

# Praktische Informatik 2

## Sortieren

Thomas Röfer

Cyber-Physical Systems  
Deutsches Forschungszentrum für  
Künstliche Intelligenz

Multisensorische Interaktive Systeme  
Fachbereich 3, Universität Bremen



## Sortieren

- Eine sortierte Folge **S** ist eine Permutation einer Folge **T**: **S = perm(T)**
- **S** wird als **aufsteigend sortiert** bezeichnet, wenn jedes Element **s<sub>i</sub>** größer oder gleich seinem Vorgänger in der Folge ist
  - **s<sub>0</sub> ≤ s<sub>1</sub> ≤ ... ≤ s<sub>N-1</sub>**
- **S** wird als **absteigend sortiert** bezeichnet, wenn jedes Element **s<sub>i</sub>** kleiner oder gleich seinem Vorgänger in der Folge ist
  - **s<sub>0</sub> ≥ s<sub>1</sub> ≥ ... ≥ s<sub>N-1</sub>**

## Sortieren

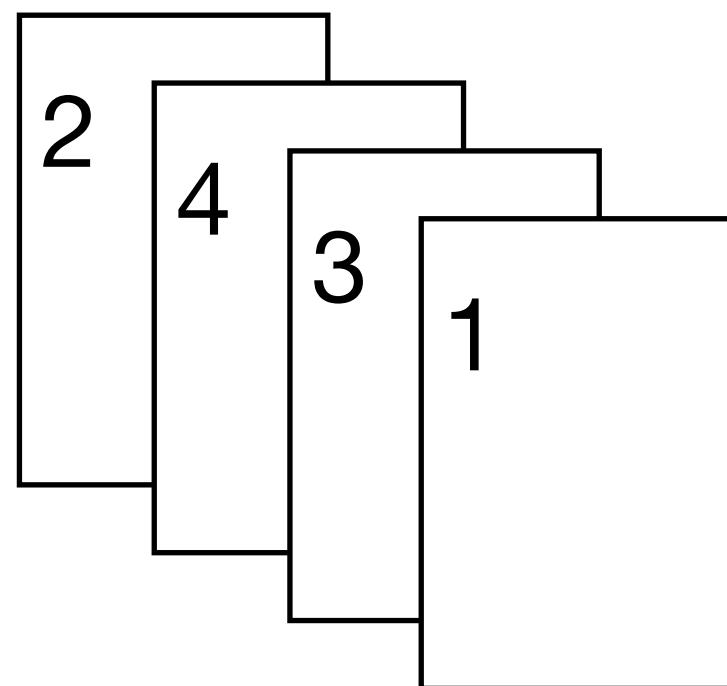
- Es wird immer nach einem **Sortierkriterium** sortiert (**Sortierschlüssel**)
  - Der Sortierschlüssel kann sich aus mehreren Teilen zusammensetzen, z.B. Name, Vorname, Telefonnummer
  - Je komplexer der Sortierschlüssel, desto komplexer sind die zum Sortieren notwendigen Vergleiche
- Zu einer Menge von Daten können mehrere sortierte **Indizes** vorgehalten werden
- Ein Sortierverfahren ist **stabil**, wenn es die Reihenfolge von Elementen mit dem gleichen Sortierschlüssel-Wert nicht ändert

## Naives Sortieren

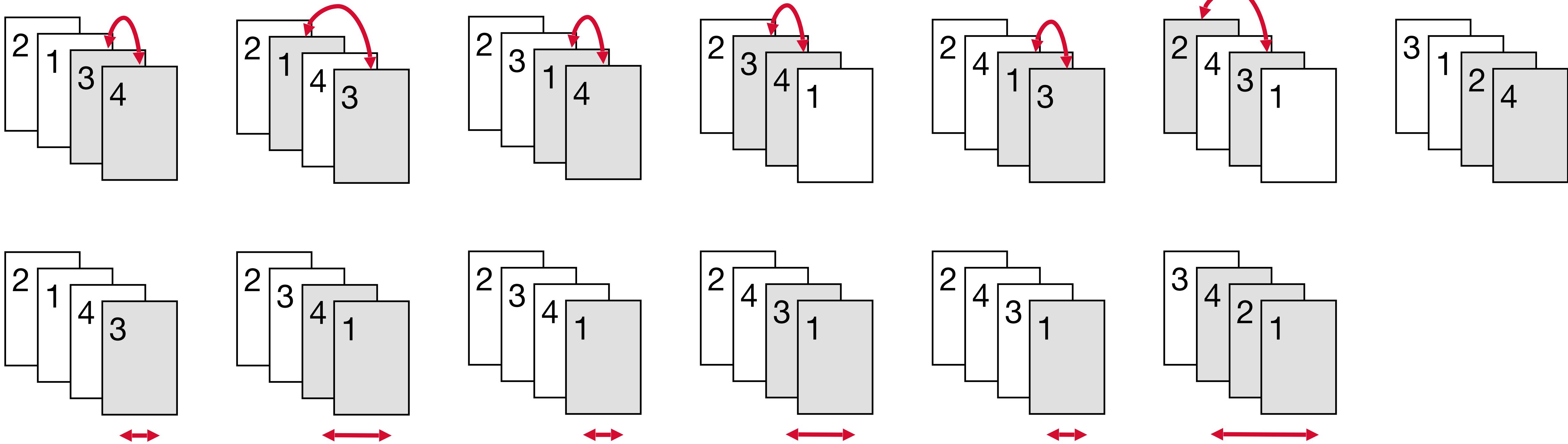
- Erzeuge Permutationen, bis eine entsprechend des Kriteriums sortiert ist
- Bester Fall:  **$O(N)$**  (vorsortiert)
- Schlechtester Fall:  **$O(\frac{1}{2}N \cdot N!) = O(N \cdot N!)$** 
  - Erst die letzte Permutation ist sortiert
  - Nicht-Sortiertheit wird im Mittel nach der Hälfte der Vergleiche festgestellt
- Durchschnittlicher Fall:  **$O(\frac{1}{2}N \cdot \frac{1}{2}N!) = O(N \cdot N!)$** 
  - Sortierte Folge wird nach der Hälfte der Permutationen gefunden
  - Nicht-Sortiertheit wird im Mittel nach der Hälfte der Vergleiche festgestellt

## Permutieren nach Dijkstra

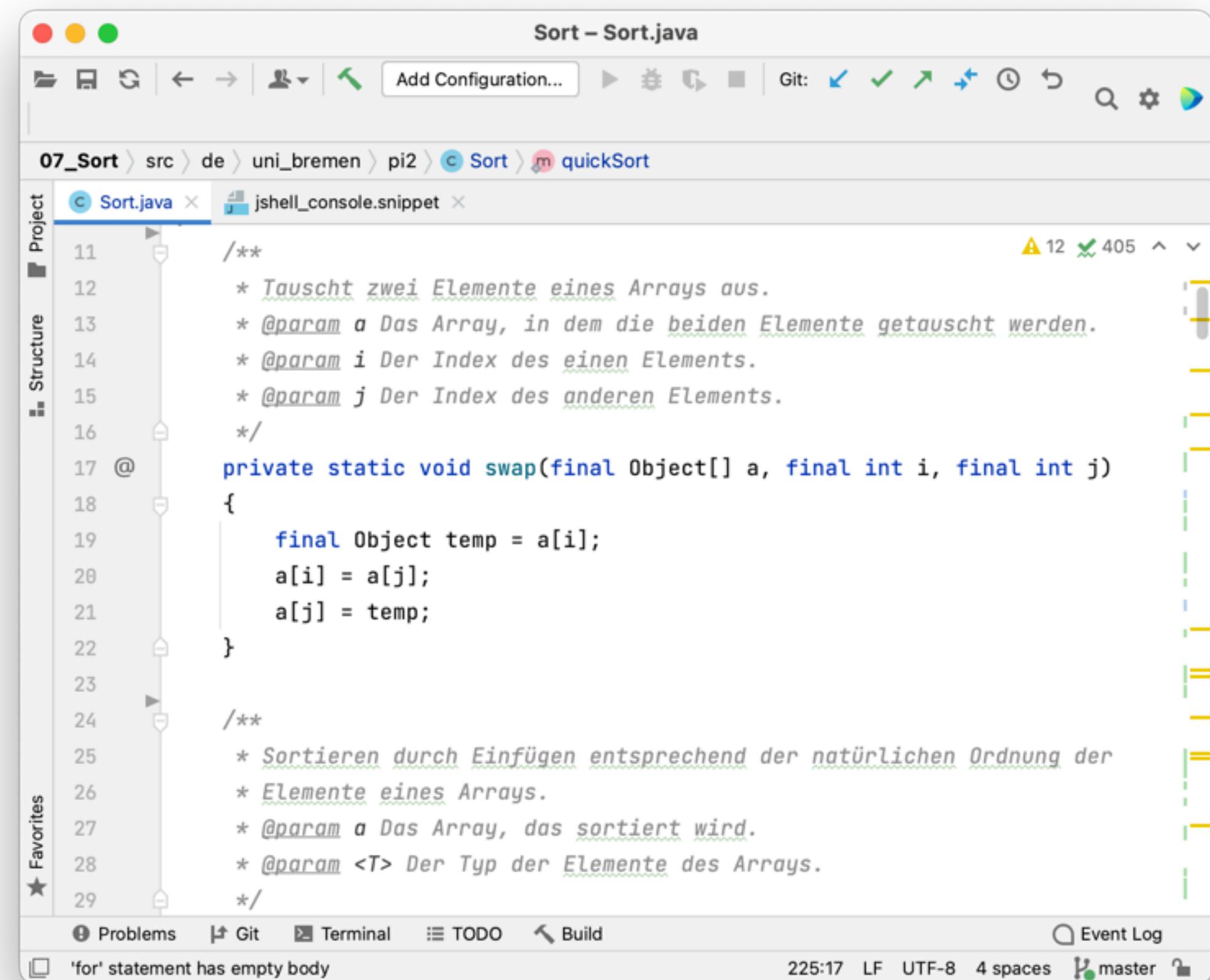
- Eine Funktion, die aus einer Permutation die jeweils nächste erzeugt
- Benötigt eine Ordnungsrelation zwischen den Elementen der Folge
- Algorithmus für eine Folge  $a_0 \dots a_{n-1}$ 
  - Finde das letzte  $a_i$ , das kleiner als  $a_{i+1}$  ist
  - Finde das letzte  $a_j$ , das größer als  $a_i$  ist, und vertausche die beiden
  - Drehe die Reihenfolge der Elemente ab  $a_{i+1}$  um
  - Durchschnittlicher Aufwand: **O(1)**



## Permutieren nach Dijkstra: Beispiel



# Naives Sortieren: Demo



The screenshot shows a Java code editor in an IDE. The title bar says "Sort - Sort.java". The project navigation bar shows the path: 07\_Sort > src > de > uni\_bremen > pi2 > Sort > quickSort. The code editor has two tabs: "Sort.java" (selected) and "jshell\_console.snippet". The "Sort.java" tab contains the following code:

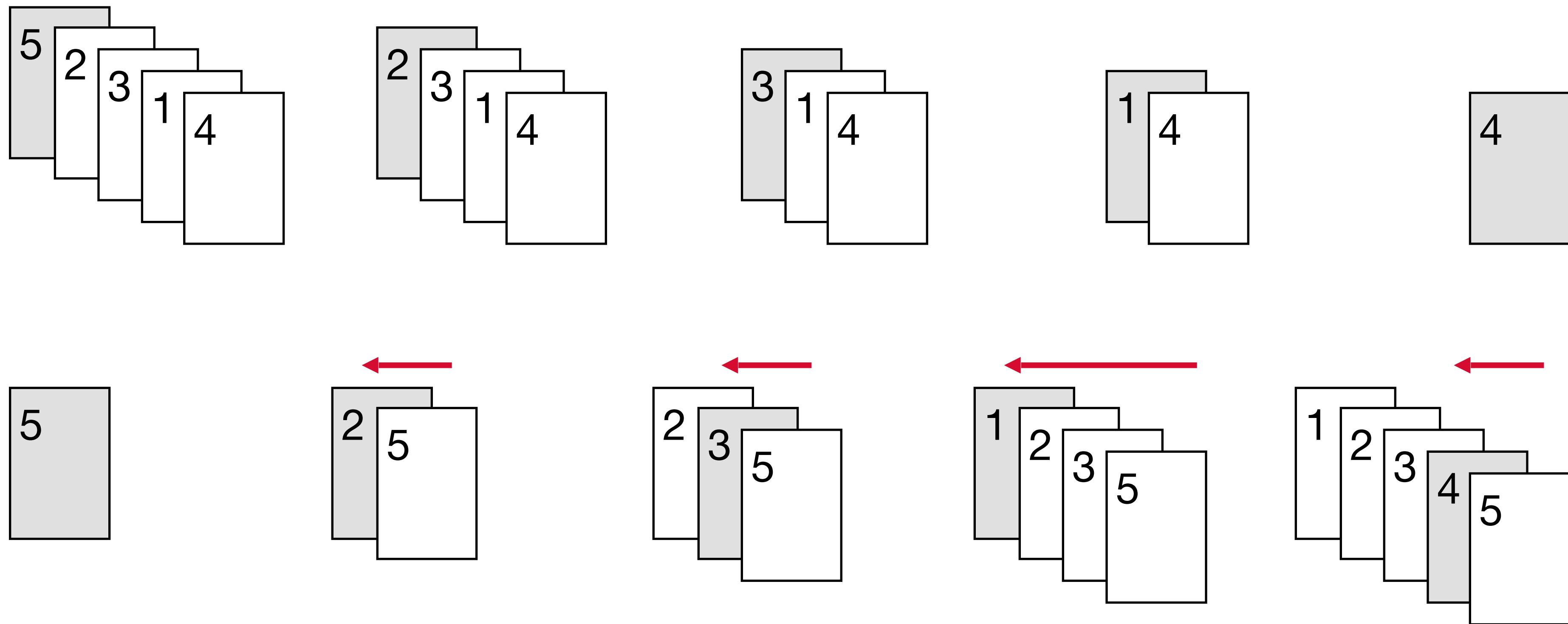
```
11  /**
12  * Tauscht zwei Elemente eines Arrays aus.
13  * @param a Das Array, in dem die beiden Elemente getauscht werden.
14  * @param i Der Index des einen Elements.
15  * @param j Der Index des anderen Elements.
16  */
17  private static void swap(final Object[] a, final int i, final int j)
18  {
19      final Object temp = a[i];
20      a[i] = a[j];
21      a[j] = temp;
22  }
23
24  /**
25  * Sortieren durch Einfügen entsprechend der natürlichen Ordnung der
26  * Elemente eines Arrays.
27  * @param a Das Array, das sortiert wird.
28  * @param <T> Der Typ der Elemente des Arrays.
29  */
```

The code editor interface includes a "Project" view on the left, a "Structure" view, and a "Favorites" view. The bottom of the editor shows tabs for "Problems", "Git", "Terminal", "TODO", "Build", and "Event Log". A status bar at the bottom indicates "225:17 LF UTF-8 4 spaces master".

## Greedy-Sortierverfahren

- Sortieren durch Einfügen
- Sortieren durch Auswählen
- Bubble Sort

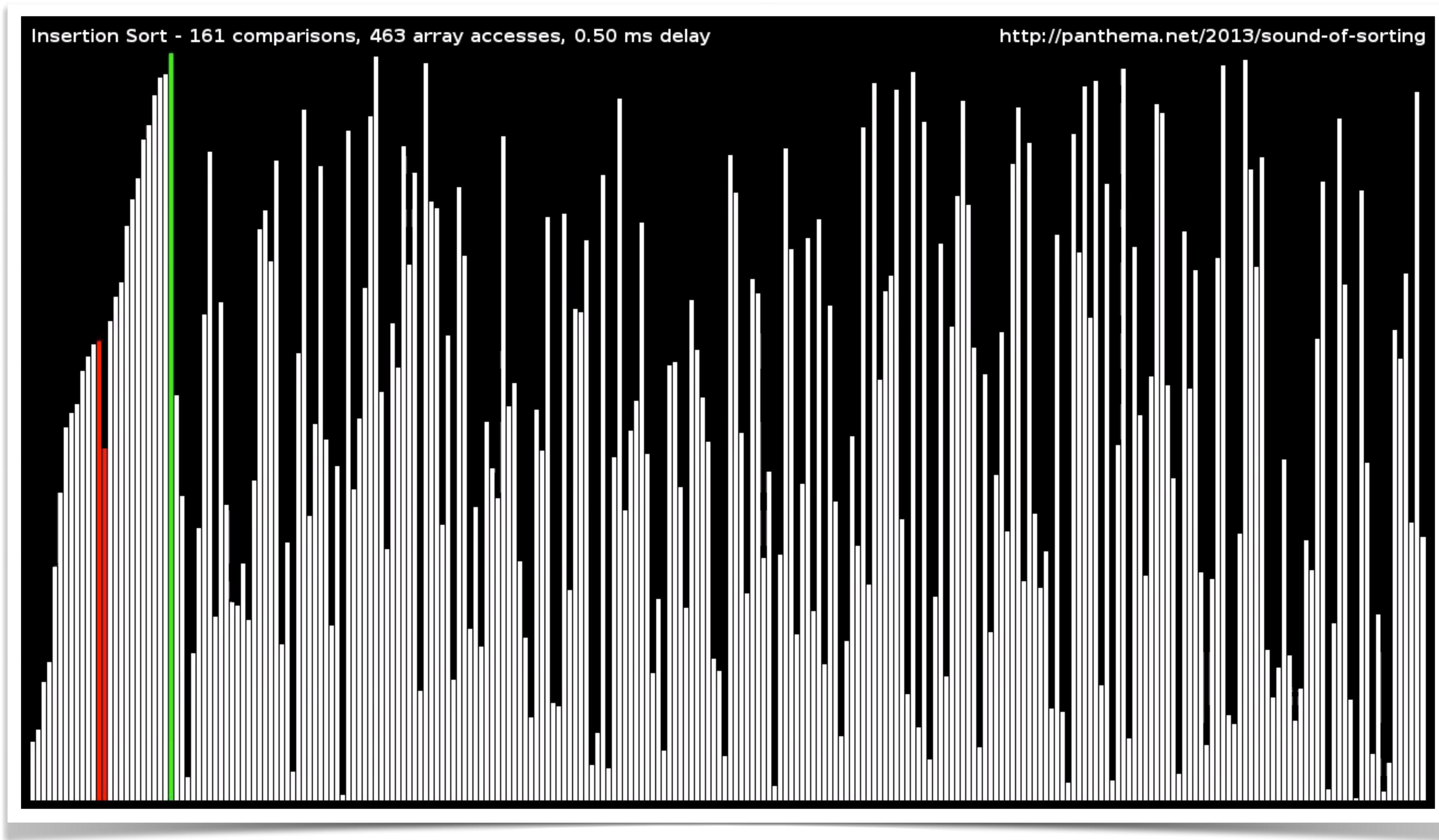
## Sortieren durch Einfügen: Beispiel



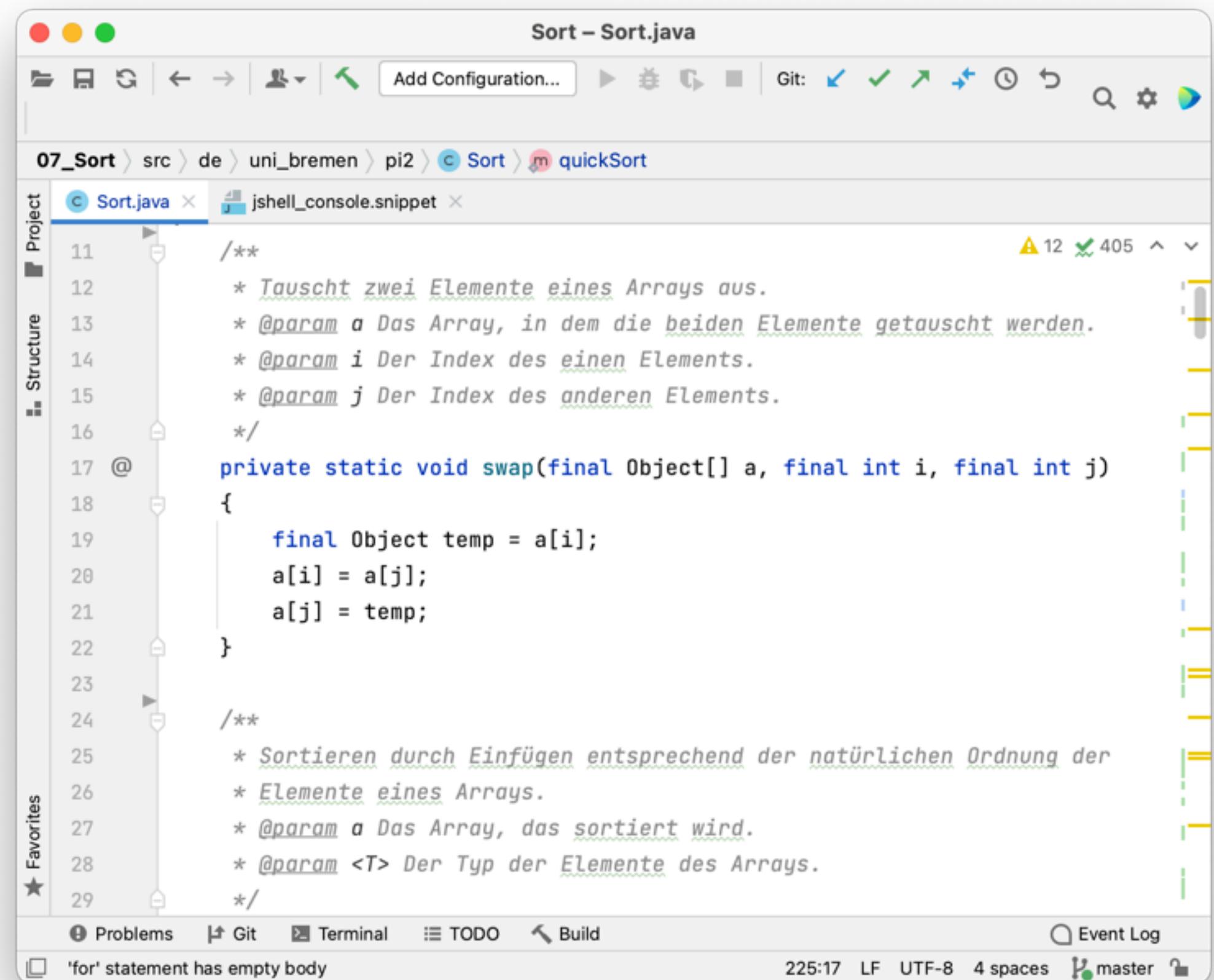
# Sortieren durch Einfügen

- Sortiertes Einfügen aller Elemente in eine ursprünglich leere Folge
- Bei Arrays Verschieben aller Elemente hinter der Einfügeposition
- Bei Arrays Nutzung des Platzes von entnommenen Elementen für Ergebnis
- Aufwand
  - Bester Fall:  **$O(N)$**  (vorsortiert)
  - Schlechtester Fall:  **$O(\frac{1}{2}N^2) = O(N^2)$**  (rückwärts sortiert, im Mittel in halbe Folge einsortieren)
  - Durchschnittlicher Fall:  **$O(\frac{1}{4}N^2) = O(N^2)$**  (Einsortieren im Mittel in Mitte von halber Folge)
- Stabil

# Sortieren durch Einfügen: Demo



# Sortieren durch Einfügen: Demo



The screenshot shows a Java code editor with the following code:

```
Sort - Sort.java
07_Sort > src > de > uni_bremen > pi2 > Sort > quickSort

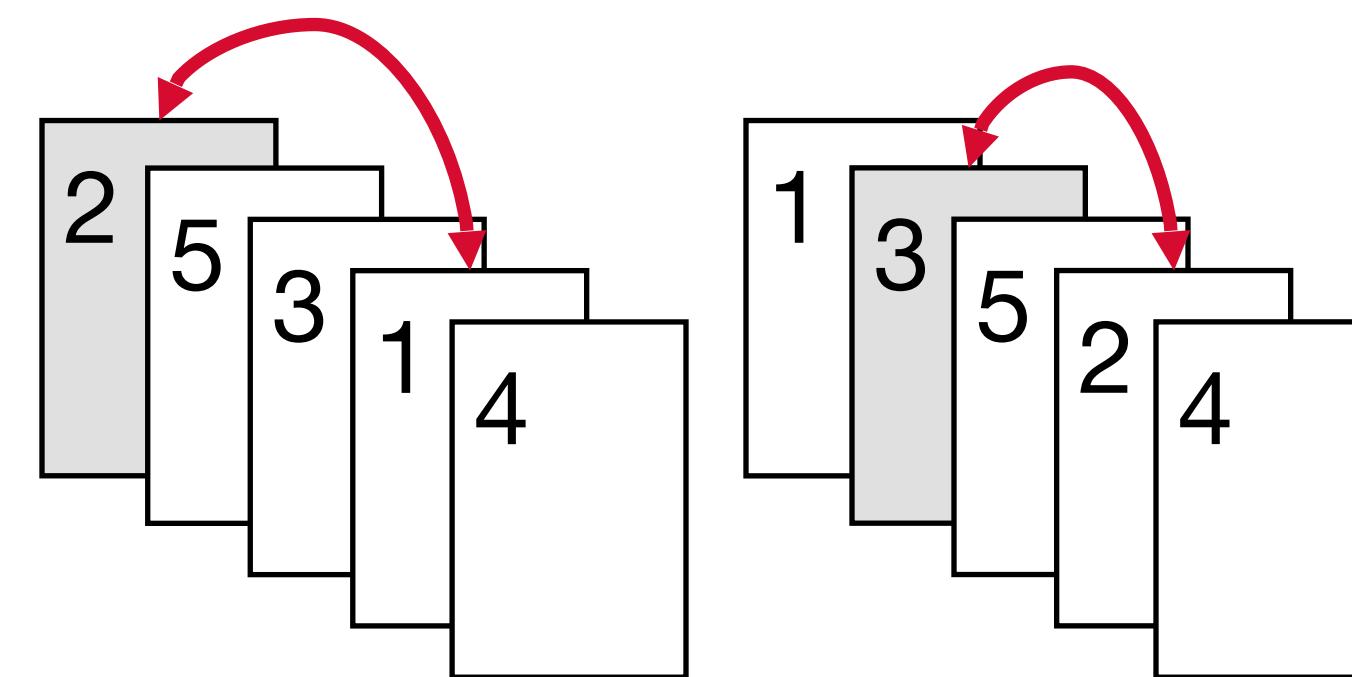
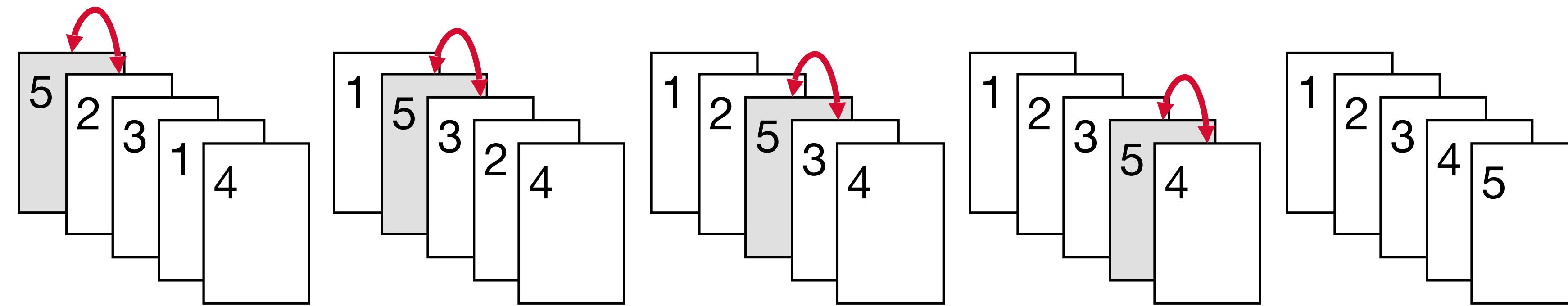
Sort.java
jshell_console.snippet

11  /**
12   * Tauscht zwei Elemente eines Arrays aus.
13   * @param a Das Array, in dem die beiden Elemente getauscht werden.
14   * @param i Der Index des einen Elements.
15   * @param j Der Index des anderen Elements.
16   */
17  @
18  private static void swap(final Object[] a, final int i, final int j)
19  {
20      final Object temp = a[i];
21      a[i] = a[j];
22      a[j] = temp;
23  }
24  /**
25   * Sortiert durch Einfügen entsprechend der natürlichen Ordnung der
26   * Elemente eines Arrays.
27   * @param a Das Array, das sortiert wird.
28   * @param <T> Der Typ der Elemente des Arrays.
29  */

Event Log
```

The code implements a swap method and a quickSort method. The quickSort method is annotated with Javadoc and uses the swap method to sort an array. The swap method swaps elements at indices i and j.

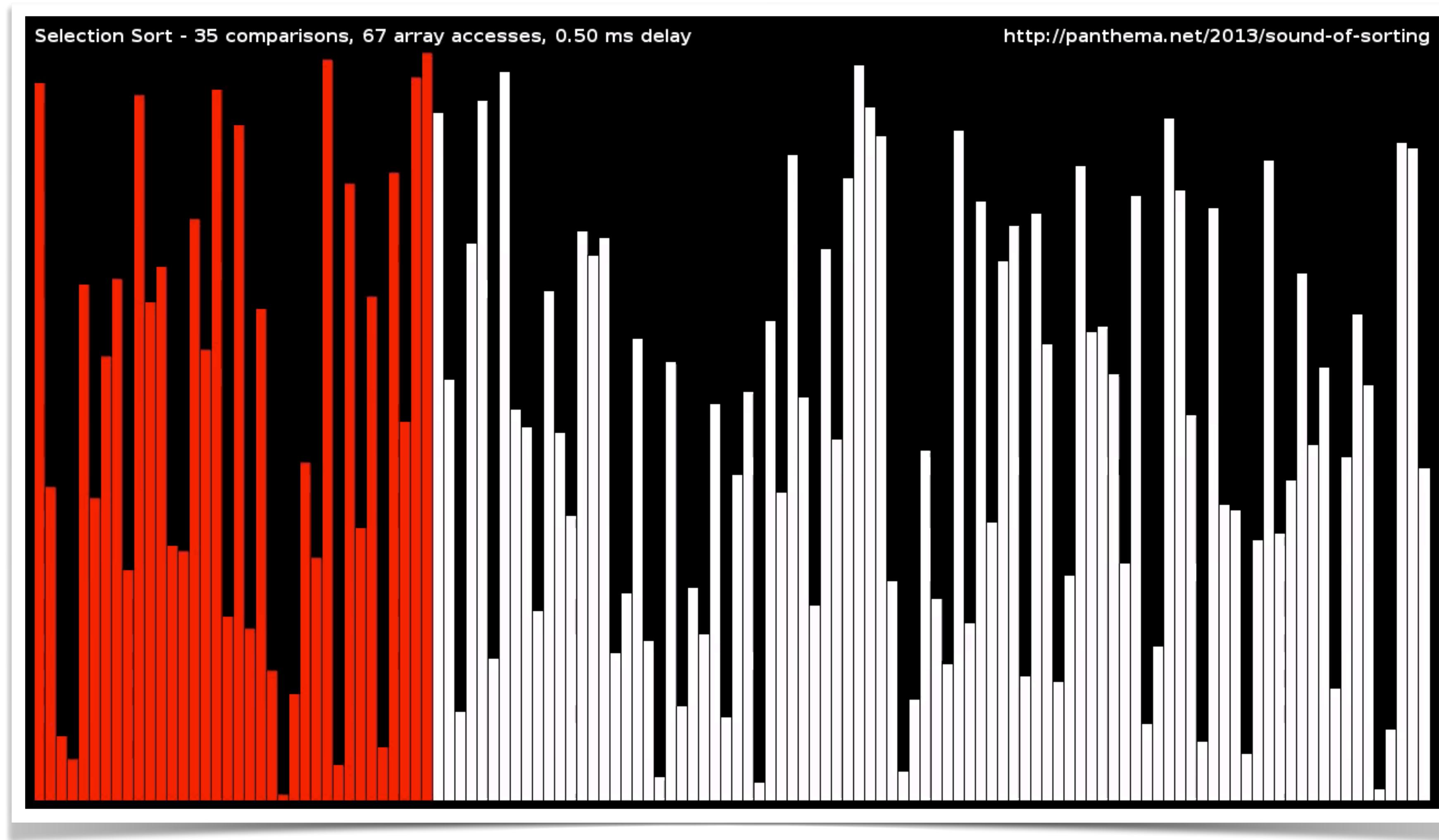
## Sortieren durch Auswählen: Beispiel



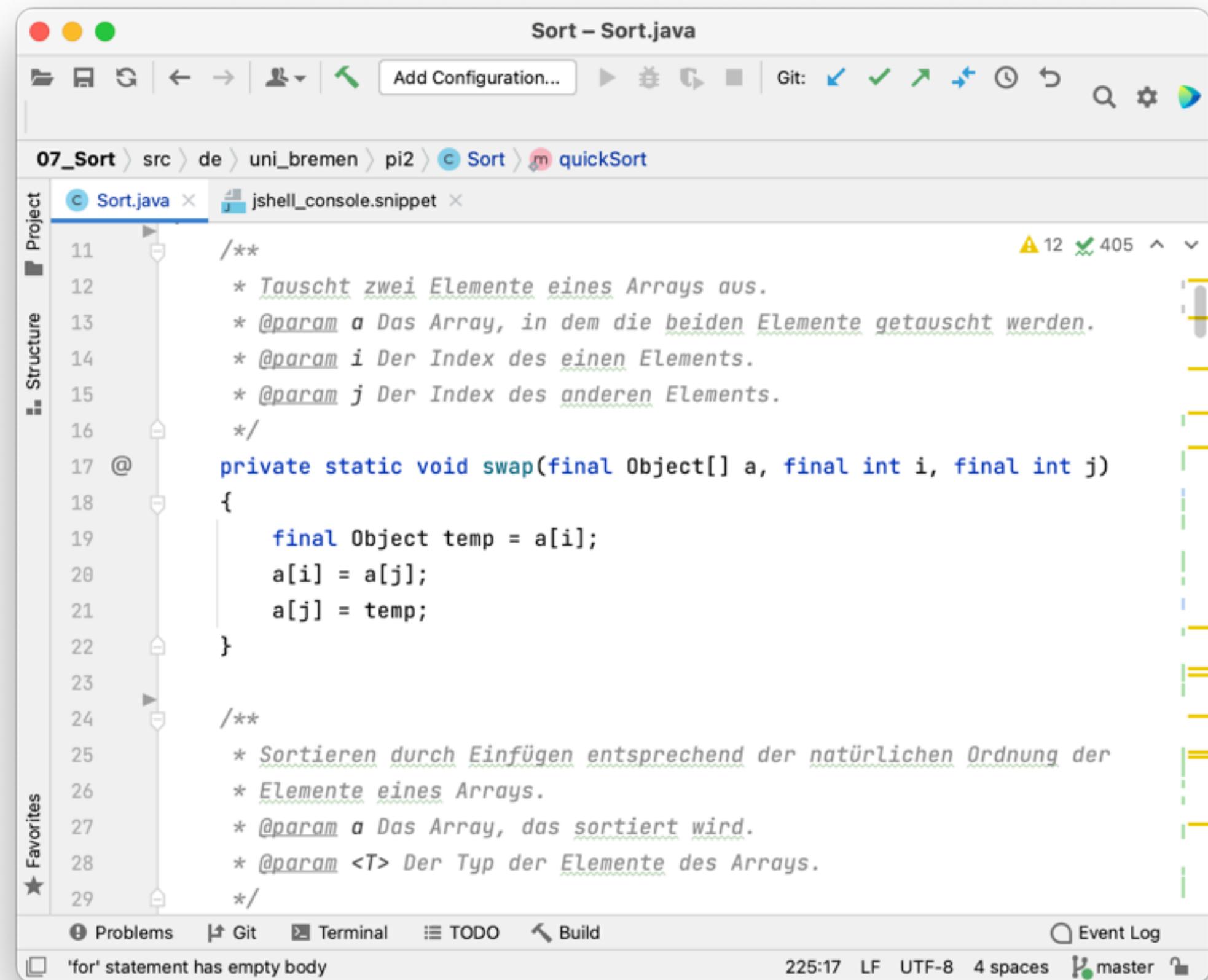
# Sortieren durch Auswählen

- Algorithmus
  - Falls nur noch ein Element übrig ist, ist die Folge sortiert
  - Sonst suche nach kleinstem Element und stelle es an den Anfang
  - Sortiere den Rest
- Suchen nach dem kleinsten Element wird oft durch sukzessives Vertauschen kleinerer Elemente mit dem Anfangselement implementiert
- Aufwand:  **$O(\frac{1}{2}N^2) = O(N^2)$**  (In jedem Durchgang wird ein Element ausgewählt und dazu müssen im Mittel  **$\frac{1}{2}N$**  Elemente durchsucht werden)
- Nicht stabil: Das jeweils ersetzte Element bekommt „zufälligen“ Platz

# Sortieren durch Auswählen: Demo



# Sortieren durch Auswählen: Demo



The screenshot shows a Java code editor with the following code:

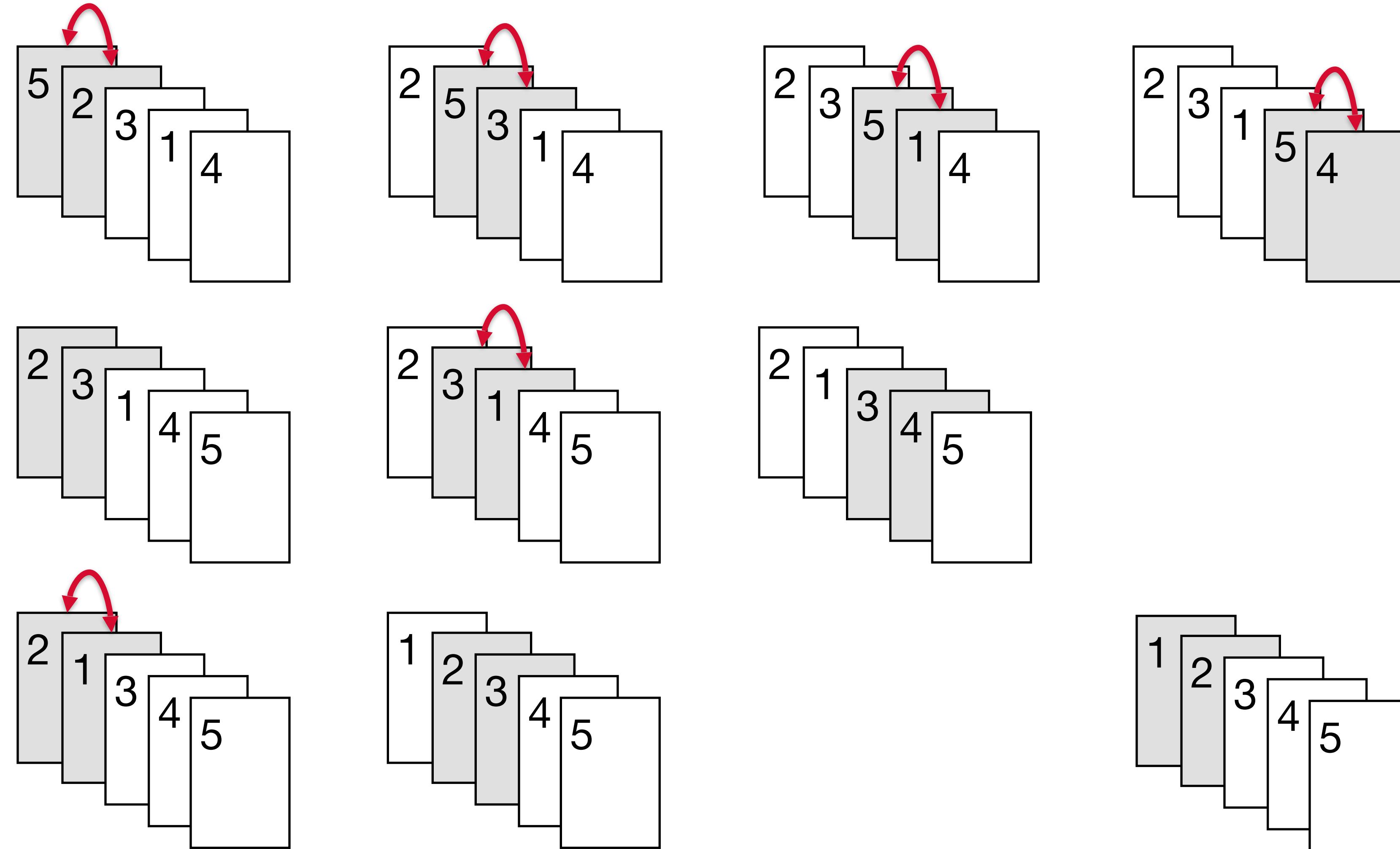
```
Sort - Sort.java
07_Sort > src > de > uni_bremen > pi2 > Sort > quickSort
Sort.java x jshell_console.snippet x

11  /**
12   * Tauscht zwei Elemente eines Arrays aus.
13   * @param a Das Array, in dem die beiden Elemente getauscht werden.
14   * @param i Der Index des einen Elements.
15   * @param j Der Index des anderen Elements.
16   */
17  @
18  private static void swap(final Object[] a, final int i, final int j)
19  {
20      final Object temp = a[i];
21      a[i] = a[j];
22      a[j] = temp;
23  }
24  /**
25   * Sortieren durch Einfügen entsprechend der natürlichen Ordnung der
26   * Elemente eines Arrays.
27   * @param a Das Array, das sortiert wird.
28   * @param <T> Der Typ der Elemente des Arrays.
29  */

Event Log
```

The code implements a swap method and a quickSort method. The swap method swaps elements at indices i and j in an array. The quickSort method is a placeholder for the quick sort algorithm.

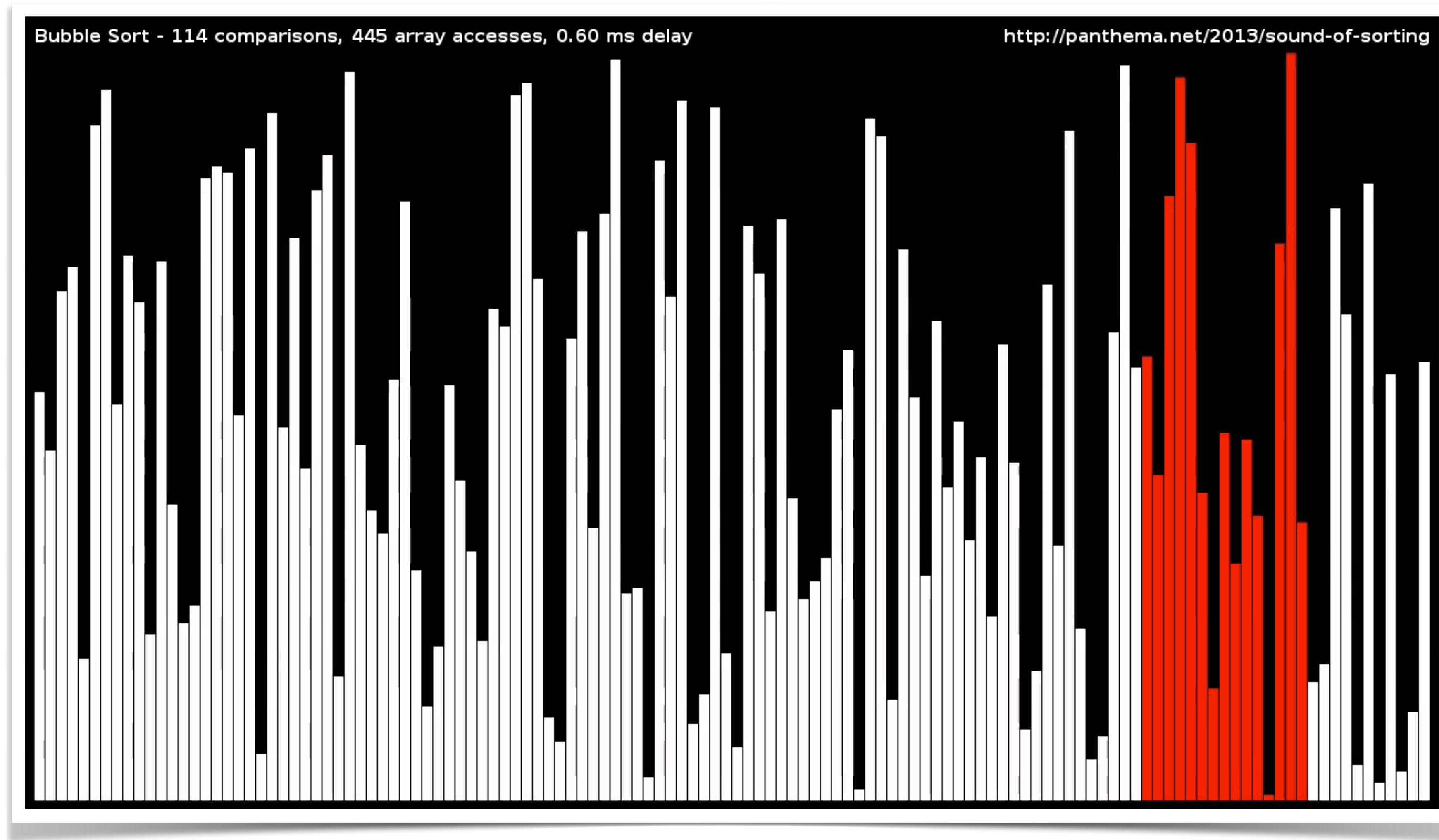
## Bubble Sort: Beispiel



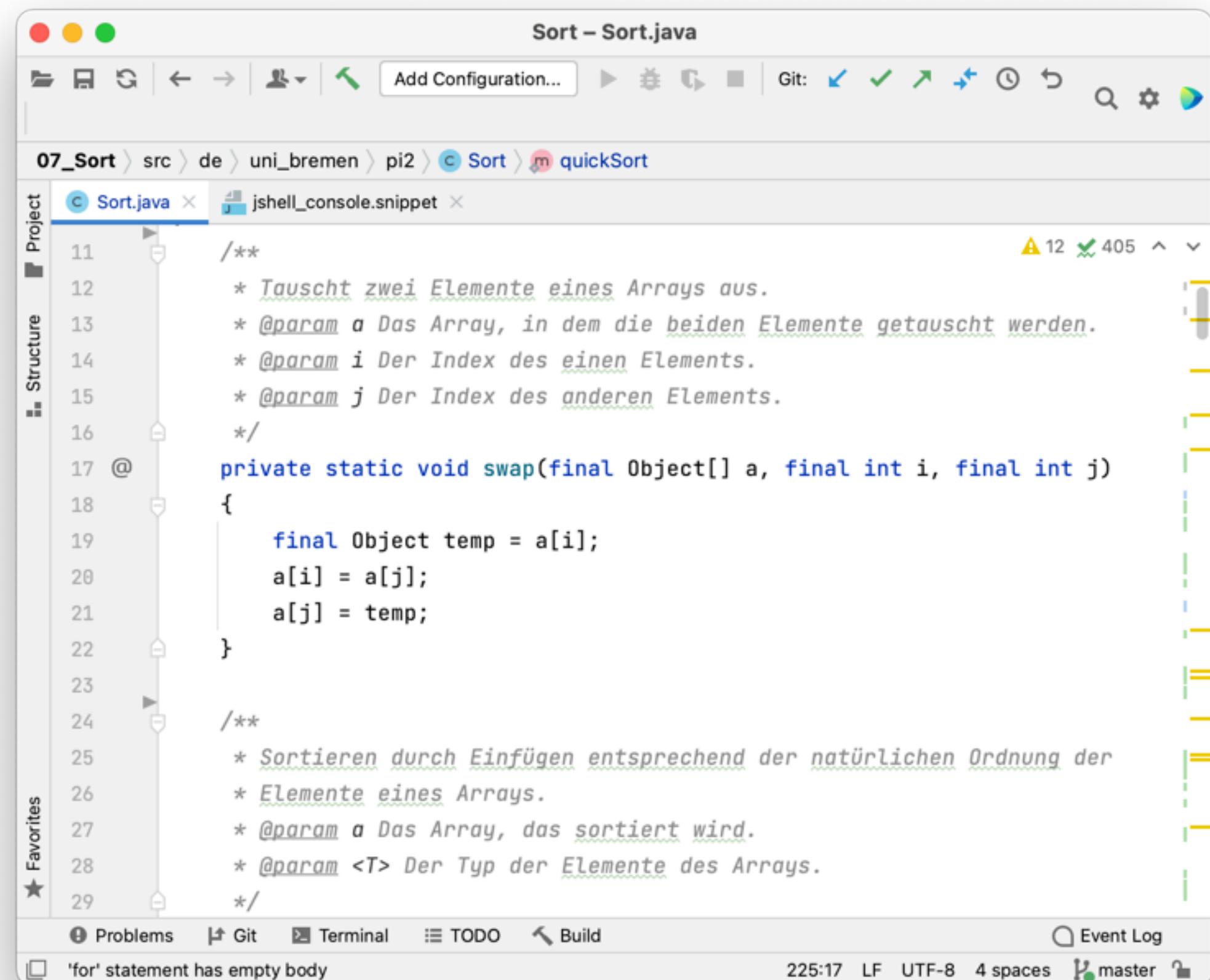
## Bubble Sort

- Algorithmus
  - Gehe der Reihe nach durch alle Elemente und vertausche zwei jeweils aufeinander folgende, wenn ihre Reihenfolge falsch ist
  - Wiederhole dies, bis keine Vertauschungen mehr notwendig sind
- Aufwand
  - Günstigster Fall:  **$O(N)$**  (vorsortiert)
  - Schlechtester Fall:  **$O(\frac{1}{2}N^2) = O(N^2)$**  (rückwärts sortiert, im Mittel halbe Folge durchlaufen)
  - Durchschnittlicher Fall:  **$O(N^2)$**  (im Detail kompliziert, wegen Wahrscheinlichkeit vorzeitigen Abbruchs)
- Stabil

## Bubble Sort: Demo



# Bubble Sort: Demo



The screenshot shows a Java code editor with the following details:

- Title Bar:** Sort - Sort.java
- Toolbar:** Includes icons for file operations, navigation, and Git status.
- Project Bar:** Shows the current project structure: 07\_Sort > src > de > uni\_bremen > pi2 > Sort > quickSort.
- Code Editor:** The file Sort.java is open, showing the implementation of the swap method and the main logic of the bubble sort algorithm.
- Toolbars:** Structure and Favorites.
- Status Bar:** Shows a warning: 'for' statement has empty body, and other build-related information.

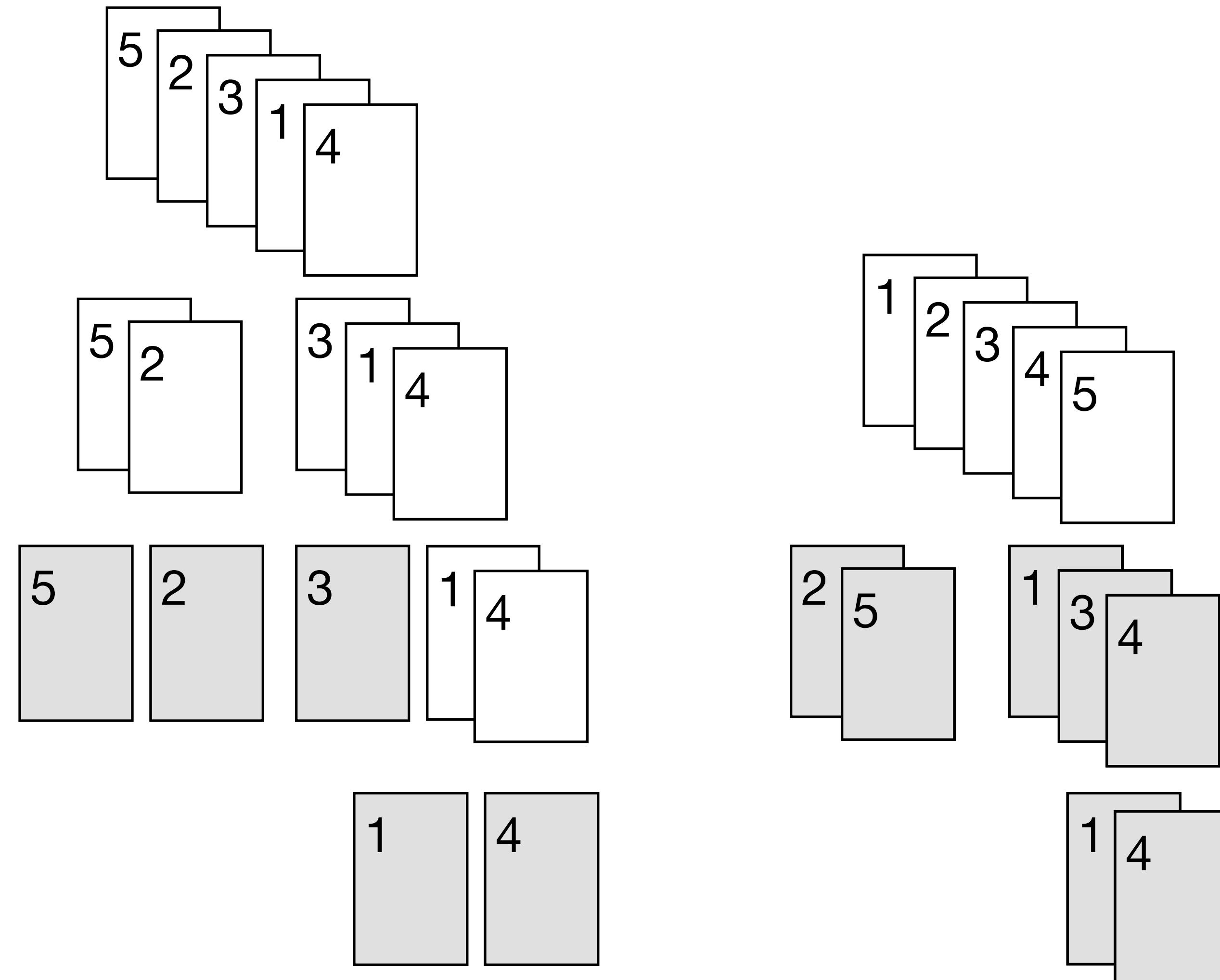
```
Sort - Sort.java
07_Sort > src > de > uni_bremen > pi2 > Sort > quickSort
Sort.java x jshell_console.snippet x
11  /**
12   * Tauscht zwei Elemente eines Arrays aus.
13   * @param a Das Array, in dem die beiden Elemente getauscht werden.
14   * @param i Der Index des einen Elements.
15   * @param j Der Index des anderen Elements.
16   */
17  @
18  private static void swap(final Object[] a, final int i, final int j)
19  {
20      final Object temp = a[i];
21      a[i] = a[j];
22      a[j] = temp;
23  }
24  /**
25   * Sortieren durch Einfügen entsprechend der natürlichen Ordnung der
26   * Elemente eines Arrays.
27   * @param a Das Array, das sortiert wird.
28   * @param <T> Der Typ der Elemente des Arrays.
29  */

```

## Sortieren mit Divide and Conquer

- Sortieren durch Mischen (Merge Sort)
- Quicksort
- Radixsort
- Heapsort
- Siehe auch: **<http://tinyurl.com/jfaycvb>** (einige der Videos in der Liste)

## Sortieren durch Mischen: Beispiel

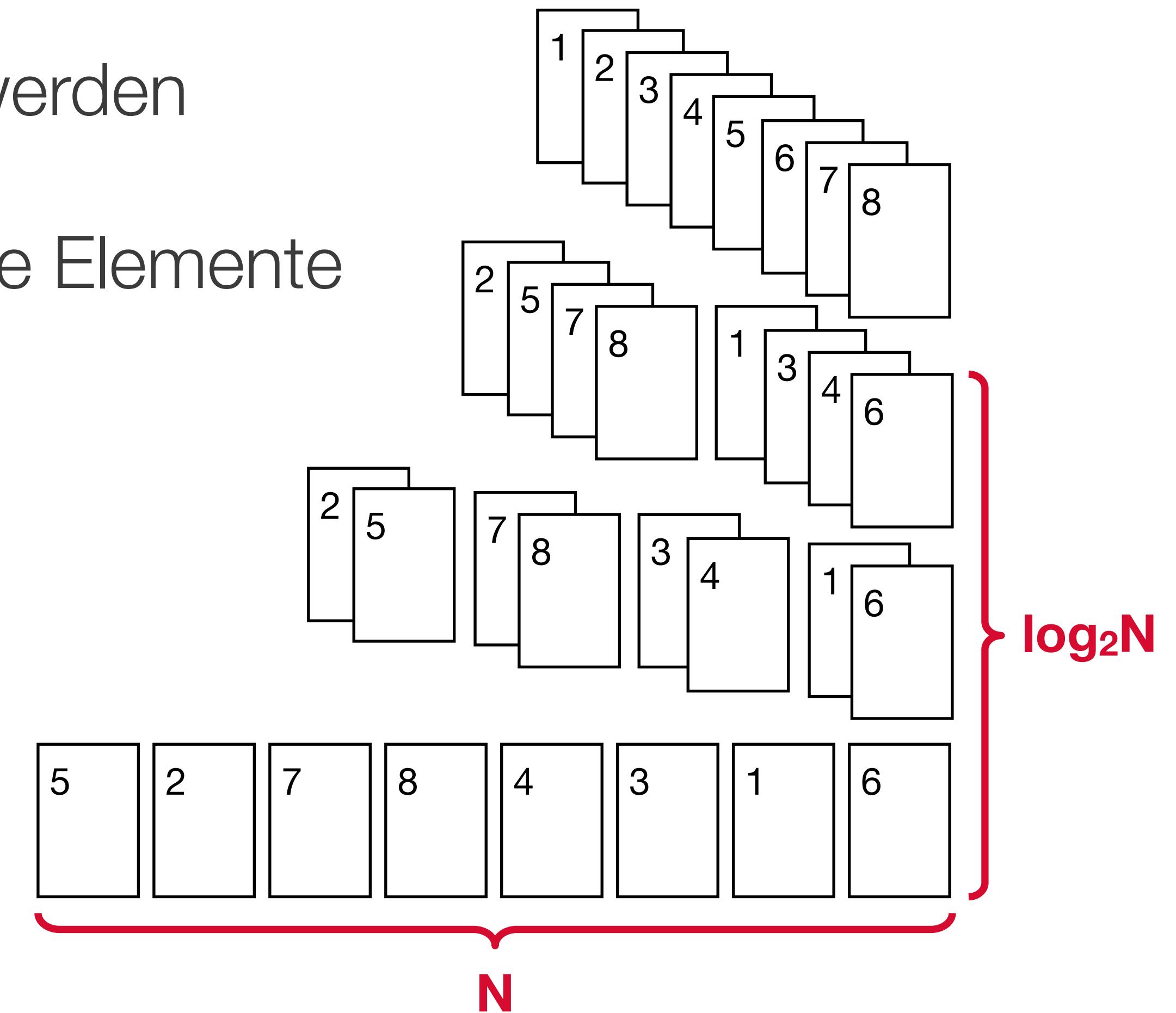


## Sortieren durch Mischen

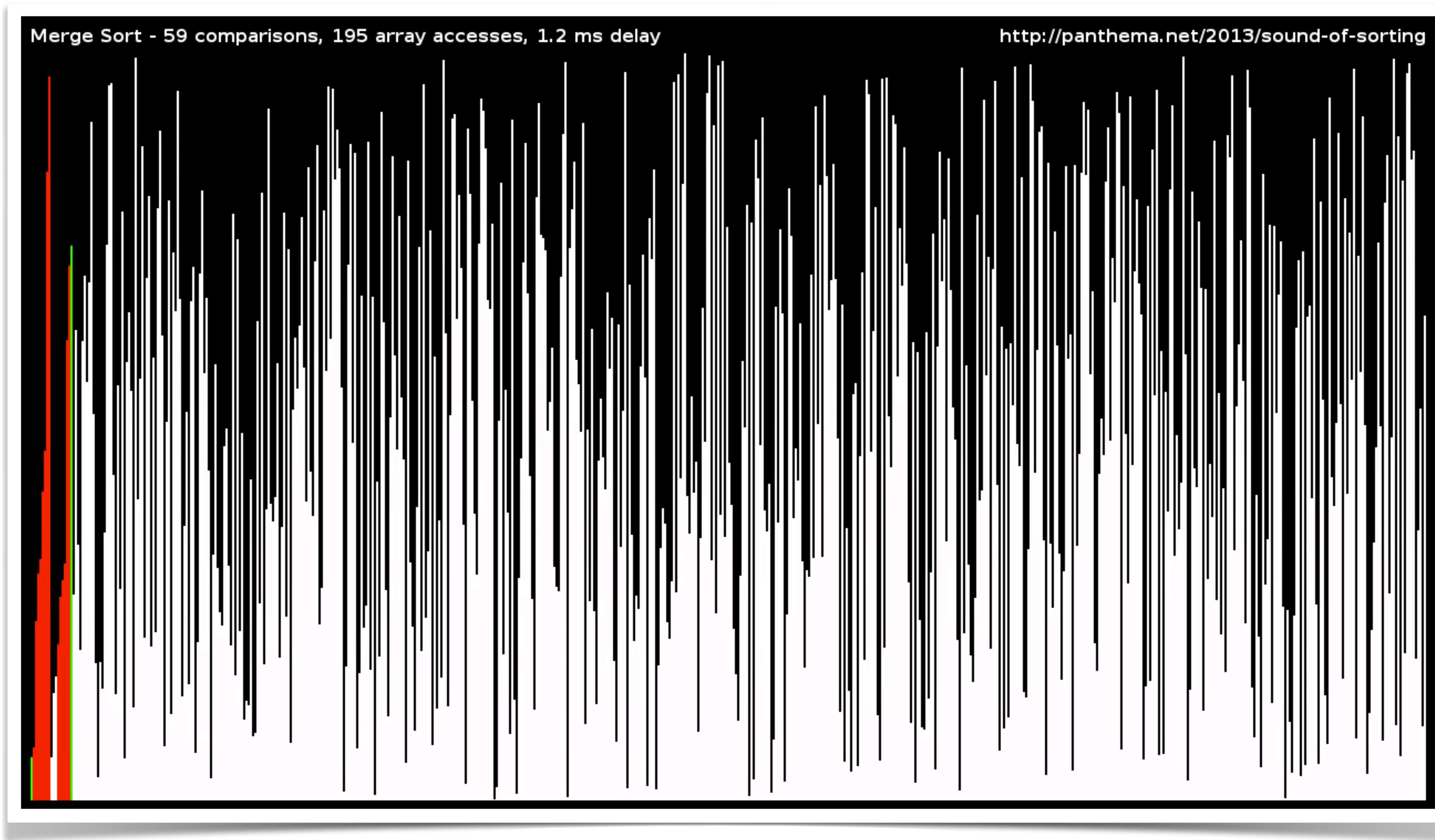
- Enthält die Folge kein oder nur ein Element, ist sie sortiert
- Ansonsten teile die Folge in zwei gleich große Hälften
- Sortiere die Hälften
- Mische die sortierten Hälften wieder zusammen
  - Vergleiche die beiden ersten Elemente der Hälften
  - Entnimm das kleinere von beiden und füge es dem Ergebnis hinzu
  - Wiederhole so lange, bis beide Hälften leer sind

## Sortieren durch Mischen: Aufwand

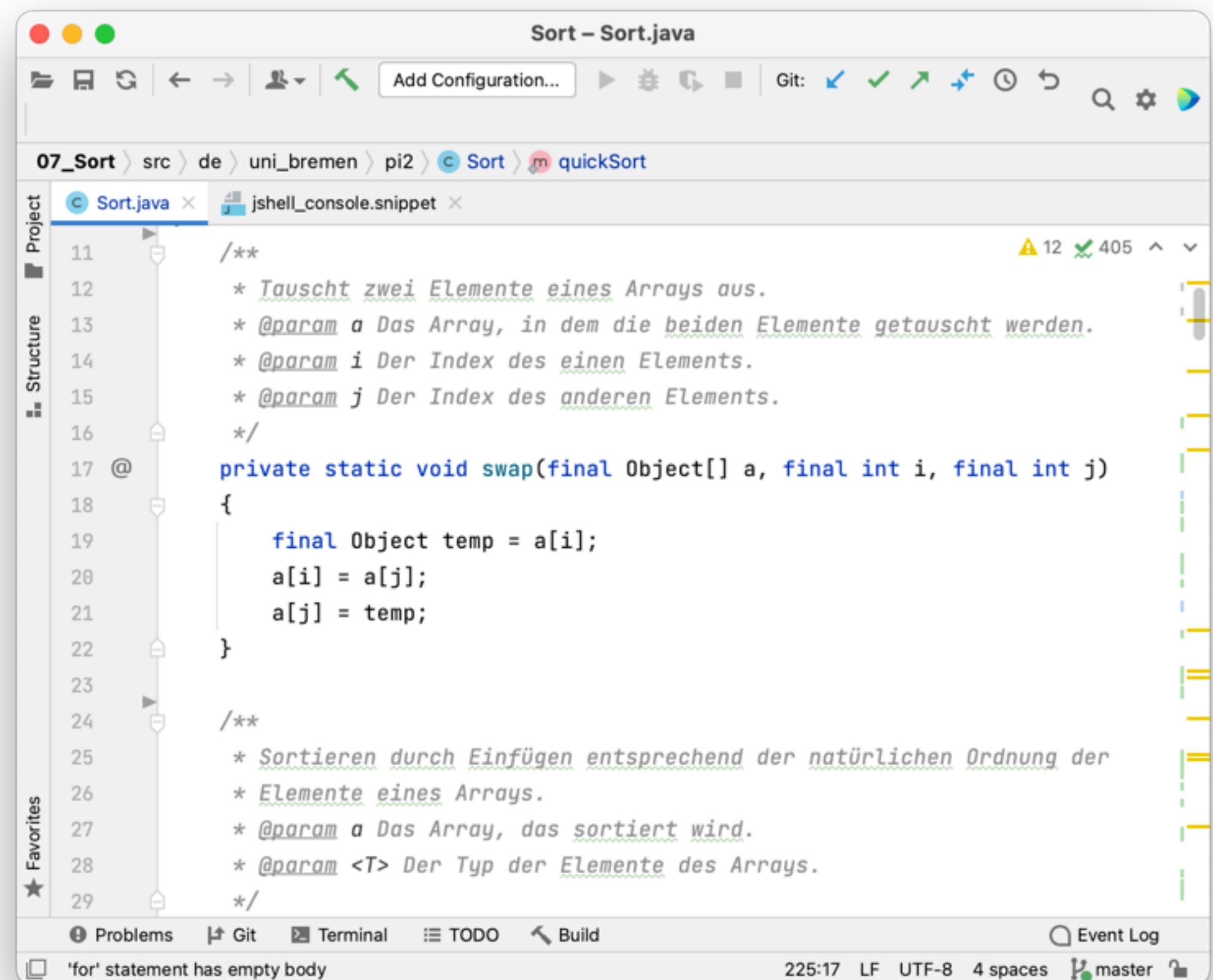
- Folge kann  $\log_2 N$ -mal in Hälften zerlegt werden
- Auf jeder Ebene der Zerlegung werden alle Elemente einmal gemischt:  $O(N)$
- Insgesamt:  $O(N \log N)$
- Platzkomplexität:  $2N$  Elemente beim Mischen von Arrays



# Merge Sort: Demo



# Merge Sort: Demo



The screenshot shows a Java code editor window with the following details:

- Title Bar:** Sort - Sort.java
- Toolbar:** Includes standard file operations (New, Open, Save, Print, Find, Replace, Copy, Paste, Cut, Undo, Redo, Find in Path, Git, Help), a search bar, and a settings gear icon.
- Project Bar:** Shows the current project structure: 07\_Sort > src > de > uni\_bremen > pi2 > Sort > quickSort.
- Code Editor:** The main pane displays the `Sort.java` file. The code is as follows:11 /\*\*
12 \* Tauscht zwei Elemente eines Arrays aus.
13 \* @param a Das Array, in dem die beiden Elemente getauscht werden.
14 \* @param i Der Index des einen Elements.
15 \* @param j Der Index des anderen Elements.
16 \*/
17 private static void swap(final Object[] a, final int i, final int j)
18 {
19 final Object temp = a[i];
20 a[i] = a[j];
21 a[j] = temp;
22 }
23
24 /\*\*
25 \* Sortieren durch Einfügen entsprechend der natürlichen Ordnung der
26 \* Elemente eines Arrays.
27 \* @param a Das Array, das sortiert wird.
28 \* @param <T> Der Typ der Elemente des Arrays.
29 \*/
- Toolbars:** Includes Project, Structure, Favorites, Problems, Git, Terminal, TODO, Build, and Event Log.
- Status Bar:** Shows the message "'for' statement has empty body", the current time (22:51), file encoding (LF), file format (UTF-8), character spacing (4 spaces), the current branch (master), and a help icon.

# Zusammenfassung

- **Aufsteigendes** und **absteigendes Sortieren**
- **Stabiles Sortieren**
- **Greedy**-Sortierverfahren haben mittleren Aufwand von  **$O(N^2)$** 
  - **Sortieren durch Einfügen** (stabil)
  - **Sortieren durch Auswählen** (nicht stabil)
  - **Bubble Sort** (stabil)
- **Divide and Conquer**-Sortierverfahren haben mittleren Aufwand von  **$O(N \log N)$** 
  - **Merge Sort** (stabil)