Recap: Apache Maven

- Maven to organise your dependency
- Works across computers
  - because dependencies are downloaded automatically from Maven central
- Formal definition of the build process
  - Replication
  - Documentation

SUMMER-HYPE-TAG!!1!!1!
- Freitag den 30.06. – 14 Uhr
- Albertus-Magnus-Platz
- Verschiedene Challenges (Flunkyball, Wasserbomben, Jeopardy und mehr)
→ Mit Hauptgewinn!
Alle Updates auf Insta: fsinfokoeln
fs-medien.info

Session 10: Sorting and Functional Objects

Fortgeschrittene Programmierung (Java 2)

Nils Reiter
nils.reiter@uni-koeln.de

June 21, 2023

Introduction

- Regularly appearing task
- Examples:
  - Sort files by size, name, ...
  - Sort words in a text by frequency
  - Sort documents by length
  - ...

Introduction

- Regularly appearing task
- Examples:
  - Sort files by size, name, ...
  - Sort words in a text by frequency
  - Sort documents by length
  - ...
- What do we need to sort things?
  - Collection to be ordered
  - Defined order between the elements, formalized as an order between two elements
    - E.g.: $5 < 7$, “A comes before B”, “Document X comes before document Y”, ...
    - Sort criterion
  - Sort algorithm

Defining Pairwise Order

Convention

$$\text{compare}(o1, o2) = \begin{cases} -1 & o1 \text{ before } o2 \\ 0 & o1 \text{ equal to } o2 \\ 1 & o1 \text{ after } o2 \end{cases}$$

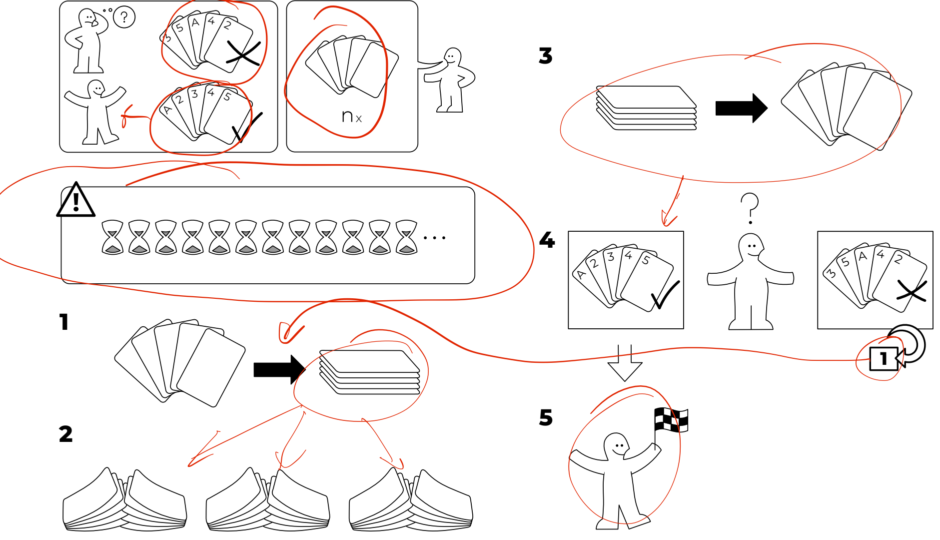
```java
public int compare(int i1, int i2) {
  if (i1 < i2) {
    return -1;
  } else if (i1 == i2) {
    return 0;
  }
  return 1;
}


// or, shorter:
public int compare(int i1, int i2) {
  return Math.signum(i1-i2);
}
```

BOGO SÖRT

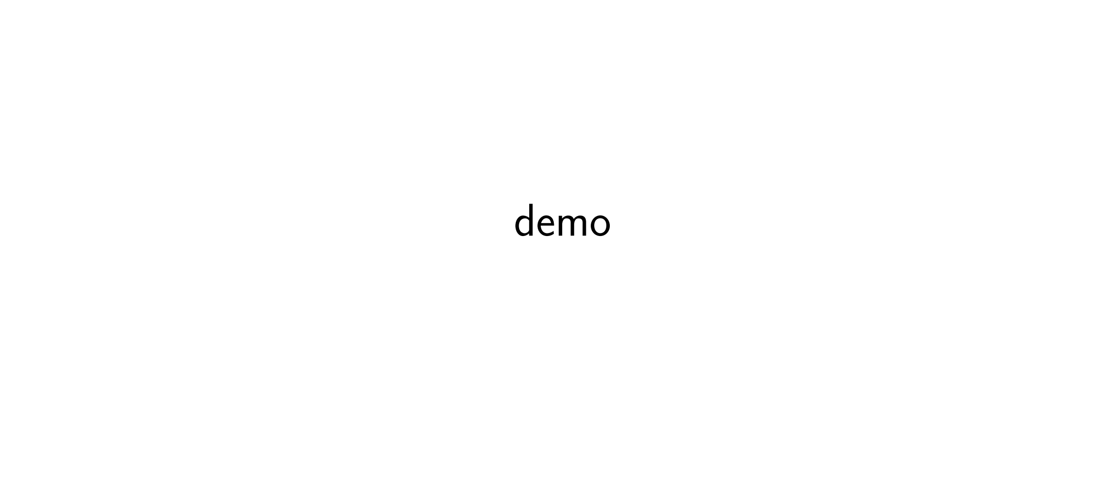
idea-instructions.com/bogo-sort/
v1.1, CC by-nc-sa 4.0

IDEA



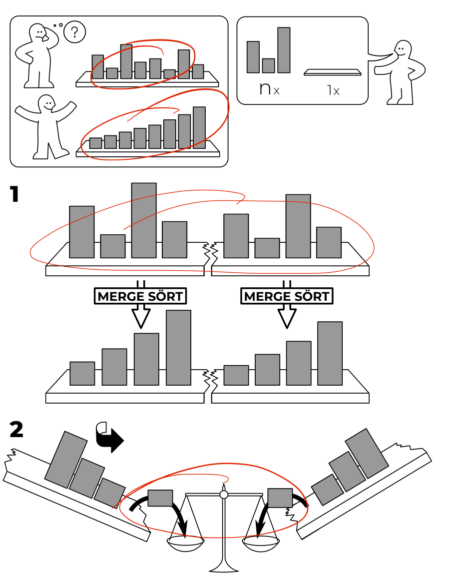
Sort Algorithm 1

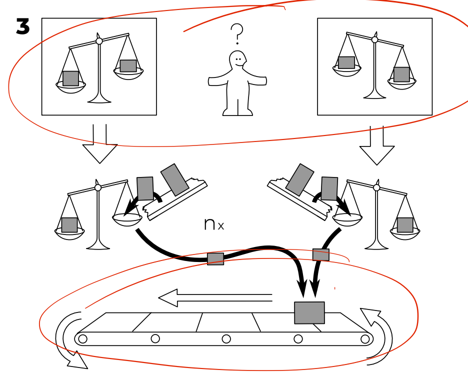
1. Input: Unsorted list L; output: sorted list L'
2. Pick top element from C from L
3. If L' is empty
   - Put C onto L'
4. Else:
   - Put C into L', but at the correct position (according to compare function)
5. Repeat until L is empty

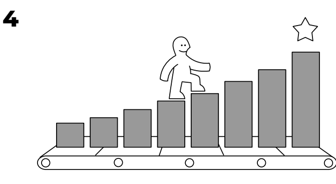

demo

MERGE SÖRT

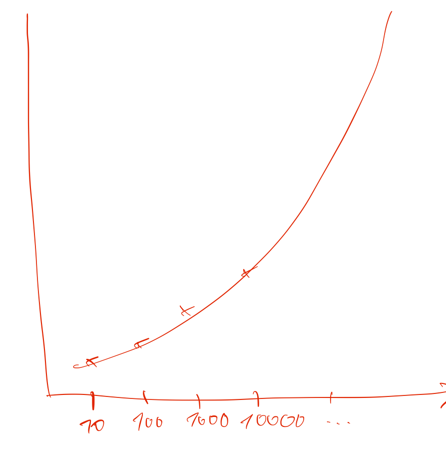
idea-instructions.com/merge-sort/
v1.2, CC by-nc-sa 4.0
IDEA







demo



Sorting in Collections Framework

- `List<E>`: `default void sort(Comparator<? super E> c)`
  - In-place sorting on demand
- `SortedSet<E>` / `SortedMap<E>`:
  - Sorting when adding (set is always sorted, map sorted by keys)

Sorting in Collections Framework

- `List<E>`: `default void sort(Comparator<? super E> c)`
  - In-place sorting on demand
- `SortedSet<E>` / `SortedMap<E>`:
  - Sorting when adding (set is always sorted, map sorted by keys)

`java.util.Comparator<T>`

- Defines a single (non-static) method `compare(o1, o2)`
  - Returns one of -1, 0, 1 for objects ot type `T`
- Allows us to define our own sorting criteria for our own objects
  - E.g., to sort objects of type `Student` by their id number, birth date, ...

Section 2

Functions as Objects

Introduction

- Need to pass code snippets into existing functions
- E.g., sorting criteria into sort functions or filter criteria into search functions
- Basic mechanism
  - Define an interface with very few methods
  - Programmers can implement interfaces with their own code
  - E.g., `java.util.Comparator<T>`

More General Solution: Functional Interfaces

-

More General Solution: Functional Interfaces

- java.util.function
- `Function<T,R>`: Maps from objects of type T to objects of type R
- `Predicate<T>`: Evaluates an object of type T and returns a boolean value
- Bi prefix: Two input objects
- Consumer/supplier: Does not produce output or does not consume input

More General Solution: Functional Interfaces

- java.util.function
- `Function<T,R>`: Maps from objects of type T to objects of type R
- `Predicate<T>`: Evaluates an object of type T and returns a boolean value
- Bi prefix: Two input objects
- Consumer/supplier: Does not produce output or does not consume input

... but why?

- Because we often need to pass such functions as objects to other functions
- And, in combination with a bit of new syntax: It makes programs shorter and easier to read

More Syntactic Sugar

```
1  class InThisClass implements Predicate<Student> {
2    public boolean test(Student s) {
3      return s.getClasses().contains("Java2");
4    }
5  },
6
7  public class Main {
8    public static void main(String[] s) {
9      List<Student> all_students = // ...
10     Predicate<Student> p_inThisClass = new InThisClass();
11     all_students.remove_if(p_inThisClass);
12   }
13 }
```

More Syntactic Sugar

```
1  class InThisClass implements Predicate<Student> {
2    public boolean test(Student s) {
3      return s.getClasses().contains("Java2");
4    }
5  },
6
7  public class Main {
8    public static void main(String[] s) {
9      List<Student> all_students = // ...
10     Predicate<Student> p_inThisClass = new InThisClass();
11     all_students.remove_if(p_inThisClass);
12   }
13 }
```

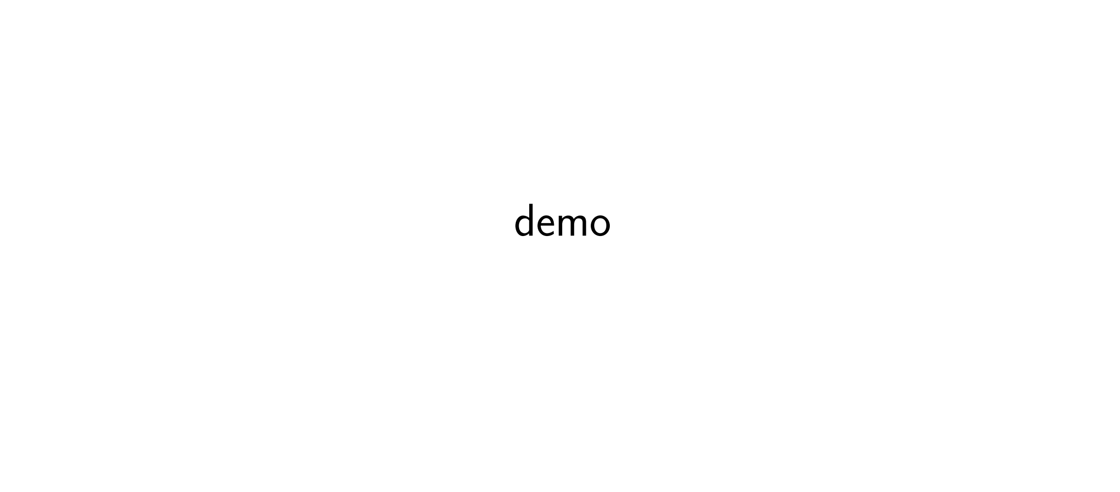
```
1  public class Main {
2    public static void main(String[] s) {
3      List<Student> all_students = // ...
4      Predicate<Student> p_inThisClass = s -> s.getClasses().contains("Java2");
5      all_students.remove_if(p_inThisClass);
6    }
7  }
```

Collections and Streams

- Interface `Collection<E>`
  - defines method `removeIf(Predicate<? super E> filter)`
  - defines method `stream()`

Collections and Streams

- Interface `Collection<E>`
  - defines method `removeIf(Predicate<? super E> filter)`
  - defines method `stream()`
- Easy to implement and read way of dealing with collections


demo

Exercise

https://github.com/idh-cologne-java-2-summer-2023/exercise-12