

Sorting algorithms

Coventry University

Introduction

Bubblesort

Stable sort

In-place

Selection sort

Other
algorithms

Quicksort

Divide & Conquer

Comparing

Quiz

Recap

- 1 Introduction
- 2 Bubblesort
 - Stable sort
 - In-place
- 3 Selection sort
- 4 Other algorithms
- 5 Quicksort
 - Divide & Conquer
- 6 Comparing
- 7 Quiz
- 8 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 (Field Of View) calculations.
- Route planning.

Lots of different algorithms, different ways to achieve the same thing.

- Going to be looking at several common/well known algorithms.
 - Bubblesort.
 - Selection sort.
 - Quick sort.
- Comparing and contrasting, advantages and disadvantages.

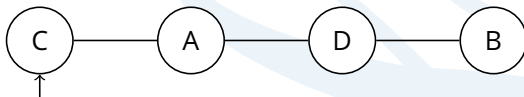
Very simple sort.

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

Pass 1

Very simple sort.

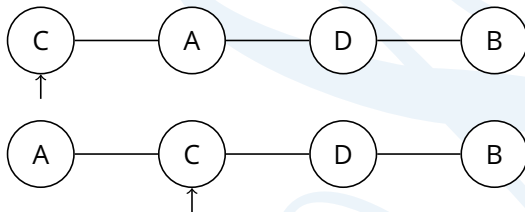
- Compares each item to the next in the sequence.
 - Swap items if in wrong order.



Pass 1

Very simple sort.

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

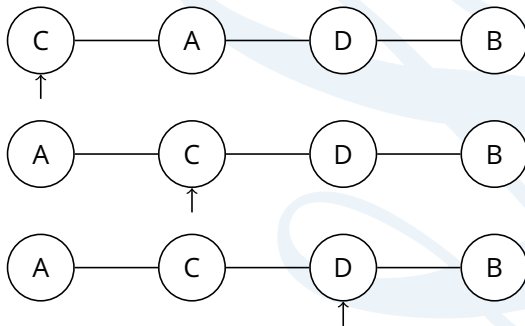


Pass 1

Very simple sort.

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

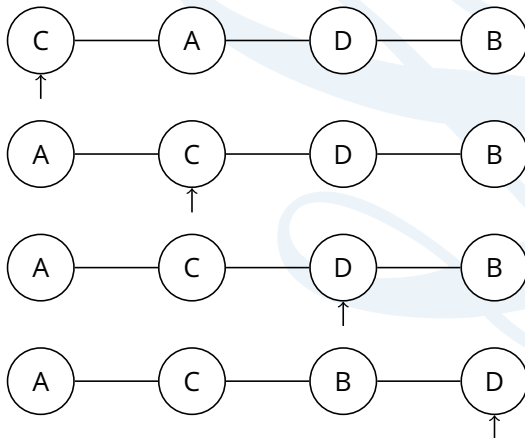
Pass 1



Very simple sort.

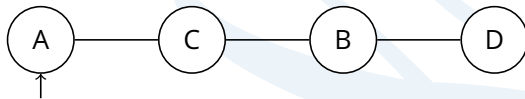
- Compares each item to the next in the sequence.
 - Swap items if in wrong order.

Pass 1



Iterating over the sequence once isn't typically enough.

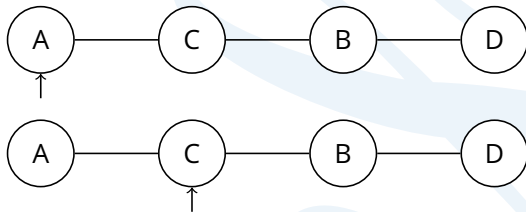
- Keep iterating over the sequence until elements are sorted.



Pass 2

Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

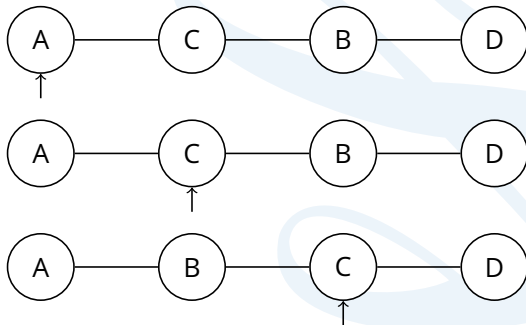


Pass 2

Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

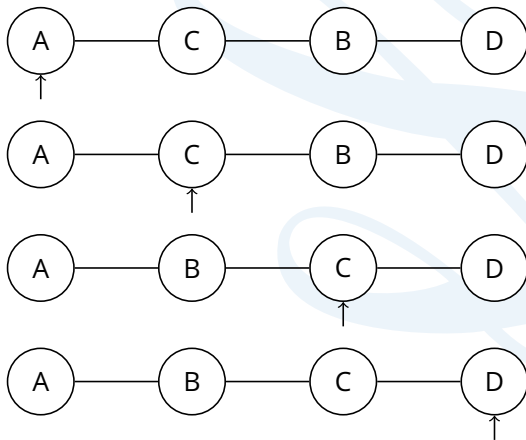
Pass 2



Iterating over the sequence once isn't typically enough.

- Keep iterating over the sequence until elements are sorted.

Pass 2



Introduction

Bubblesort

Stable sort

In-place

Selection sort

Other
algorithms

Quicksort
Divide & Conquer

Comparing

Quiz

Recap

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

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

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

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

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

Bubble sort is what's known as a 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. you are sorting people by height and then sorting them by surname.
 - People with the same surname would still be in height order.
 - Can have performance benefits.

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 slower though.
- 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.

One of the simplest sorting algorithms.

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

One of the simplest sorting algorithms.

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

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, 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, 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, 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.

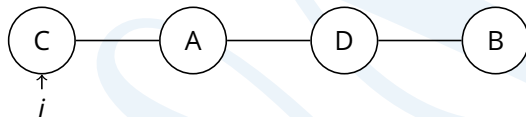
So sorting algorithms have 3 $O()$ values.

Break

- Divides sequence into sorted and unsorted regions.
 - Stable/Unstable, depends on implementation.
 - 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

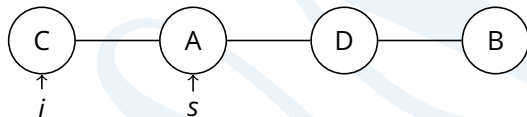
C



- 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

C

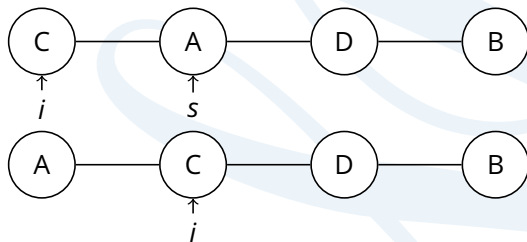


- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.

Selection sort II

C

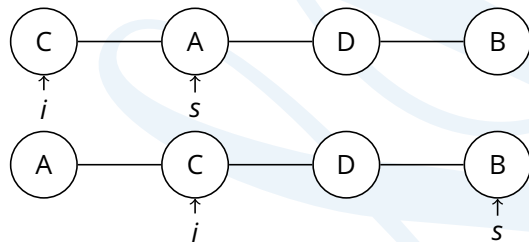
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



Selection sort II

C

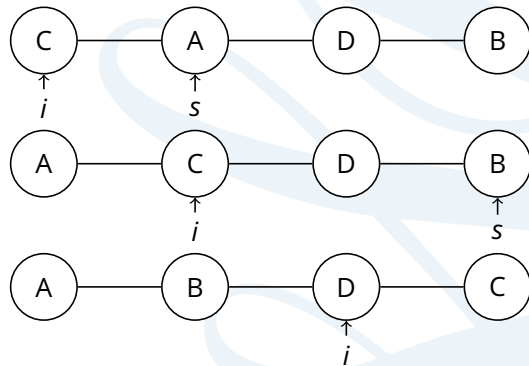
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



Selection sort II

C

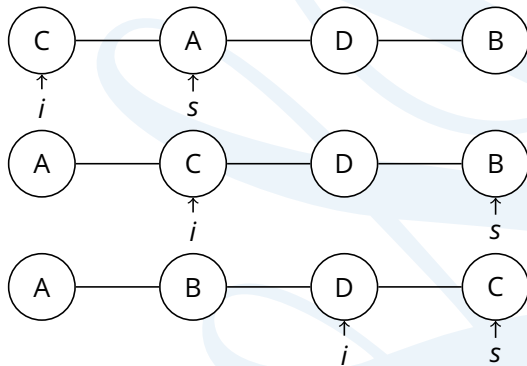
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



Selection sort II

C

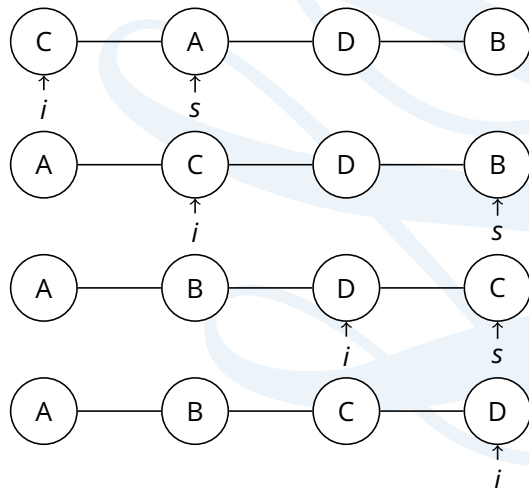
- 1 Iterate over sequence.
- 2 For each element search the remaining elements on its right for the smallest value.
- 3 Swap smallest element with current element.



Selection sort II

C

- 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.



Bubblesort is $O(n^2)$ worst and average case .

Selection sort is $O(n^2)$ worst and average case.

- Selection sort is generally faster than bubble.
 - But have same $O()$ complexity.
 - What?

Bubblesort is $O(n^2)$ worst and average case .

Selection sort is $O(n^2)$ worst and average case.

- Selection sort is generally faster than bubble.
 - But have same $O()$ complexity.
 - What?
- $O()$ notation describes how an algorithm will grow.
- Not good at absolute performances.
- Selection sort typically does fewer comparisons and swaps than bubblesort.
 - Therefore typically faster.
- Best case bubblesort is $O(n)$, selection is $O(n^2)$.
 - So is occasionally faster.

Many sorting algorithms

- Different trade-offs, performances.
- 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

Break

<https://www.youtube.com/watch?v=ZZuD6iUe3Pc>

Neither bubble or selection sort are very good.

- Simple algorithms but slow.
- Not (typically) used in real code.

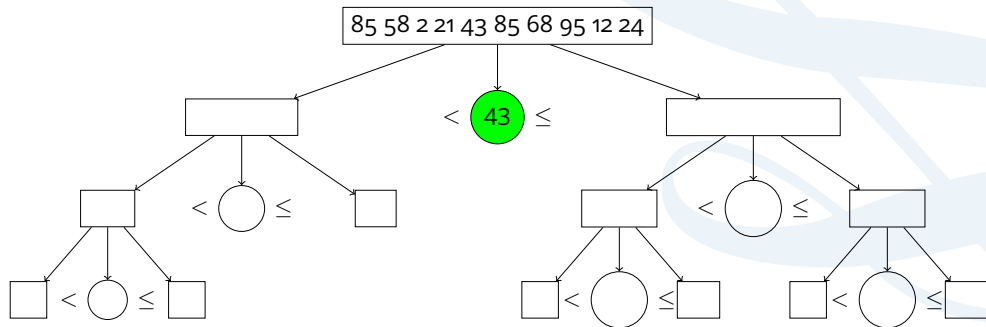
One of the fastest sorting algorithms.

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

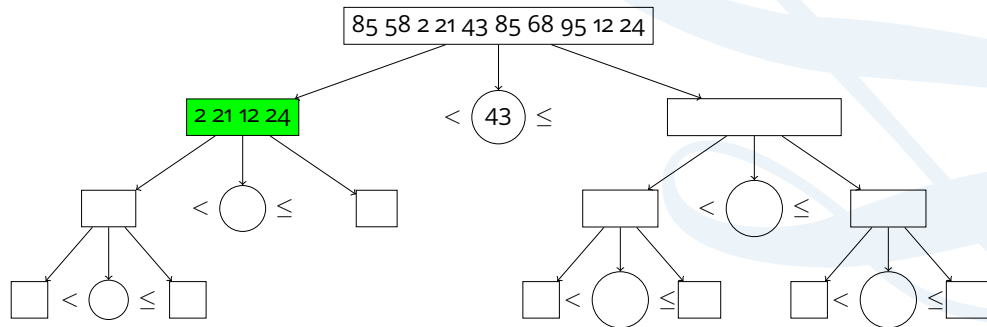
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.

-

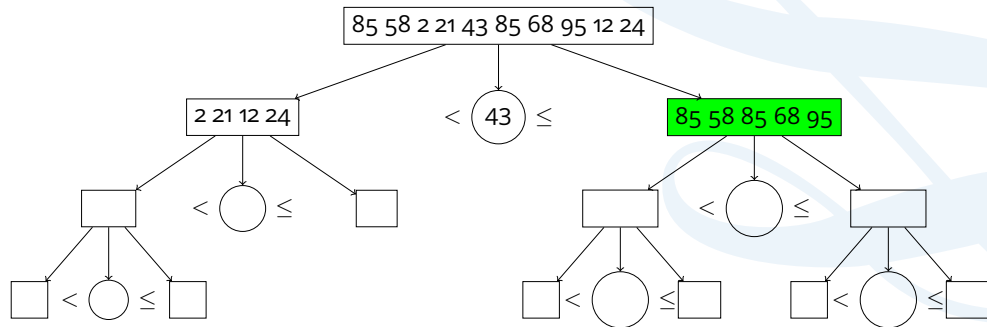
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



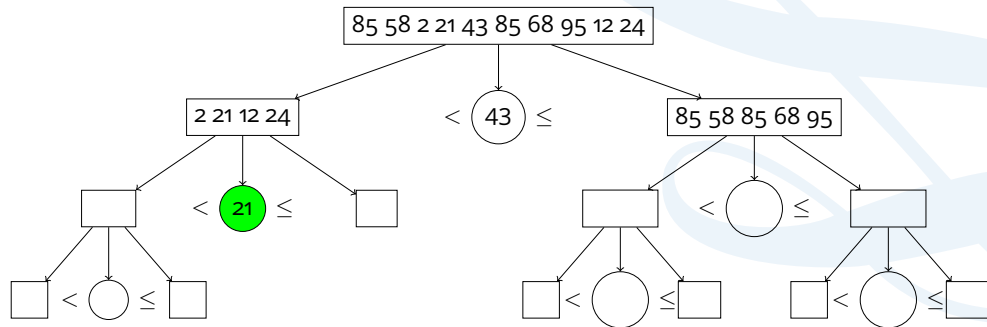
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



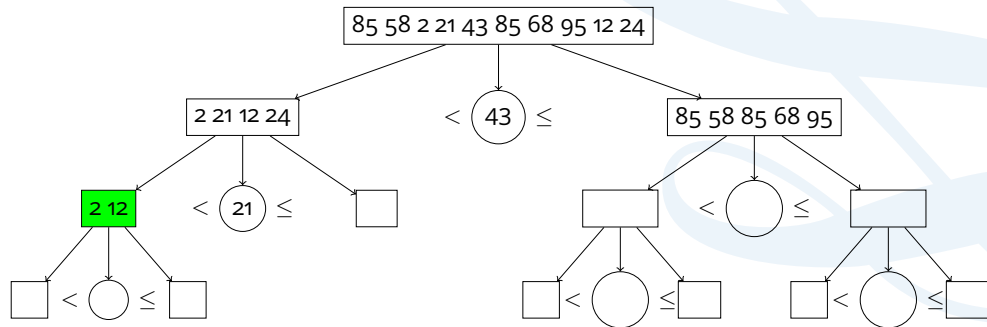
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



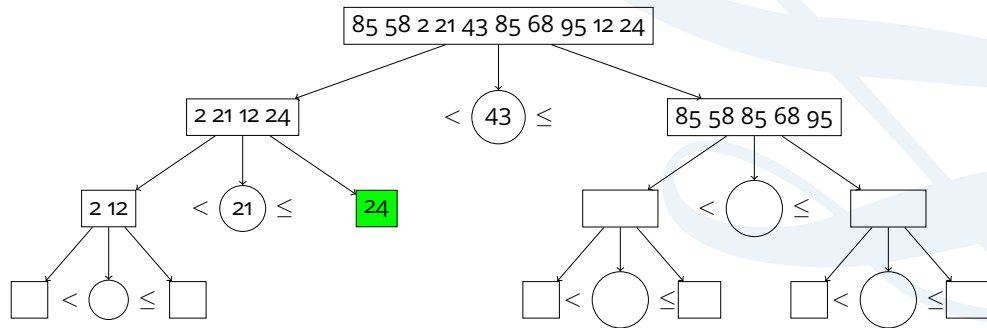
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



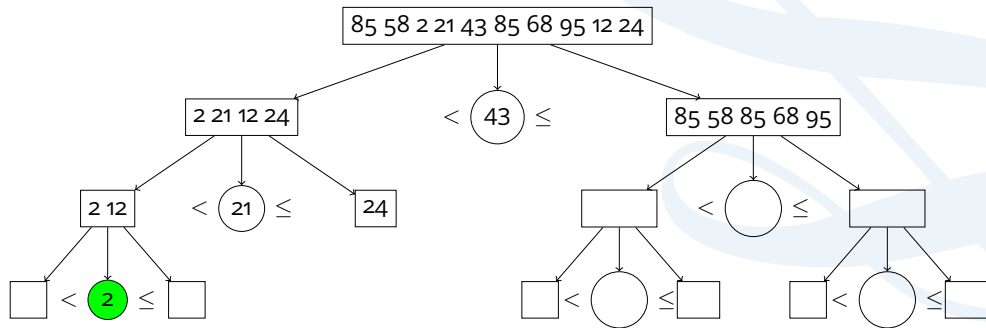
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



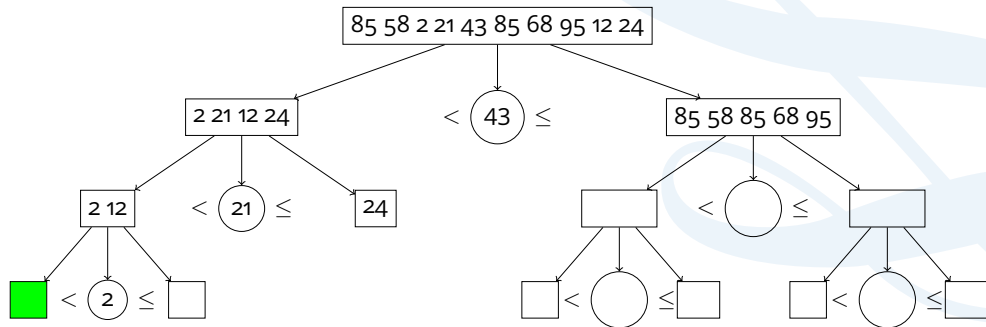
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



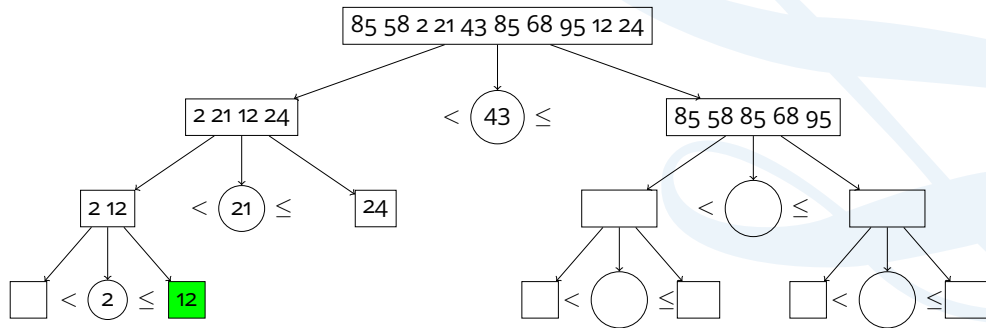
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



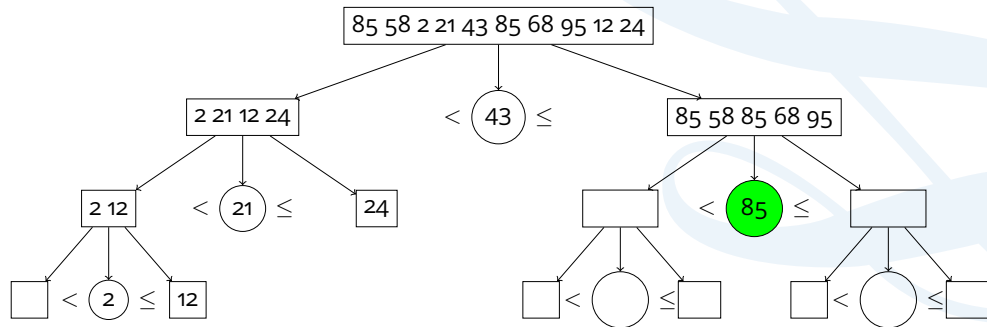
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



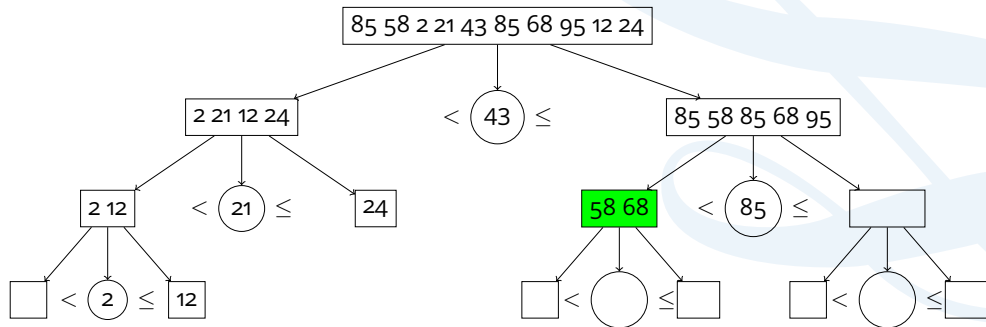
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



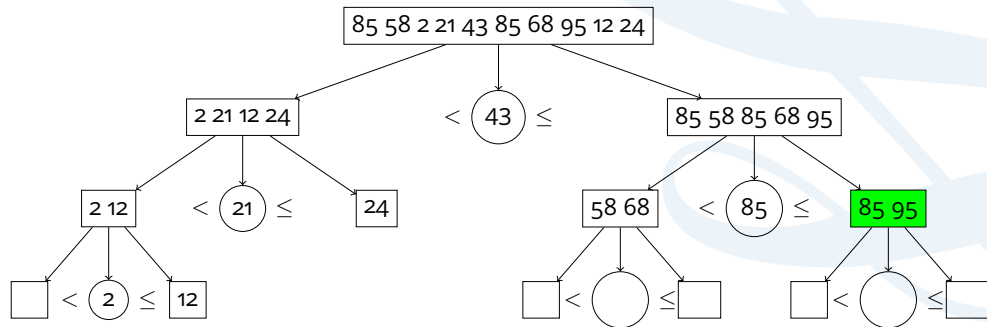
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



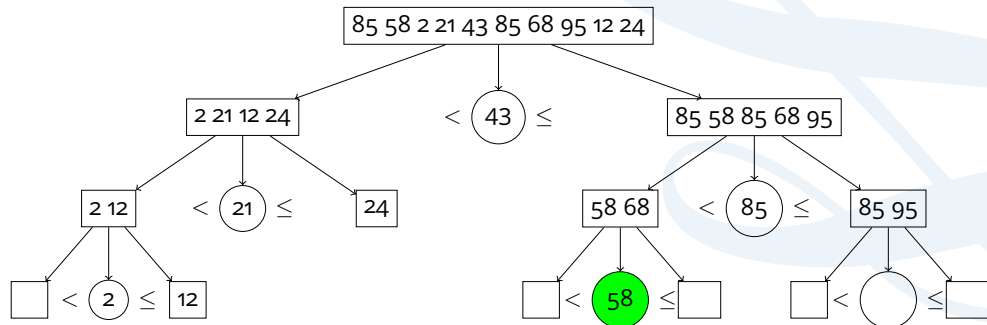
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



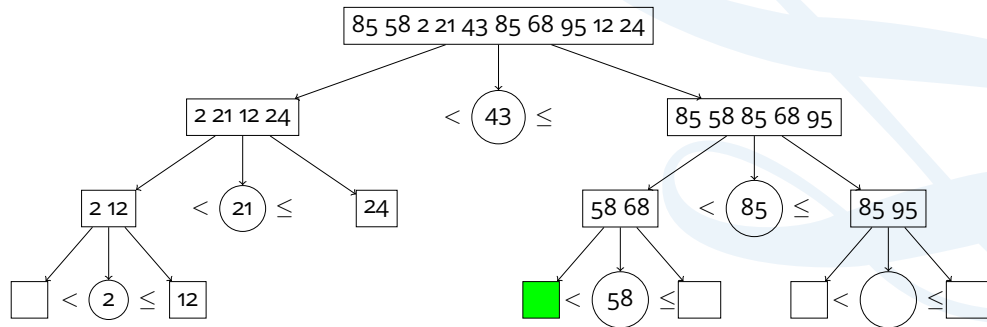
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



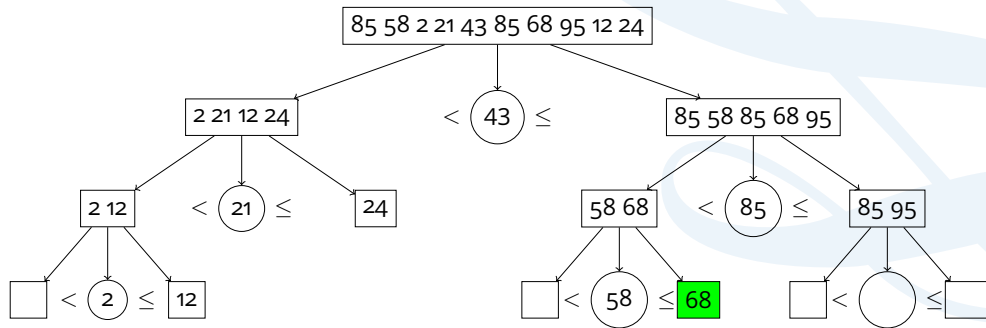
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



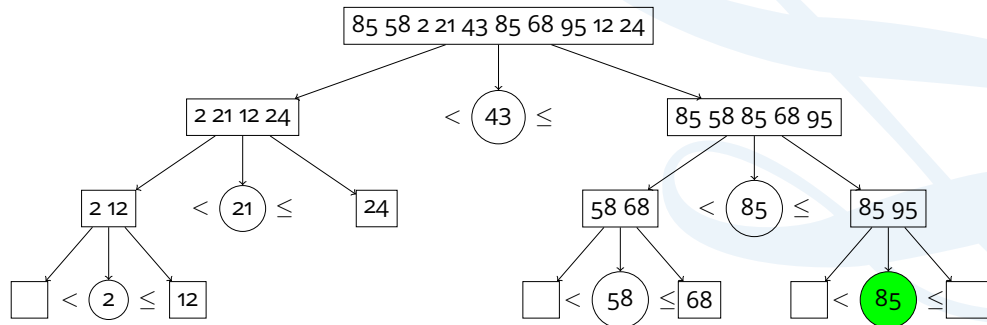
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



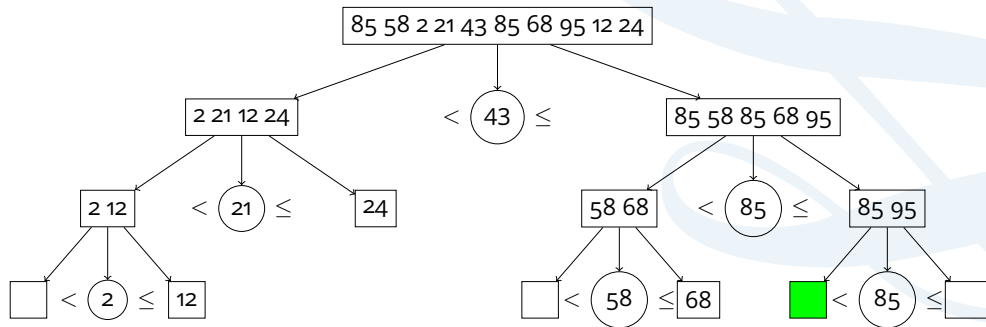
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



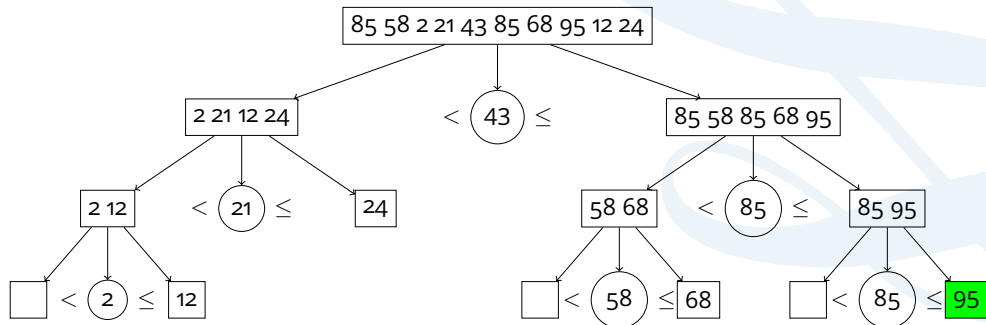
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



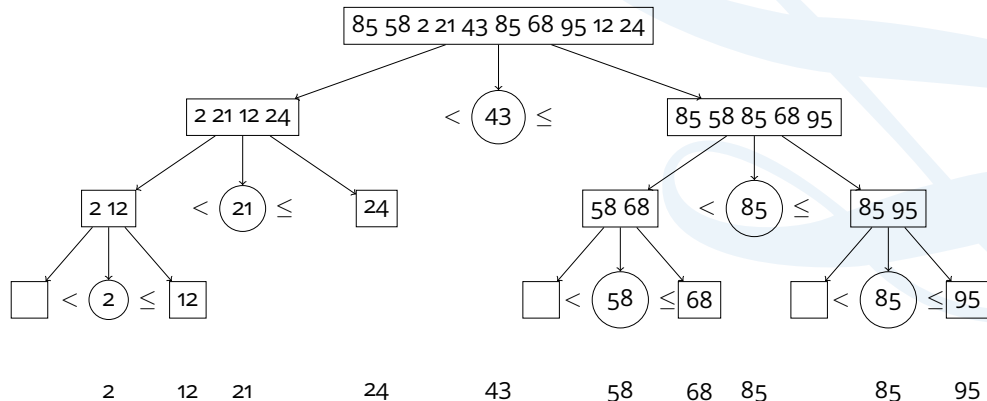
- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq pivot in another group.
- 4 Treat each group as a new sequence and repeat from step 1.



- 1 Select a value from the sequence, this is the pivot.
- 2 Put all values $<$ pivot in one group.
- 3 Put all values \geq 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 \geq 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.

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.

- Each sub problem can be processed separately.

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.



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?



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)?

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.

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 .

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.

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.

Sorting

Introduction

Bubblesort

Stable sort

In-place

Selection sort

Other
algorithms

Quicksort

Divide & Conquer

Comparing

Quiz

Recap

Quiz

Bubblesort performs best (has $O(n)$ performance) when

- The sequence is already in order.
- The sequence is in a random order.
- The sequence is in reverse order.
- The sequence contains a few distinct values that are repeated.

Bubblesort performs best (has $O(n)$ performance) when

- The sequence is already in order.
- The sequence is in a random order.
- The sequence is in reverse order.
- The sequence contains a few distinct values that are repeated.

Divide & Conquer algorithms work by _____

- Dividing the problem in half.
- Breaking problems down into smaller easier problems.
- Simplifying the code so that they run faster.
- Invading Czechoslovakia.

Divide & Conquer algorithms work by _____

- Dividing the problem in half.
- Breaking problems down into smaller easier problems.
- Simplifying the code so that they run faster.
- Invading Czechoslovakia.

Which of the following algorithms are NOT divide & conquer?

- Bubblesort.
- Bubblesort and selection sort.
- Selection sort.
- Quicksort.

Which of the following algorithms are NOT divide & conquer?

- Bubblesort.
- Bubblesort and selection sort.
- Selection sort.
- Quicksort.

Which algorithm uses a pivot value to repeatedly halve the sequence?

- Bubblesort.
- Selection sort.
- Quicksort.
- All of the above.

Which algorithm uses a pivot value to repeatedly halve the sequence?

- Bubblesort.
- Selection sort.
- Quicksort.
- All of the above.

The worst sorting algorithm is _____

- Bubblesort.
- Bogo sort.
- Sleep sort.
- Selection sort.

The worst sorting algorithm is _____

- Bubblesort.
- **Bogo sort.**
- Sleep sort.
- Selection sort.

Everyone

- Sorting algorithms are key to understanding many important concepts.
 - I.e. Binary Search Trees.
- Key to writing efficient code.
- Key to understanding memory/processor trade offs.
- Useful in teaching algorithmic thinking.
 - Algorithm design.
 - Comparing and contrasting different algorithms.
 - Divide and Conquer concepts.
- Employability skill, popular questions for programming interviews.

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

- Complete the yellow Codio exercises for this week.
- Attempt the green Codio exercises for next week.
- If you have spare time attempt the red Codio exercises.
- If you are having issues come to the PSC.
<https://gitlab.com/coventry-university/programming-support-lab/wikis/home>

The End