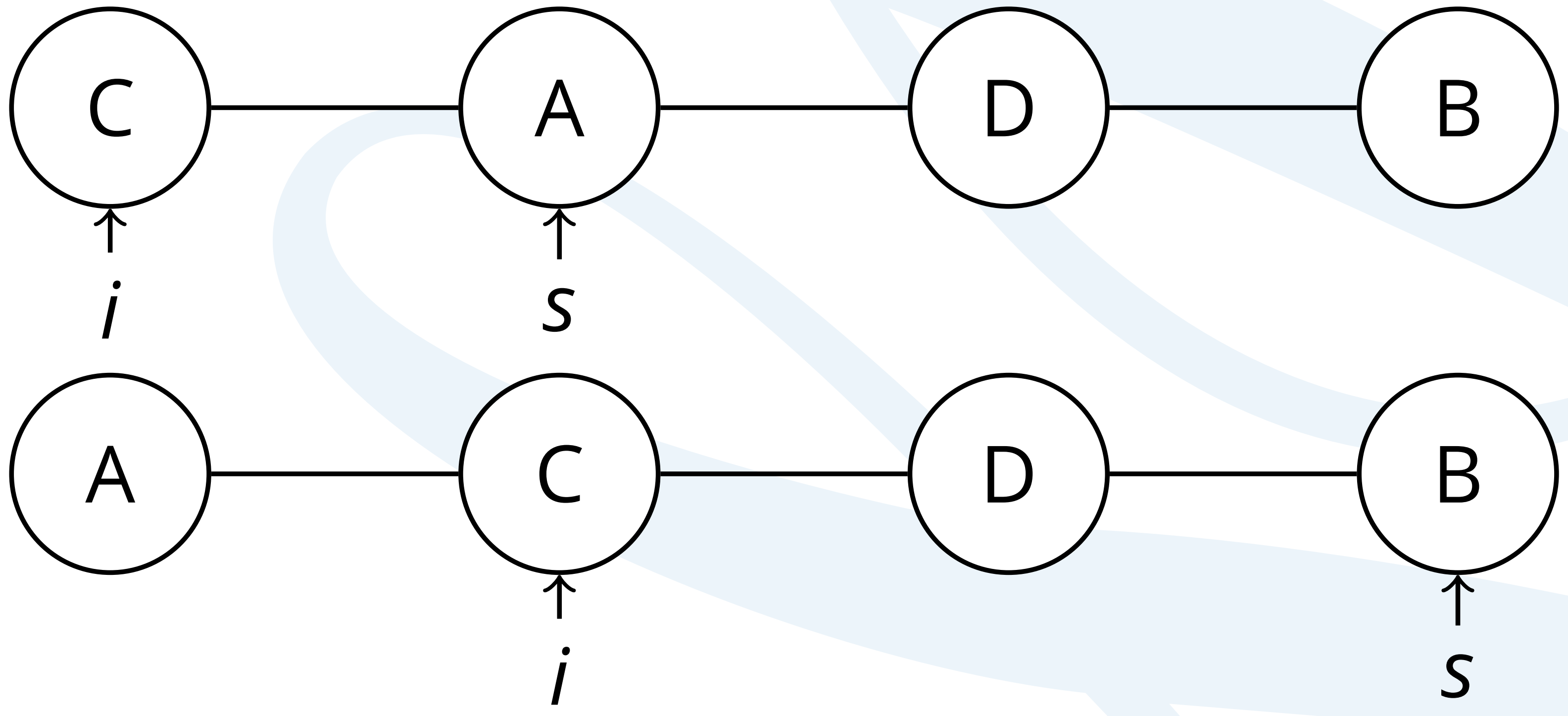
Selection sort II

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.



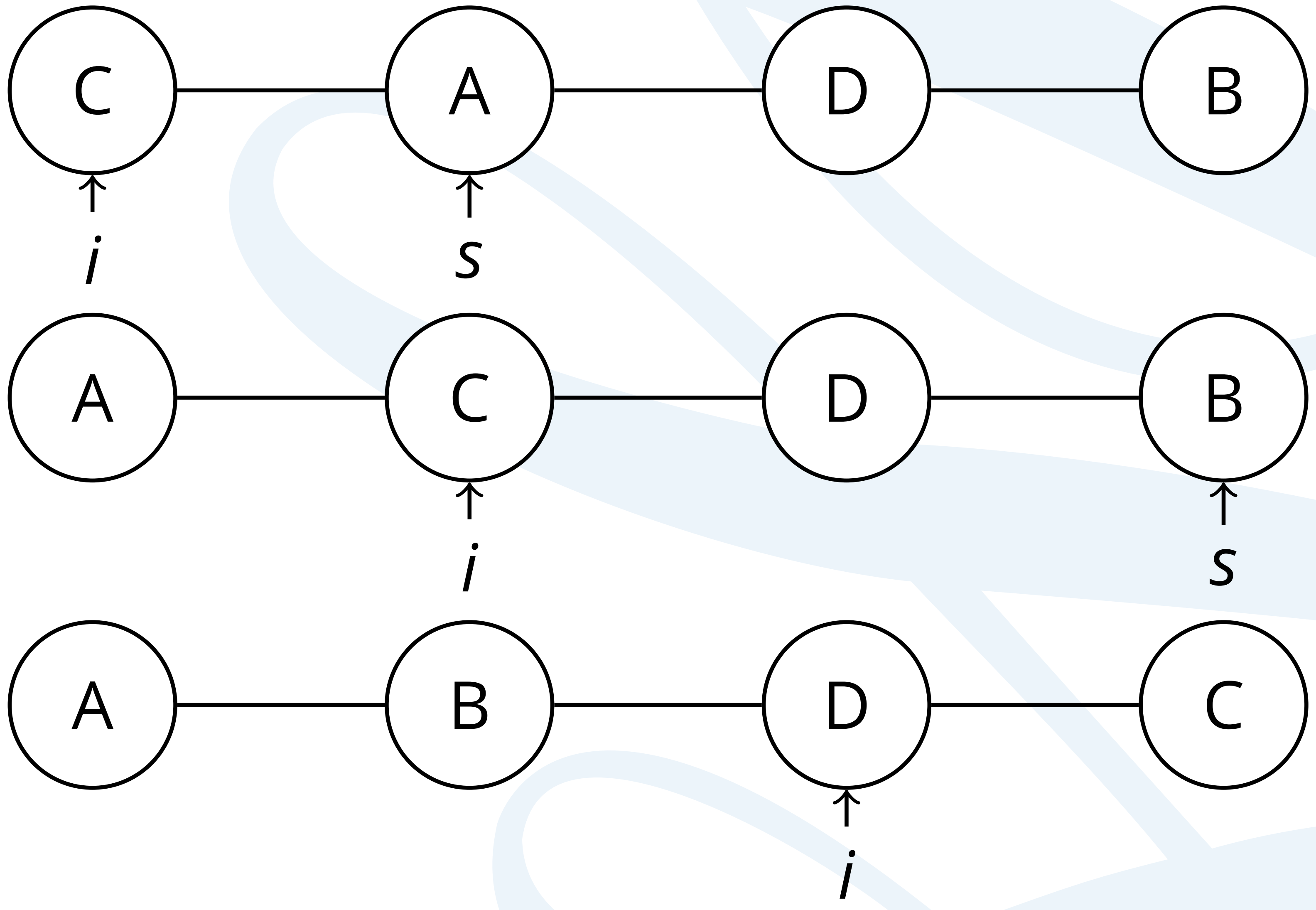
Selection sort II

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.



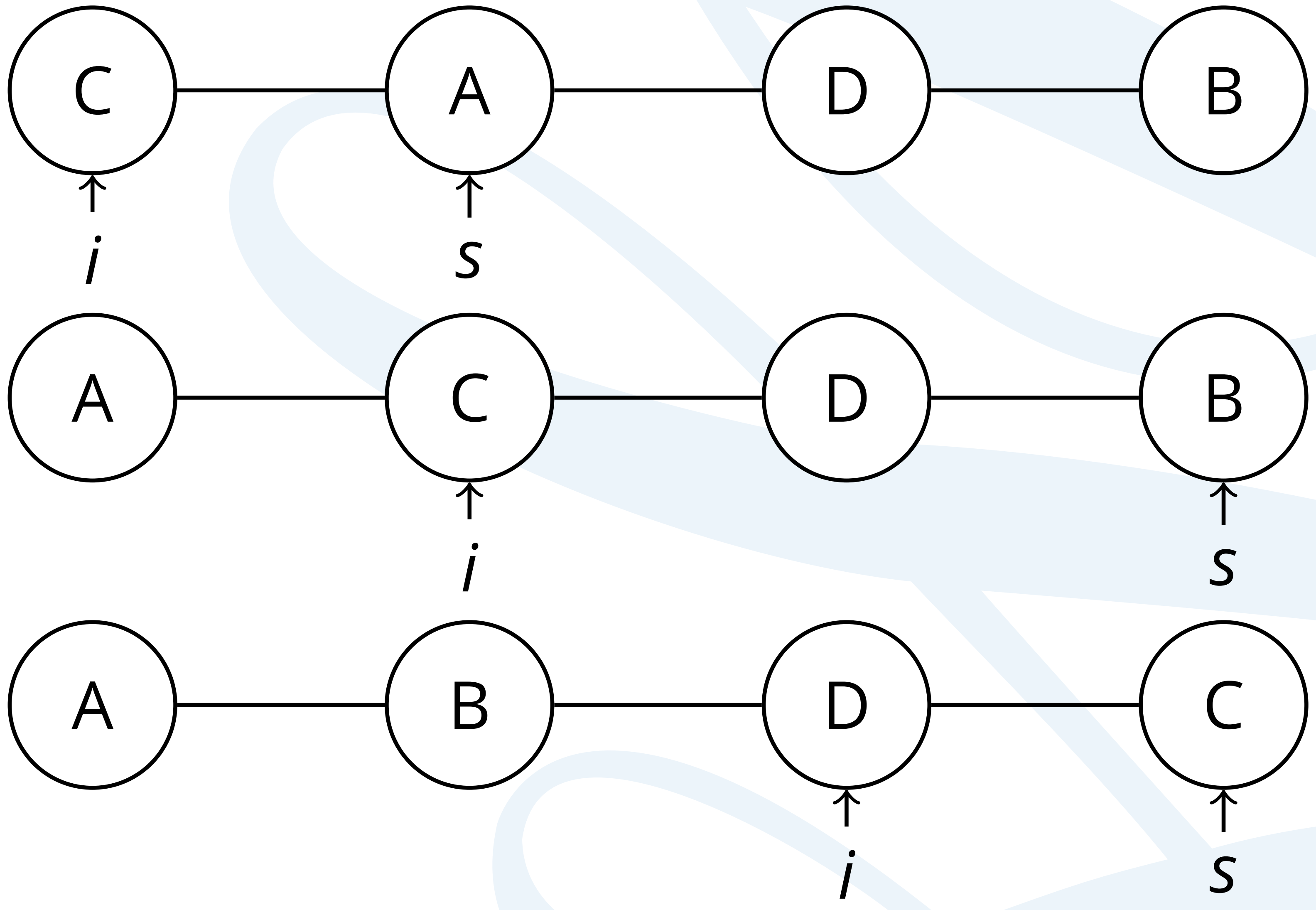
Selection sort II

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.



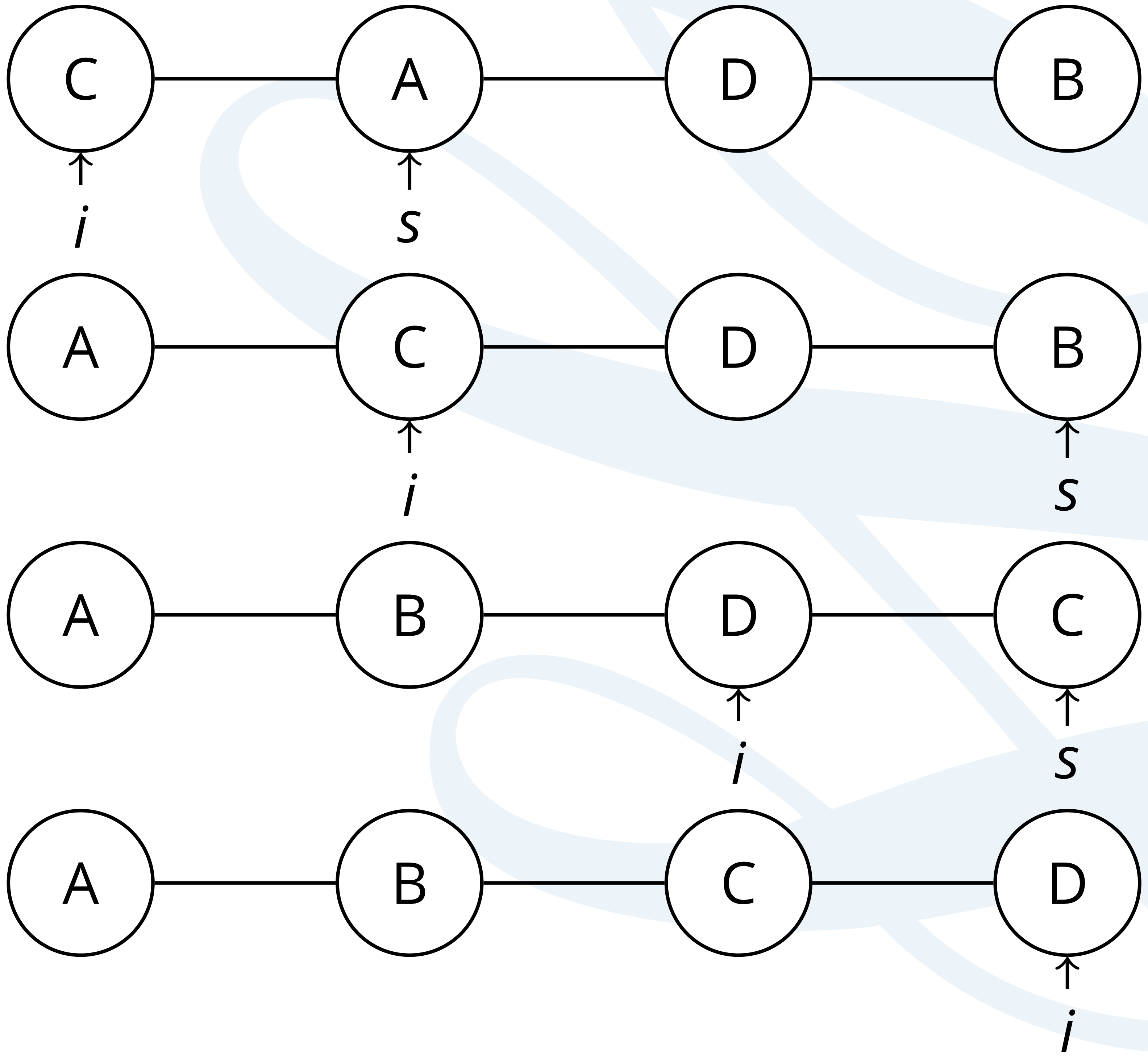
Selection sort II

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.



$O()$

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.
- Tricky for comparing relative 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

Neither bubble or selection sort are very good.

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

One of the fastest sorting algorithms.

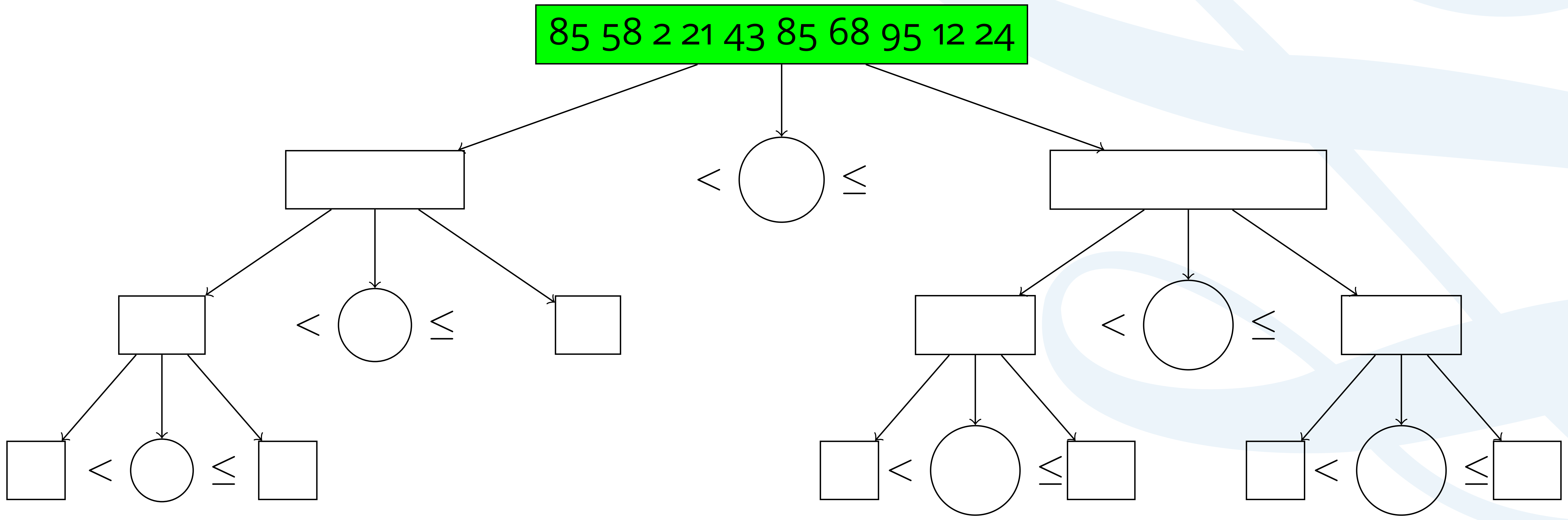
- Used in real life.
- Recursively breaks the sequence in half.
  - Divide & Conquer.

Quicksort II

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.

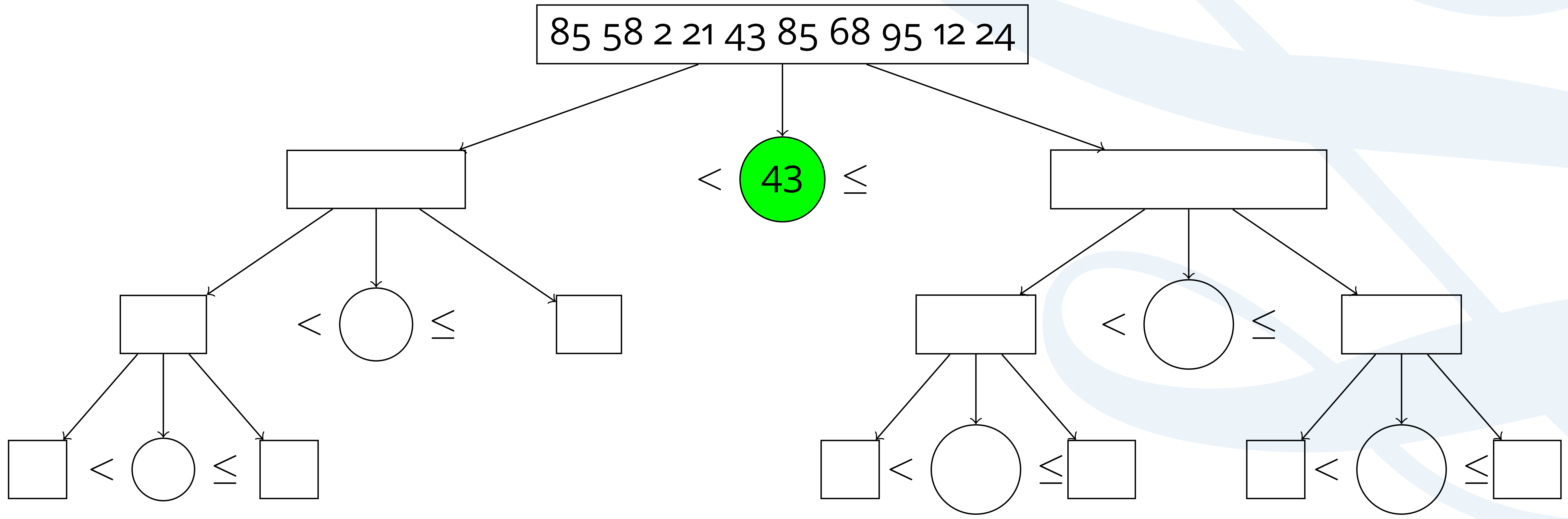
Quicksort III

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.



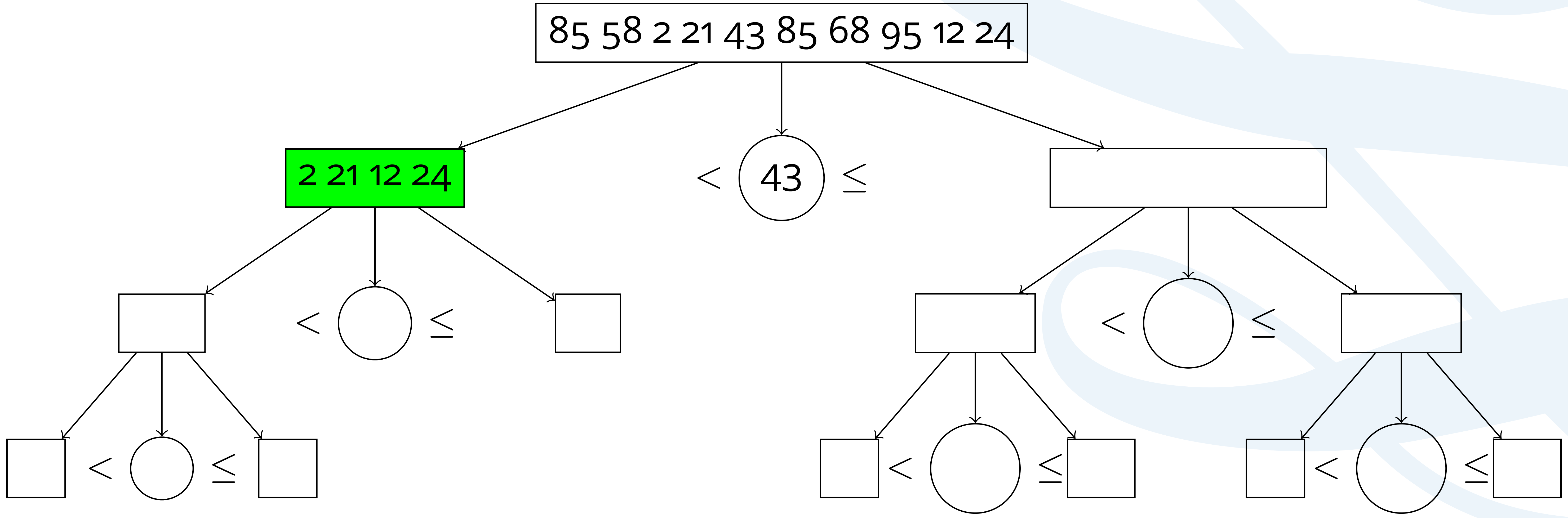
Quicksort III

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.



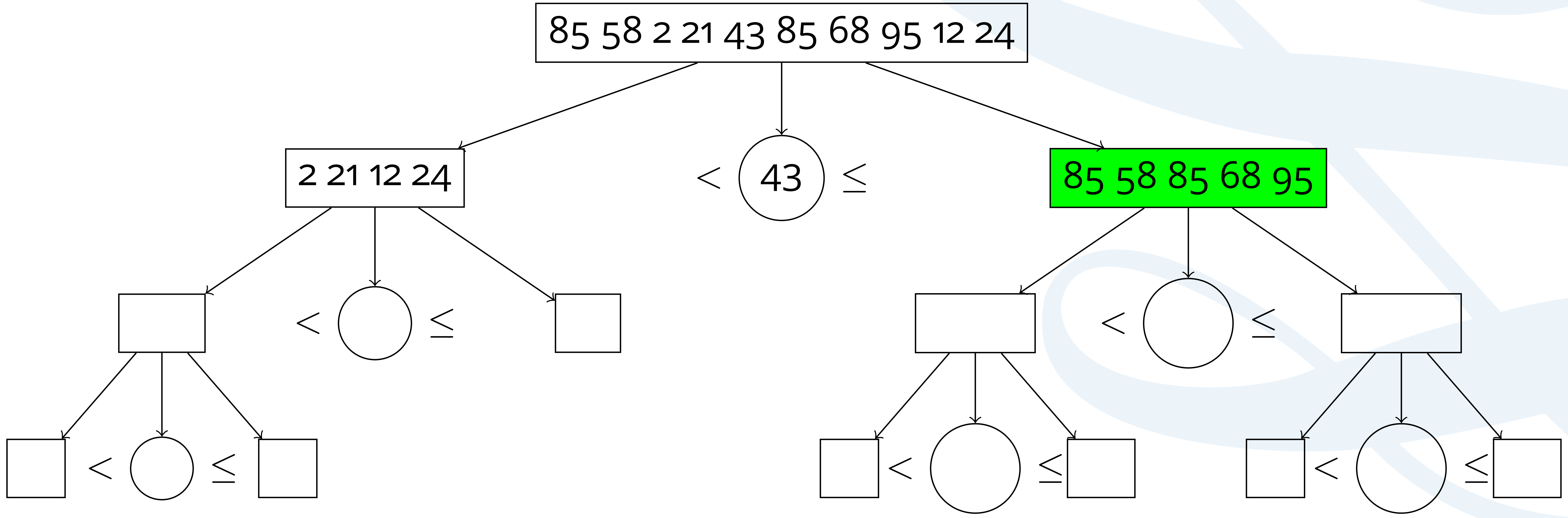
Quicksort III

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.



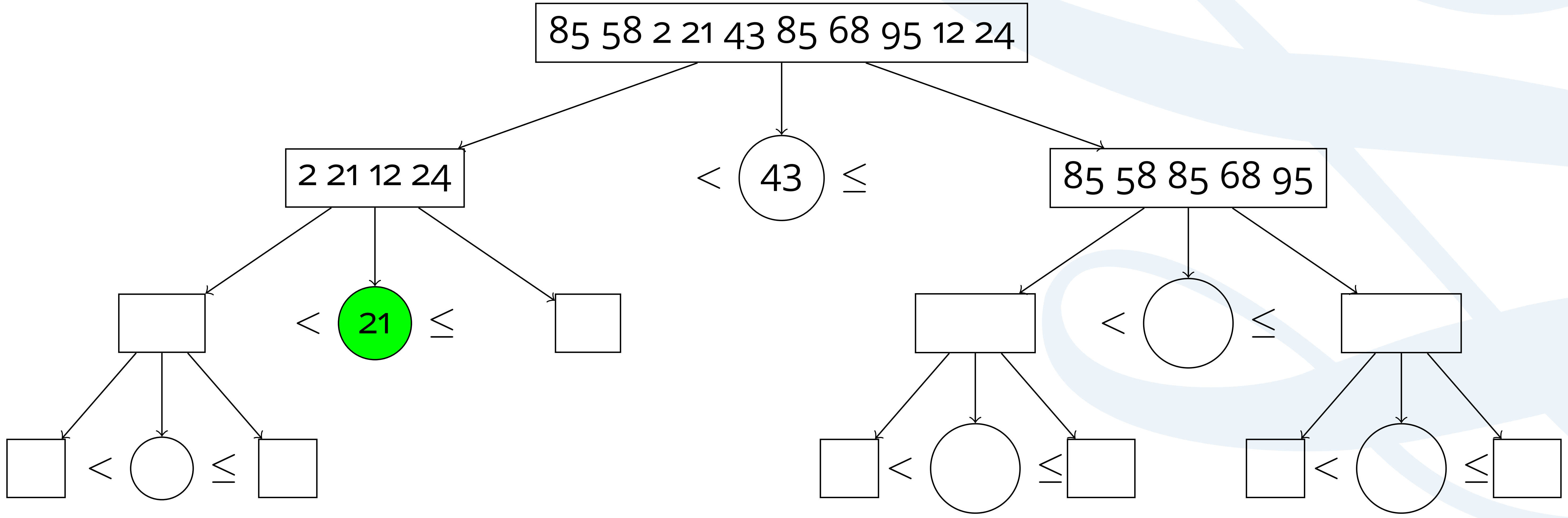
Quicksort III

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.



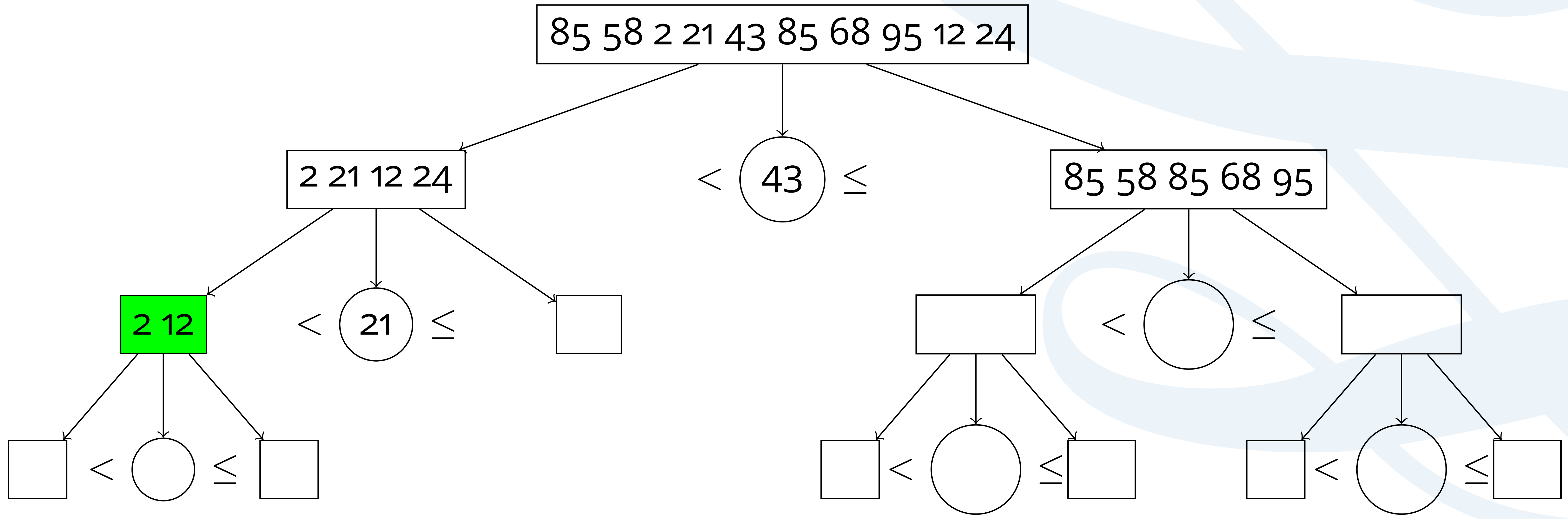
Quicksort III

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.



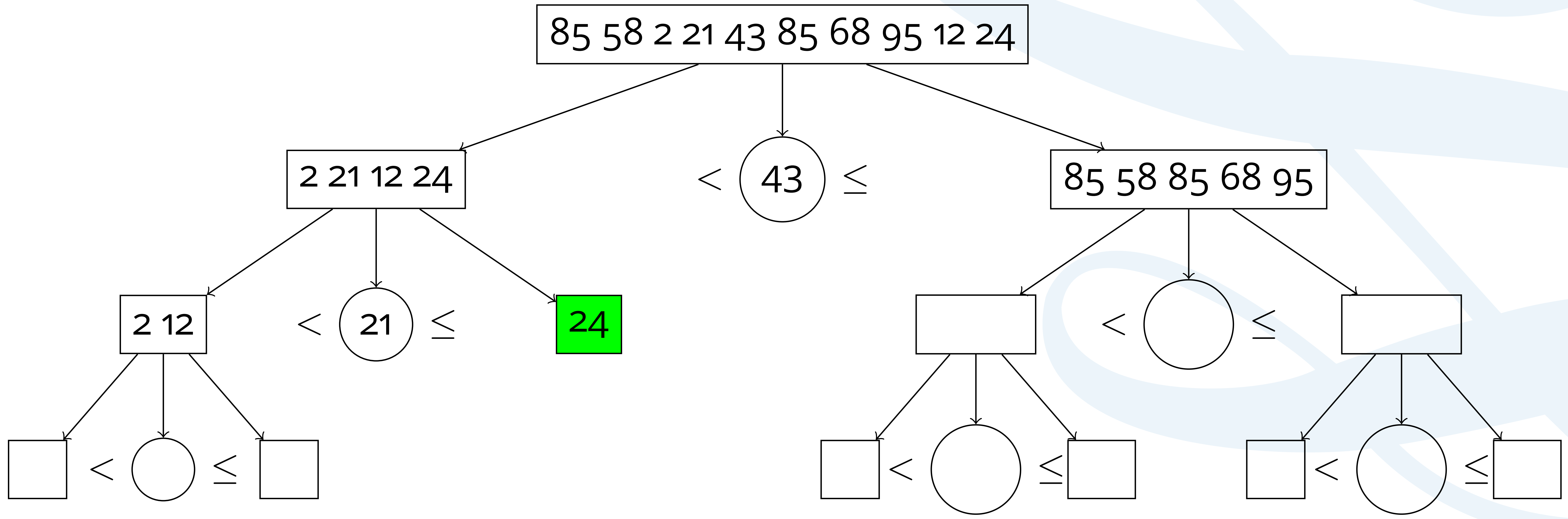
Quicksort III

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.



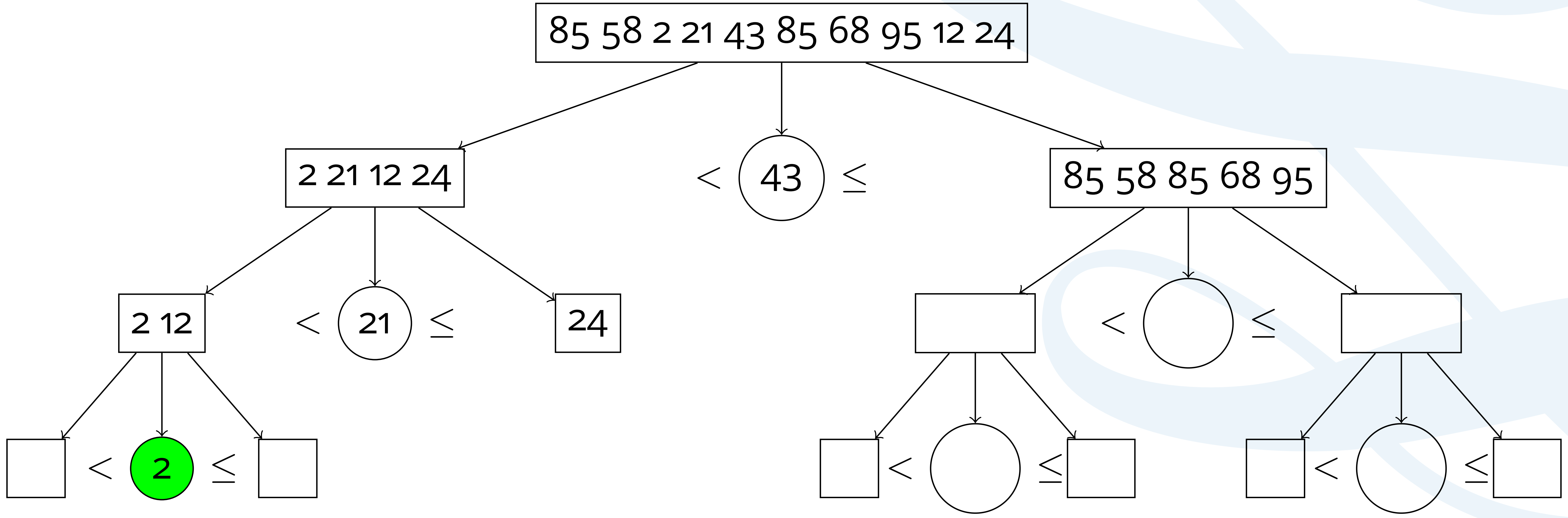
Quicksort III

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.



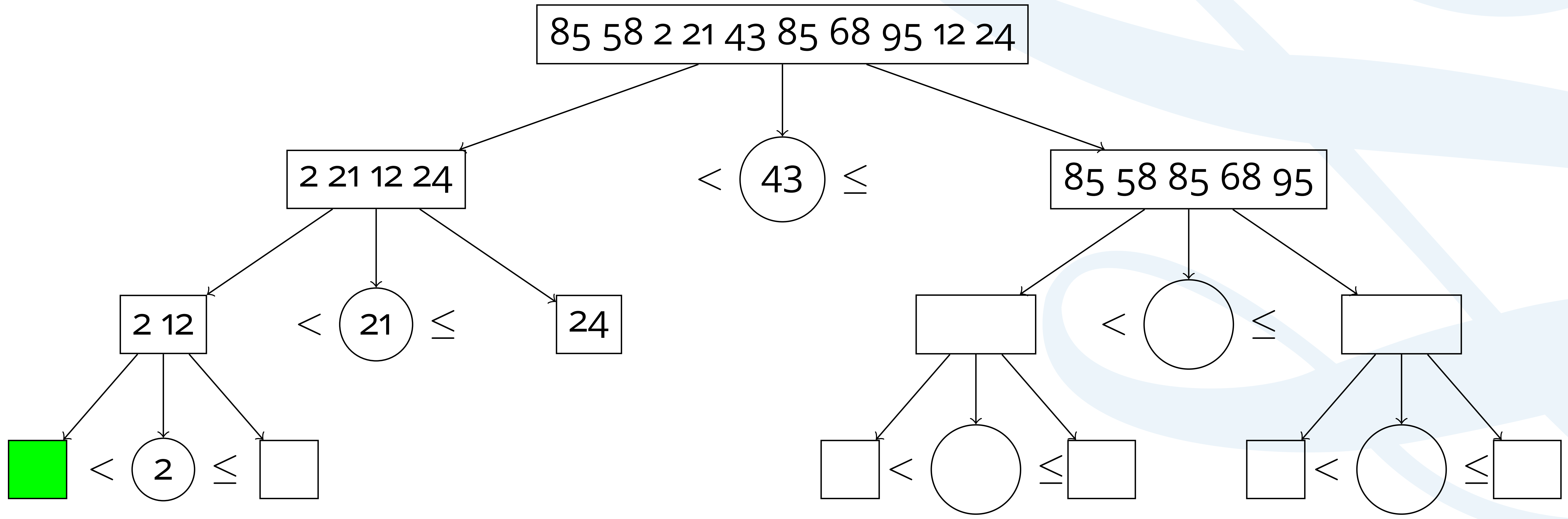
Quicksort III

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.



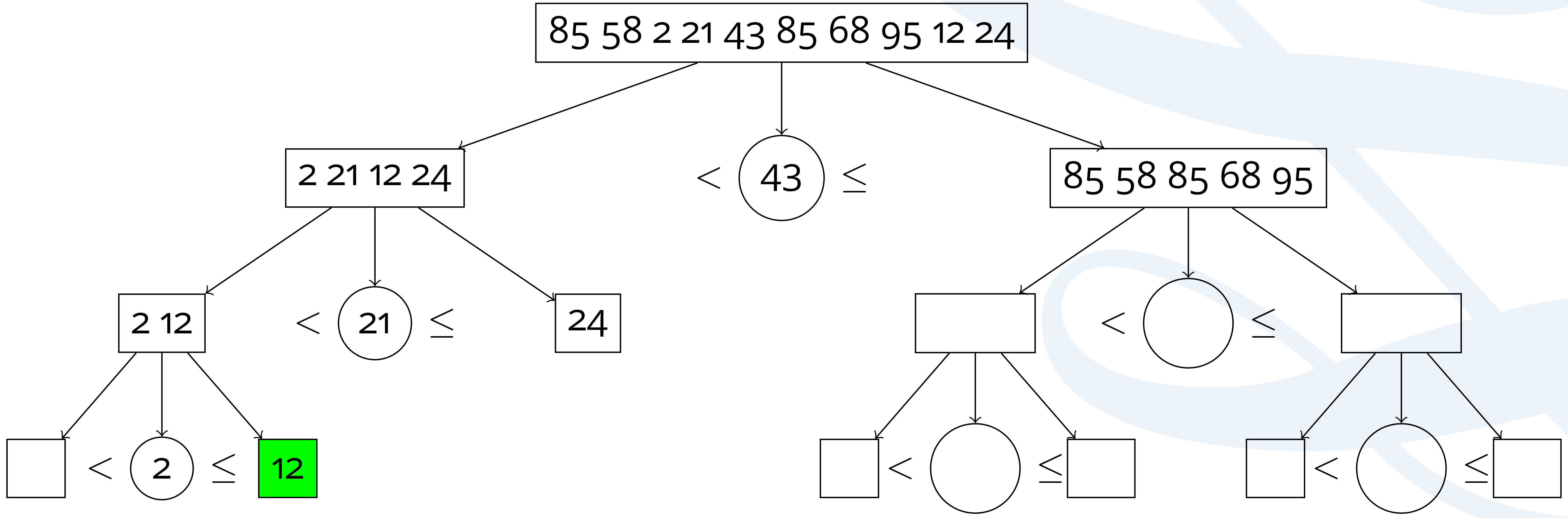
Quicksort III

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.



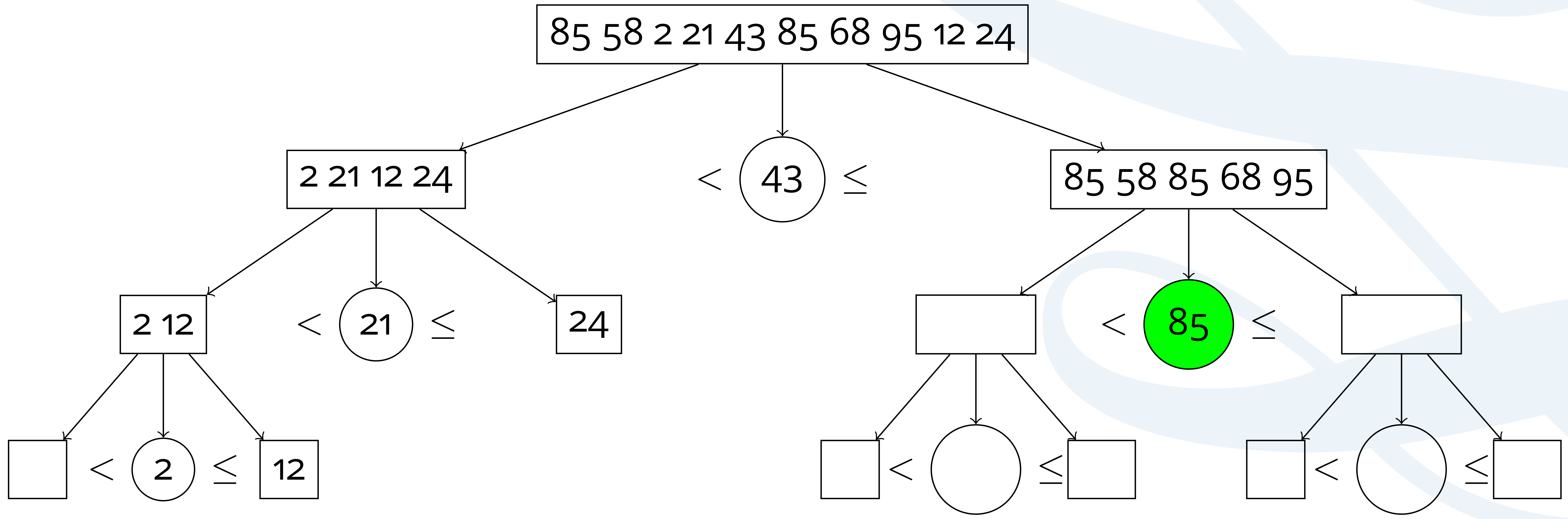
Quicksort III

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.



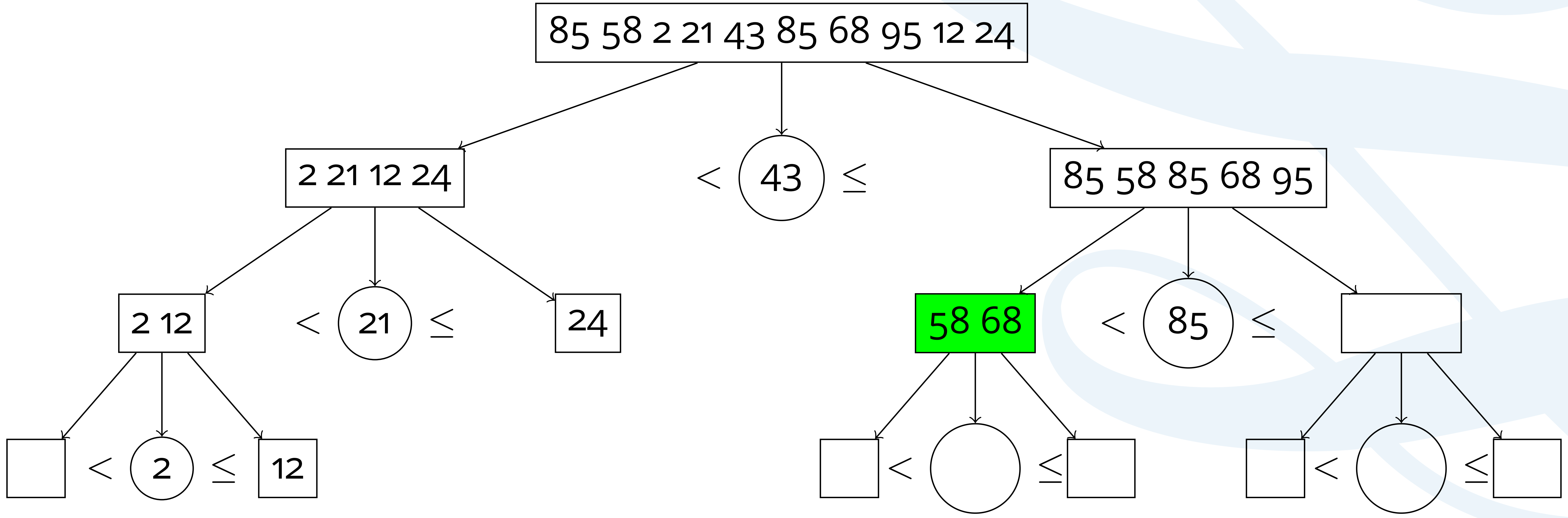
Quicksort III

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.



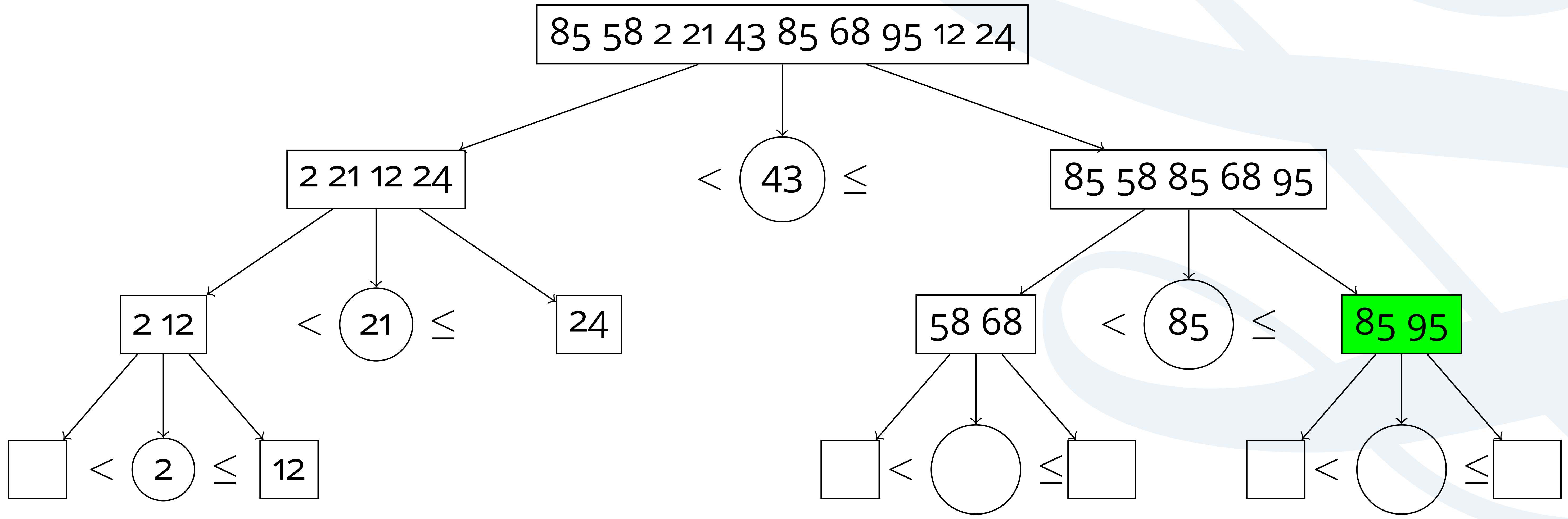
Quicksort III

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.



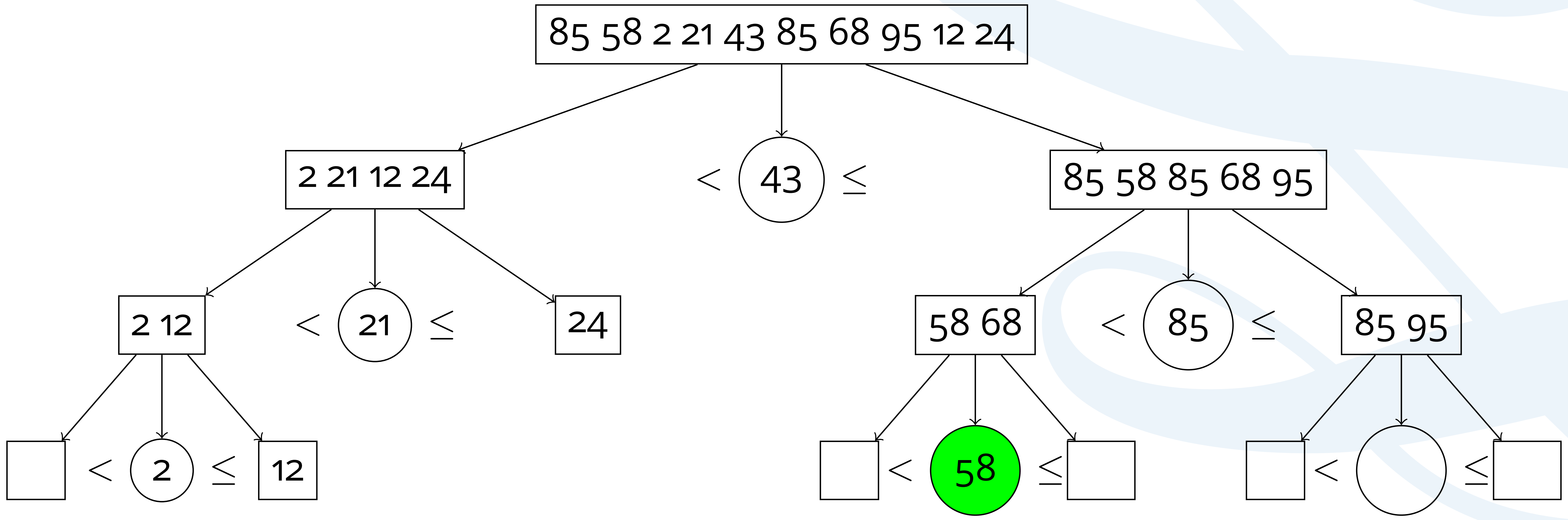
Quicksort III

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.



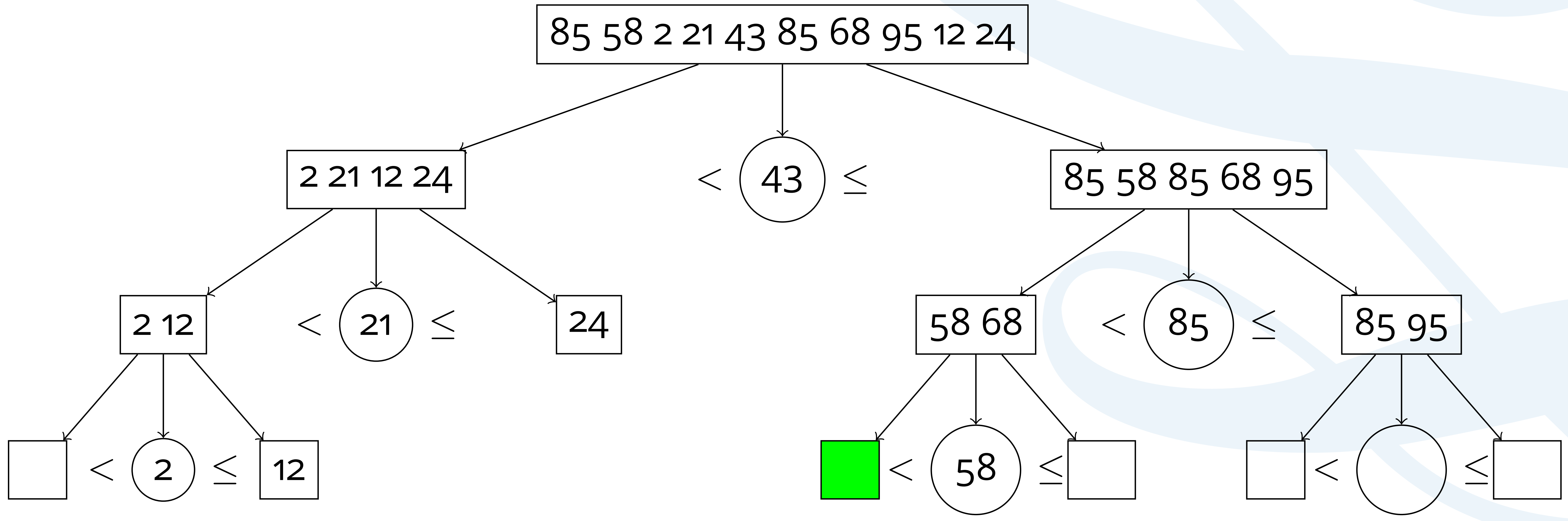
Quicksort III

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.



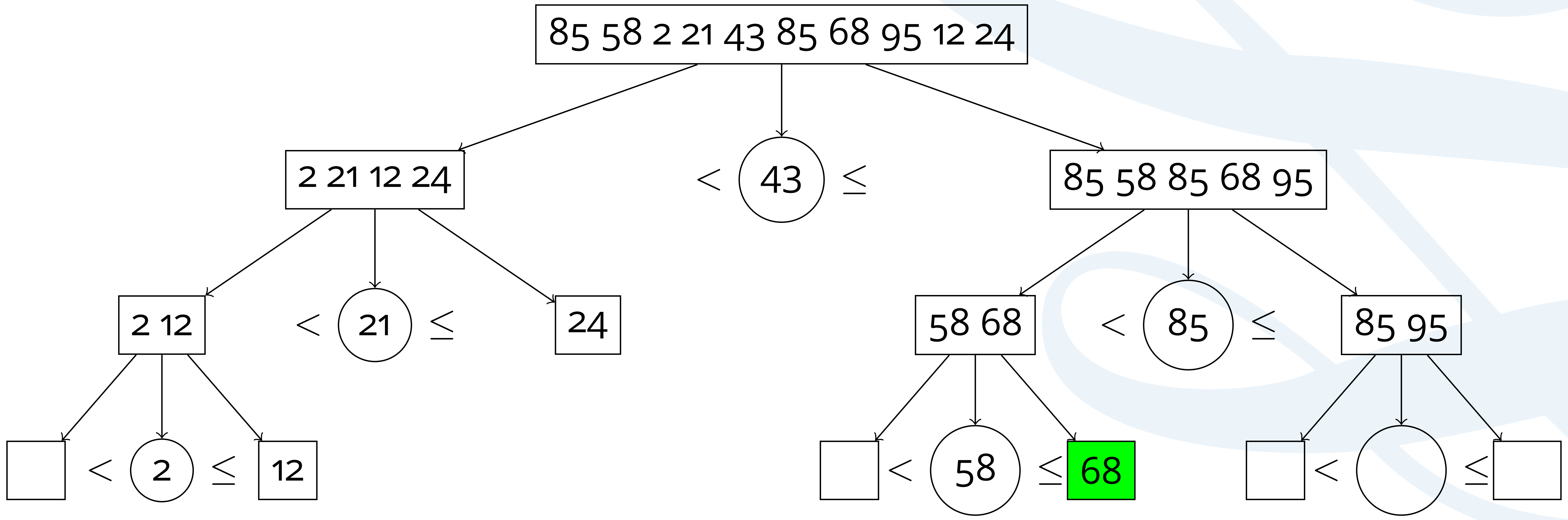
Quicksort III

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.



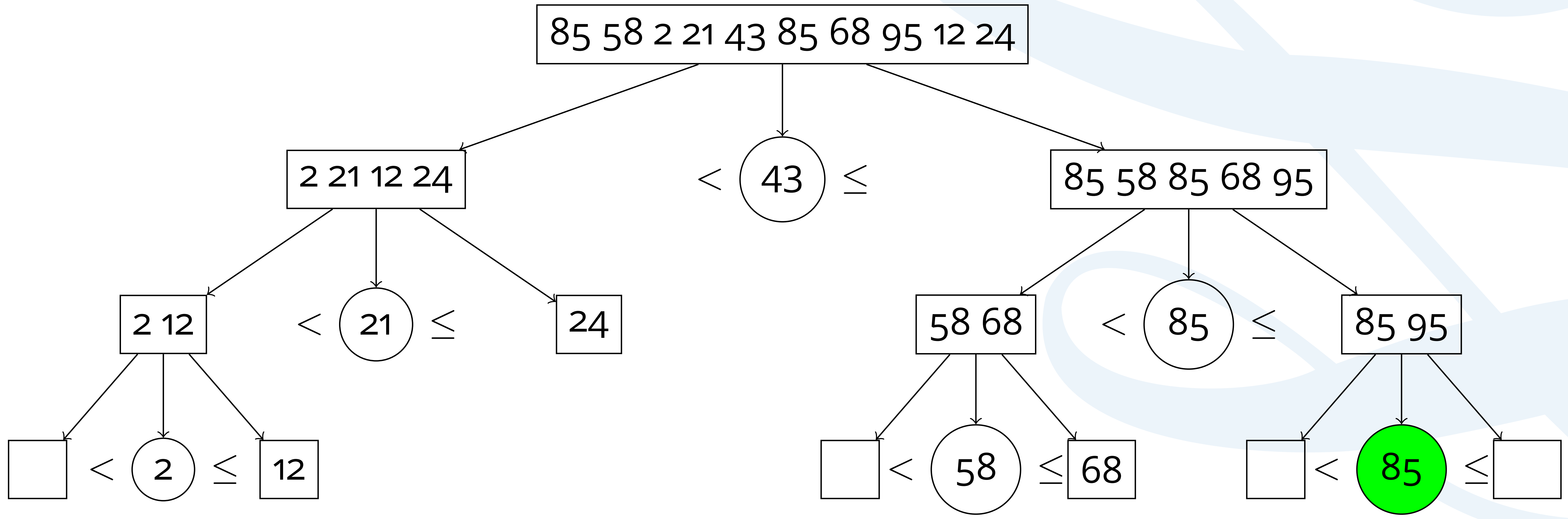
Quicksort III

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.



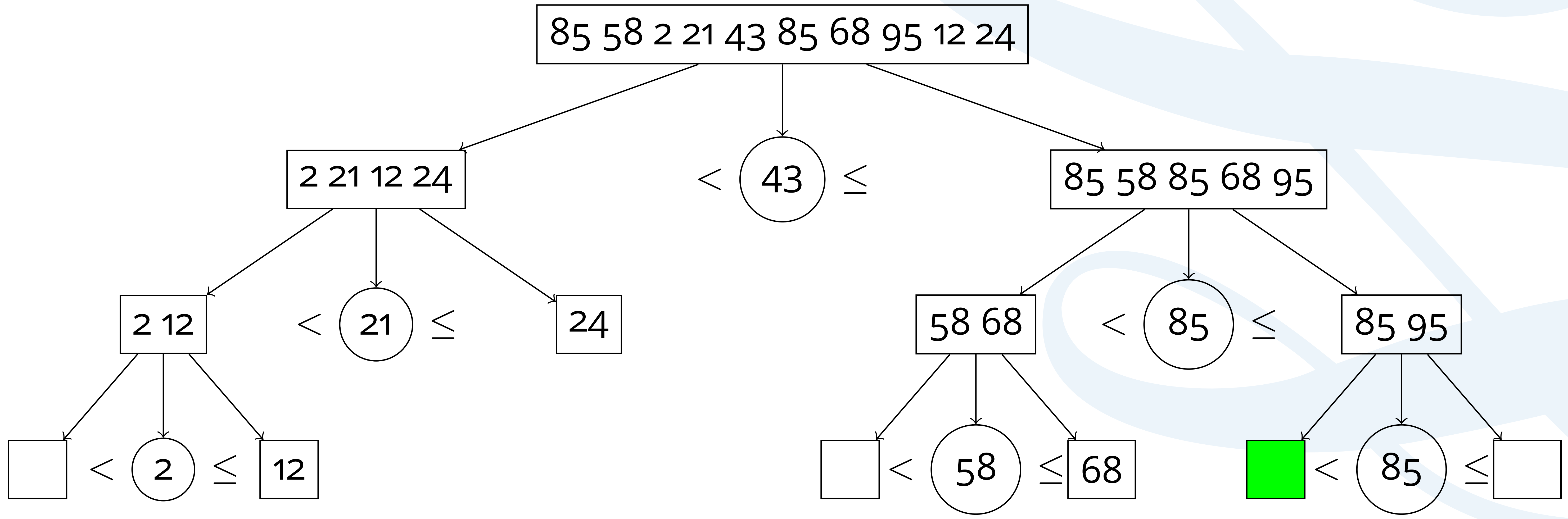
Quicksort III

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.



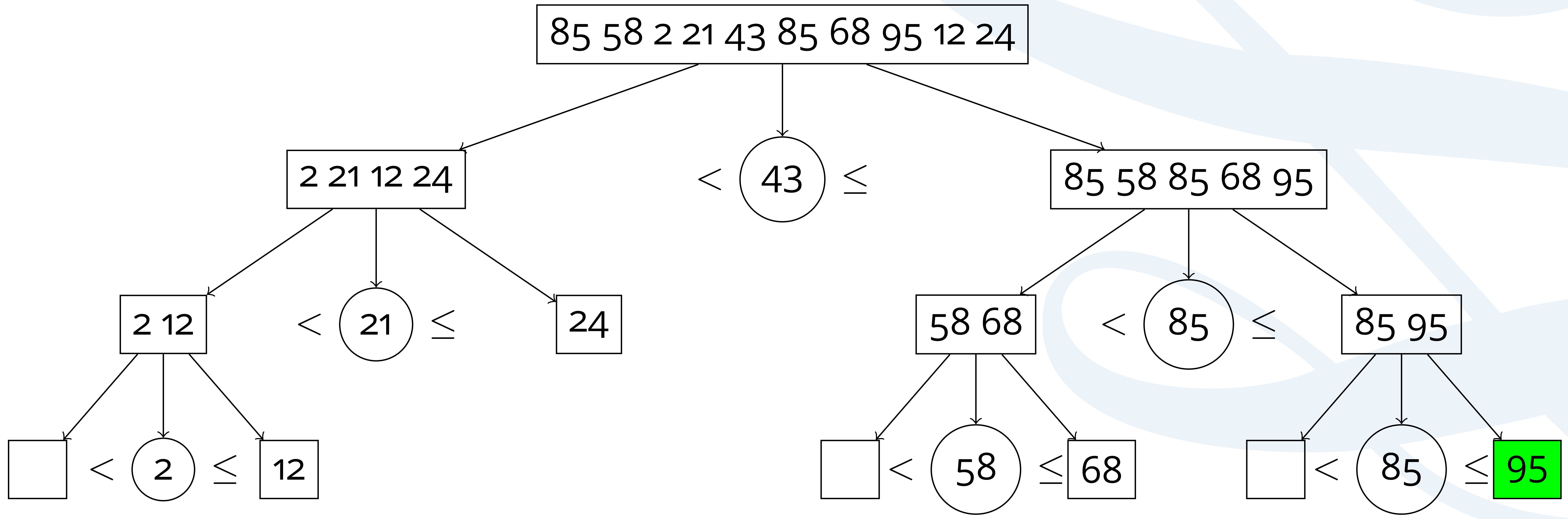
Quicksort III

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.



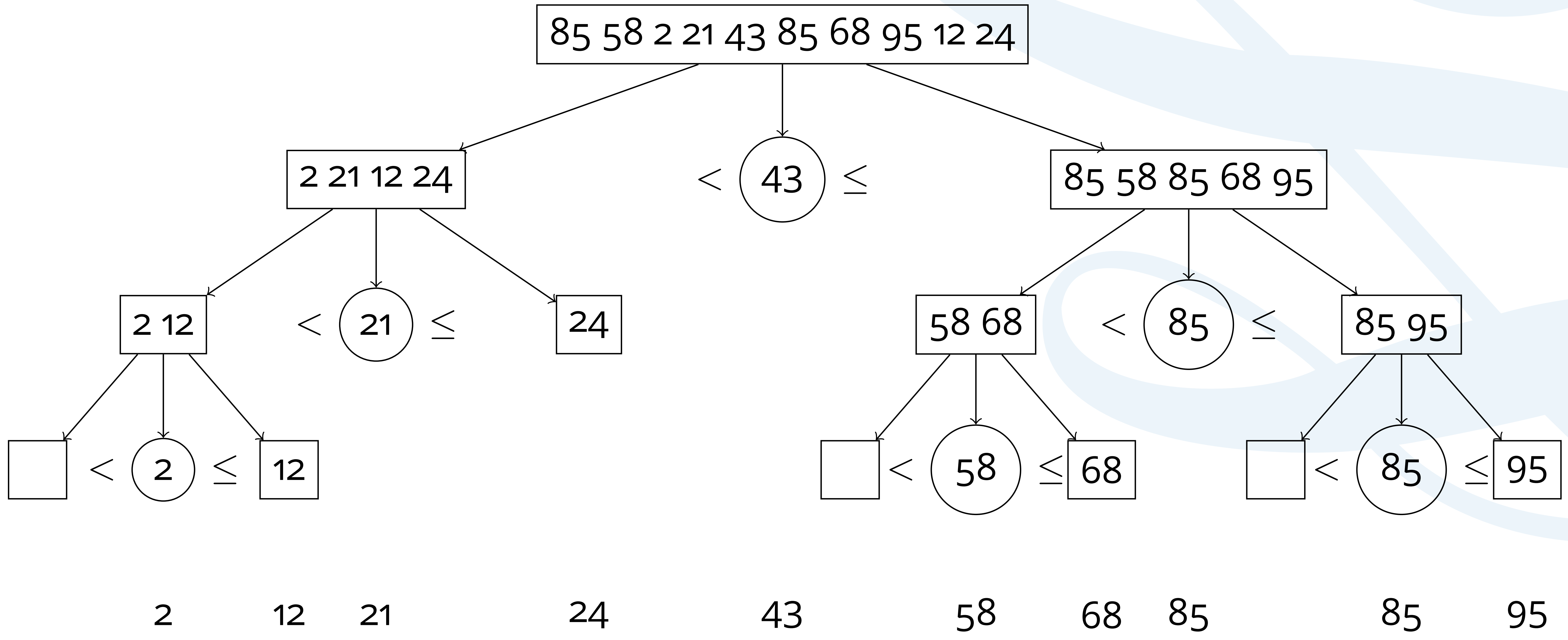
Quicksort III

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 III

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

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

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

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.

Recap

- Many sorting algorithms.

The End