Universität Bremen

Praktische Informatik 2
Sortieren

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

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_i größer oder gleich seinem Vorgänger in der Folge ist
  - $s_0 \leq s_1 \leq \ldots \leq s_{N-1}$
- **S** wird als **absteigend sortiert** bezeichnet, wenn jedes Element s_i kleiner oder gleich seinem Vorgänger in der Folge ist
  - $s_0 \geq s_1 \geq \ldots \geq s_{N-1}$

Universität Bremen

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

Universität Bremen

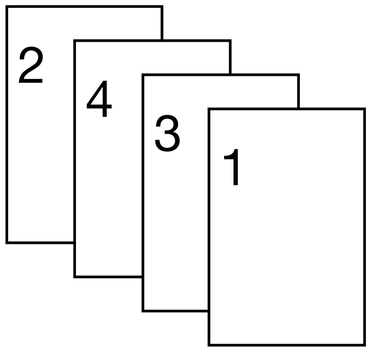
Naives Sortieren

- Erzeuge Permutationen, bis eine entsprechend des Kriteriums sortiert ist
- Bester Fall: **O(N)** (vorsortiert)
- Schlechtester Fall: **O(½N · N!) = O(N · N!)**
  - Erst die letzte Permutation ist sortiert
  - Nicht-Sortiertheit wird im Mittel nach der Hälfte der Vergleiche festgestellt
- Durchschnittlicher Fall: **O(½N · ½N!) = O(N · N!)**
  - Sortierte Folge wird nach der Hälfte der Permutationen gefunden
  - Nicht-Sortiertheit wird im Mittel nach der Hälfte der Vergleiche festgestellt

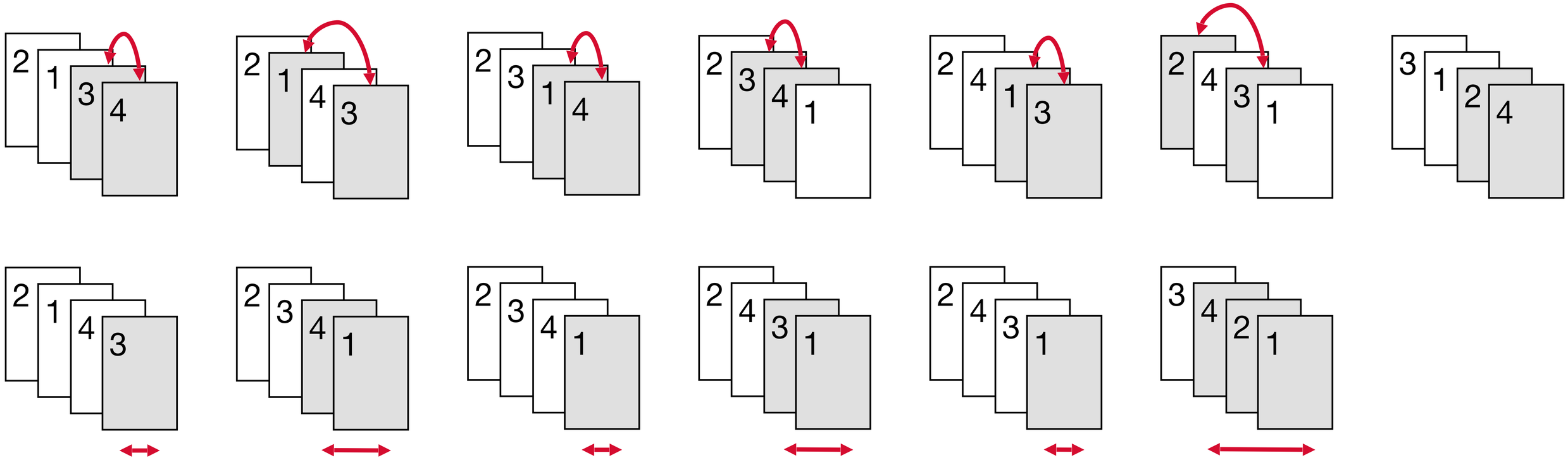
Universität Bremen

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 $a_0 \ldots a_{n-1}$
  - 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)**

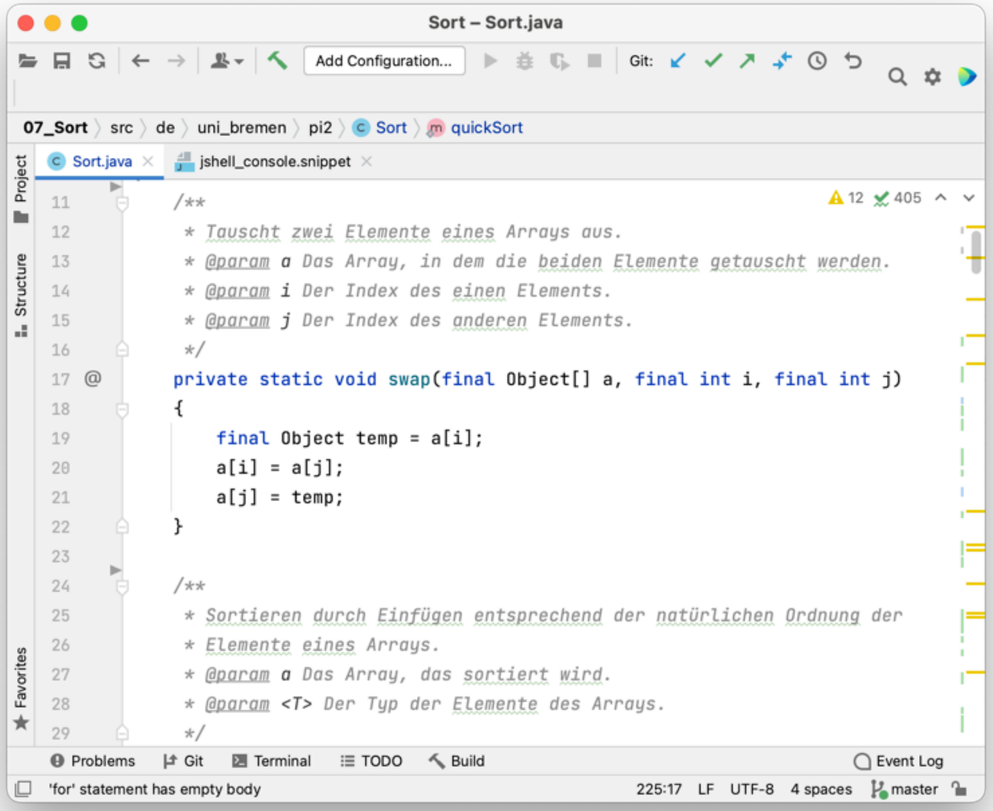


Permutieren nach Dijkstra: Beispiel



Universität Bremen

Naives Sortieren: Demo



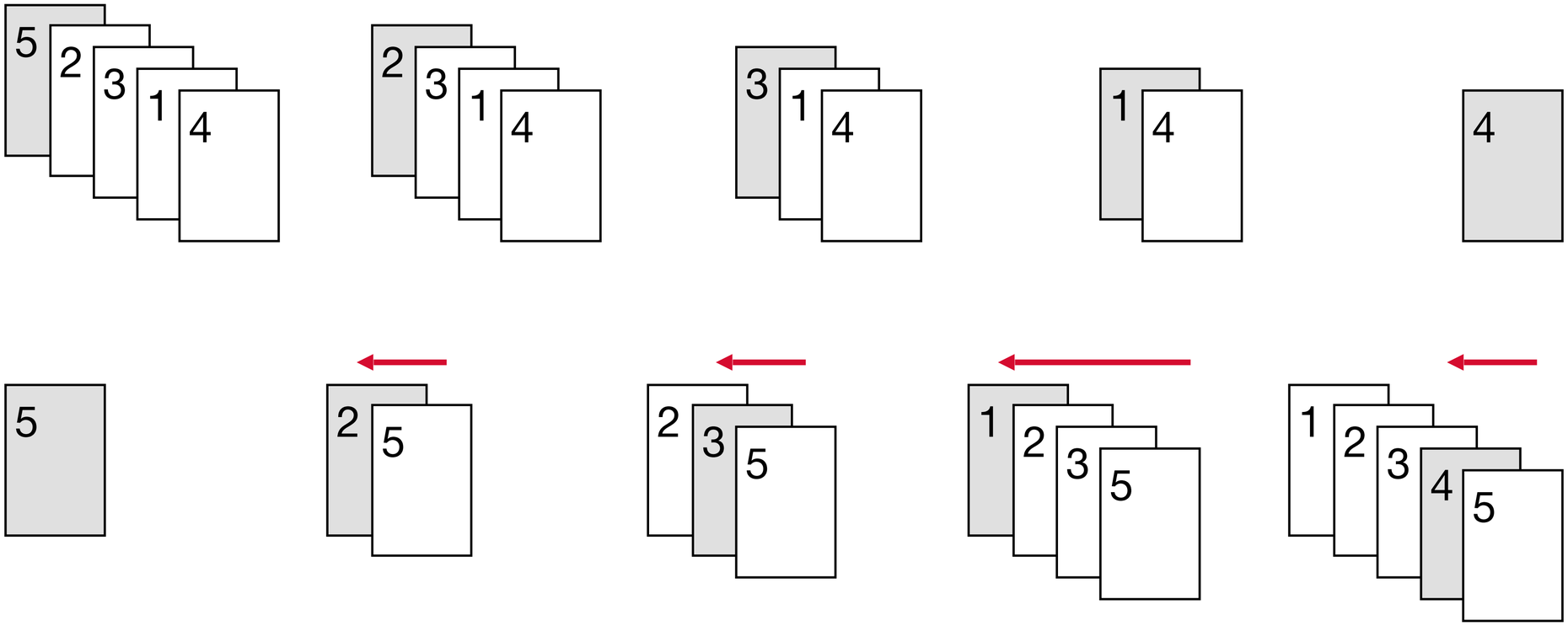
Universität Bremen

Greedy-Sortierverfahren

- Sortieren durch Einfügen
- Sortieren durch Auswählen
- Bubble Sort

Universität Bremen

Sortieren durch Einfügen: Beispiel

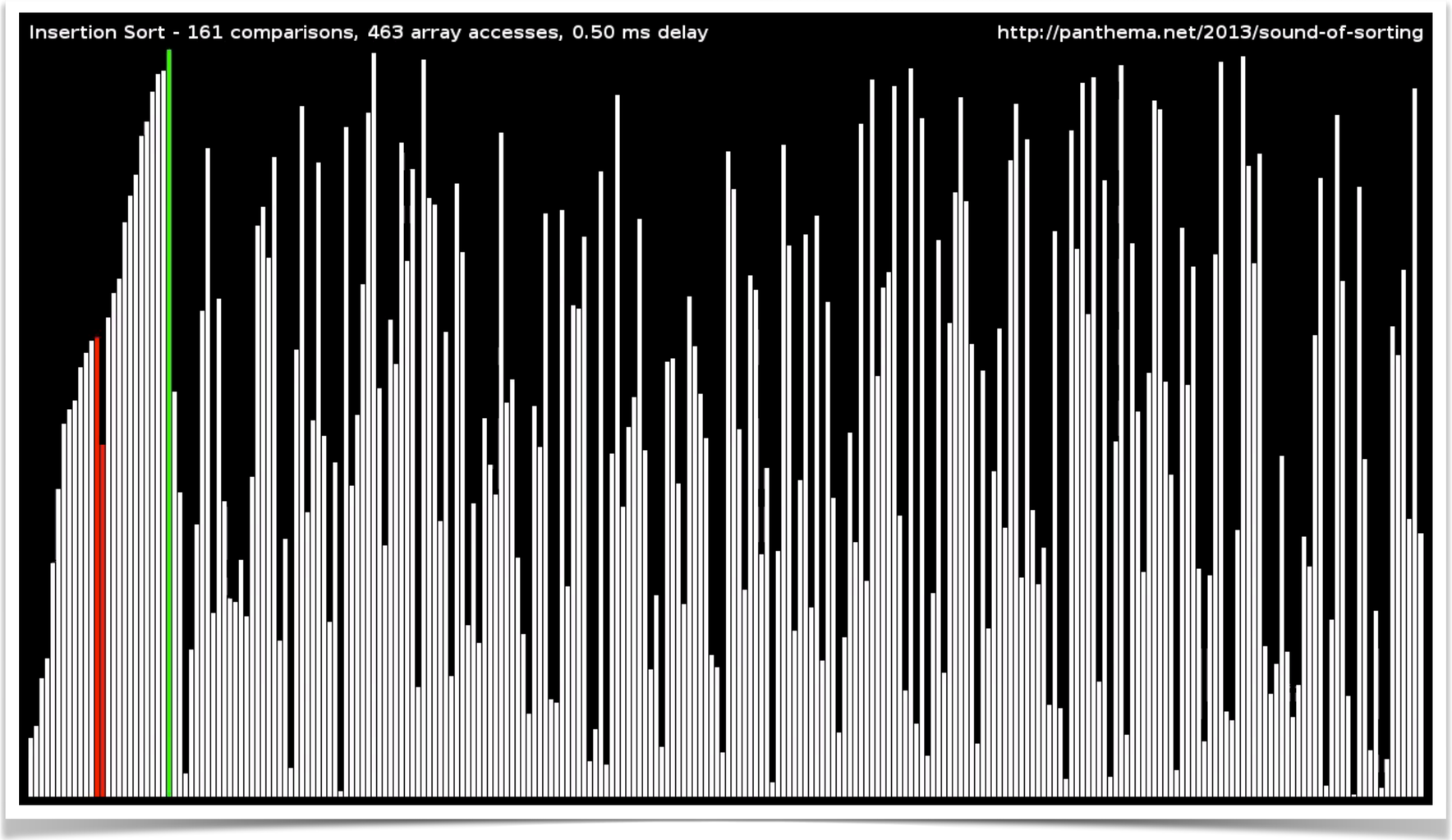


Universität Bremen

Sortieren durch Einfügen

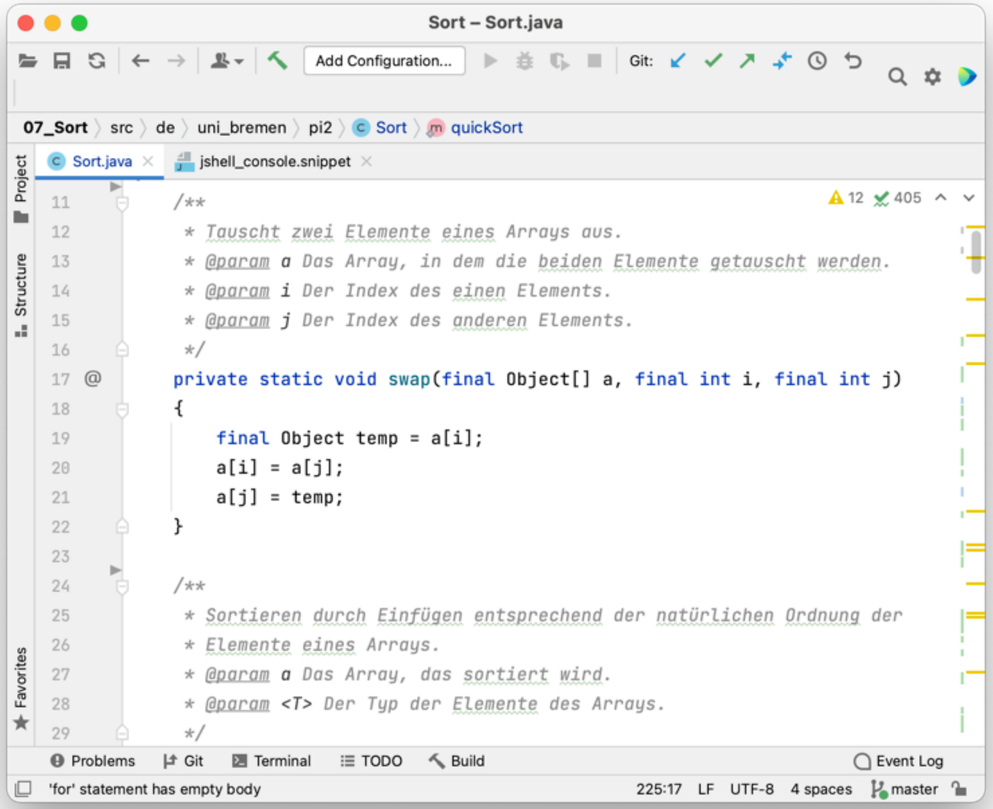
- Sortiertes Einfügen aller Elemente in eine ursprünglich leere Folge
- Bei Arrays Verschieben aller Elemente hinter der Einfügeposition
- Bei Arrays Nutzung des Platzes von entnommenen Elementen für Ergebnis
- Aufwand
  - Bester Fall: **$O(N)$** (vorsortiert)
  - Schlechtester Fall: **$O(\frac{1}{2}N^2) = O(N^2)$** (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



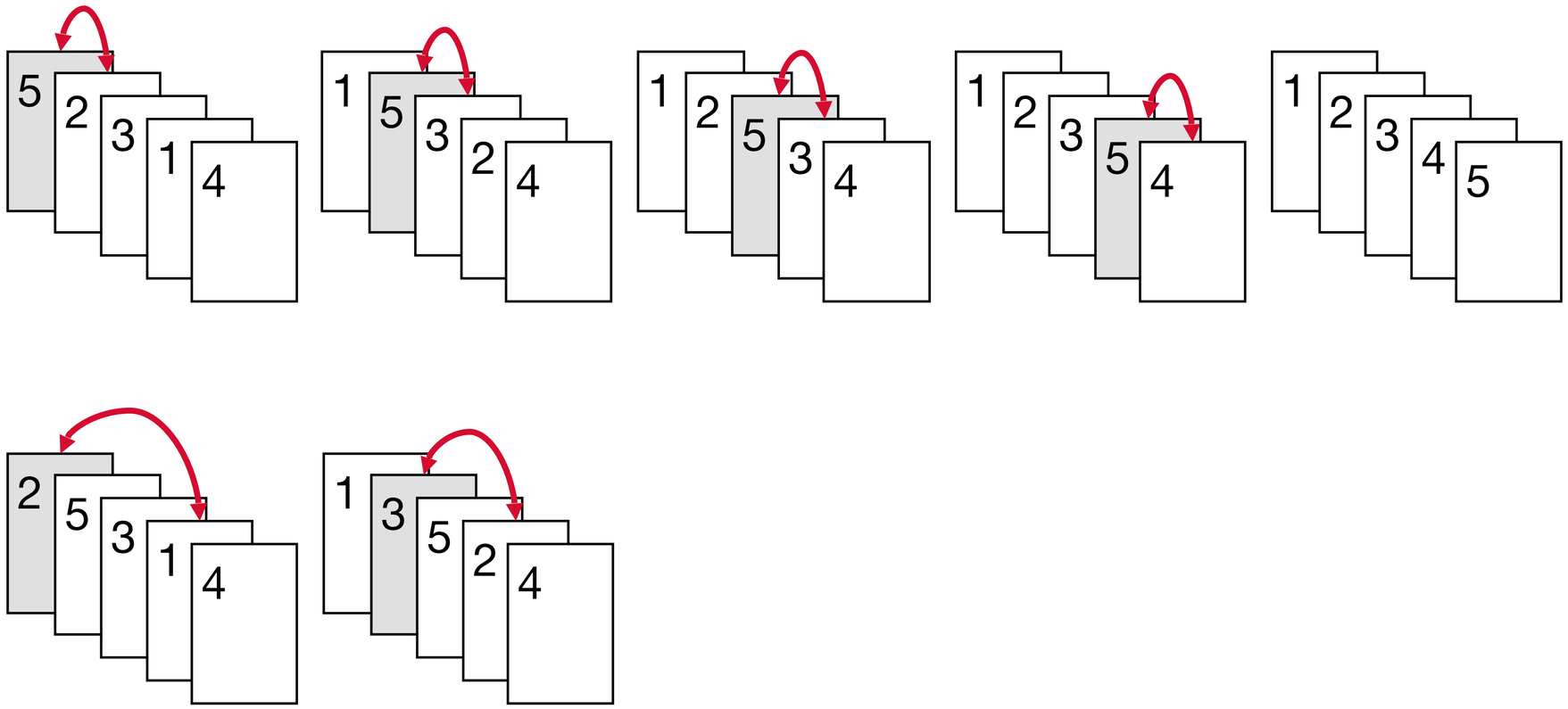
Universität Bremen

Sortieren durch Einfügen: Demo



Universität Bremen

Sortieren durch Auswählen: Beispiel



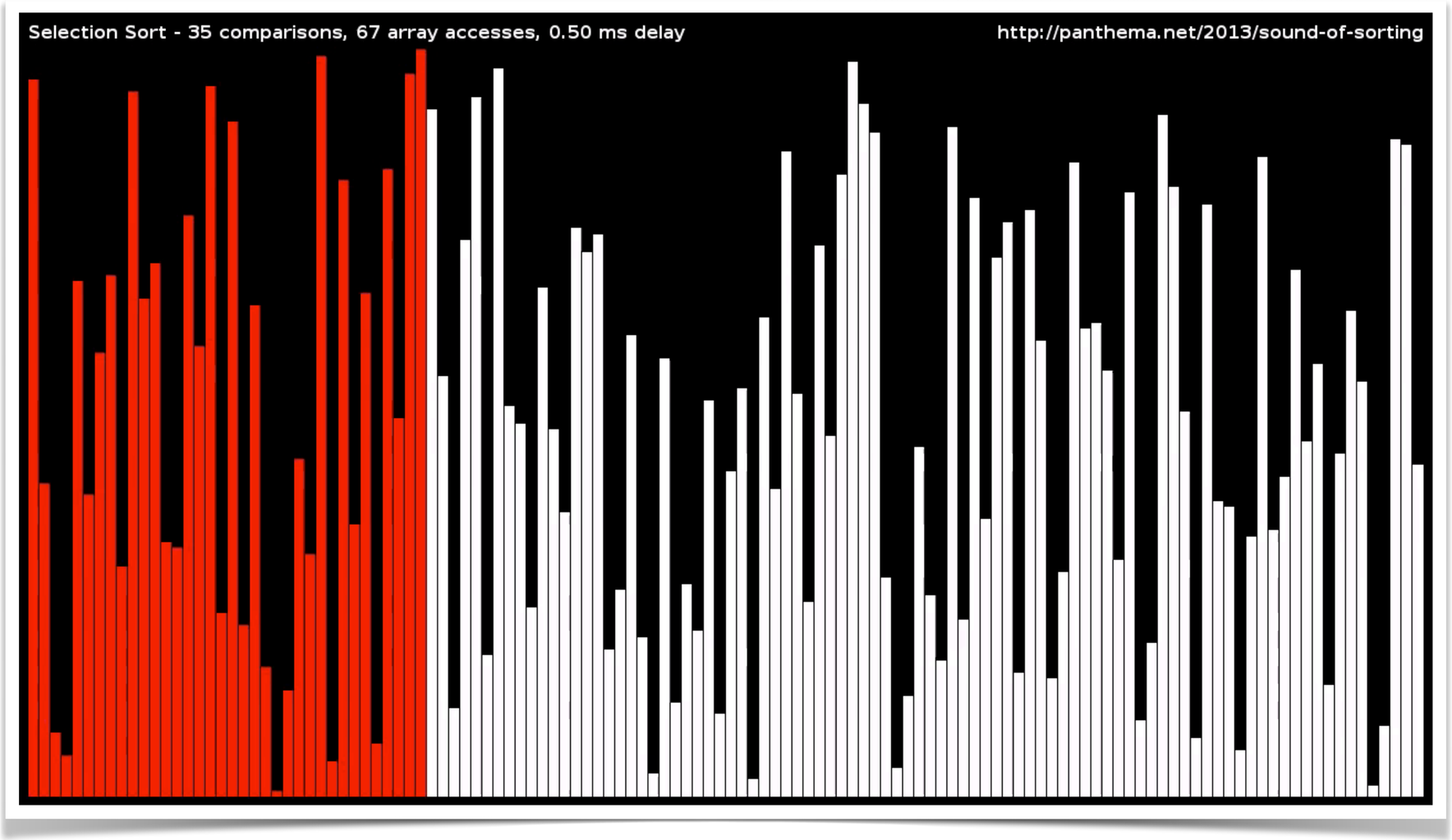
Universität Bremen

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

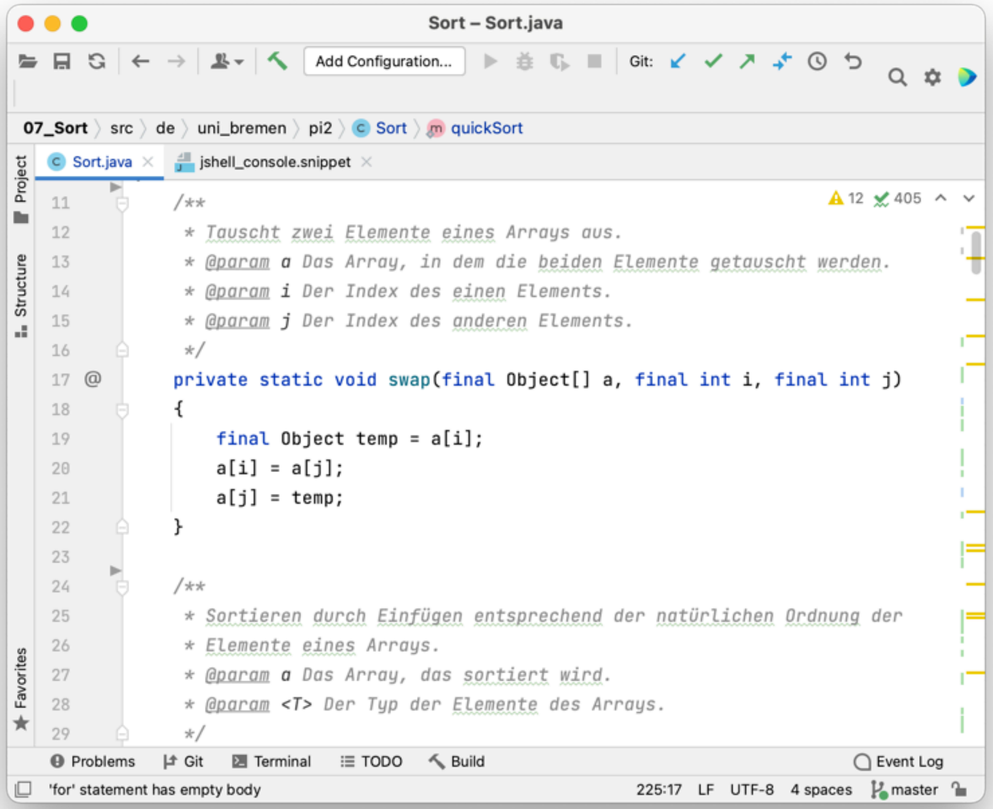
Universität Bremen

Sortieren durch Auswählen: Demo

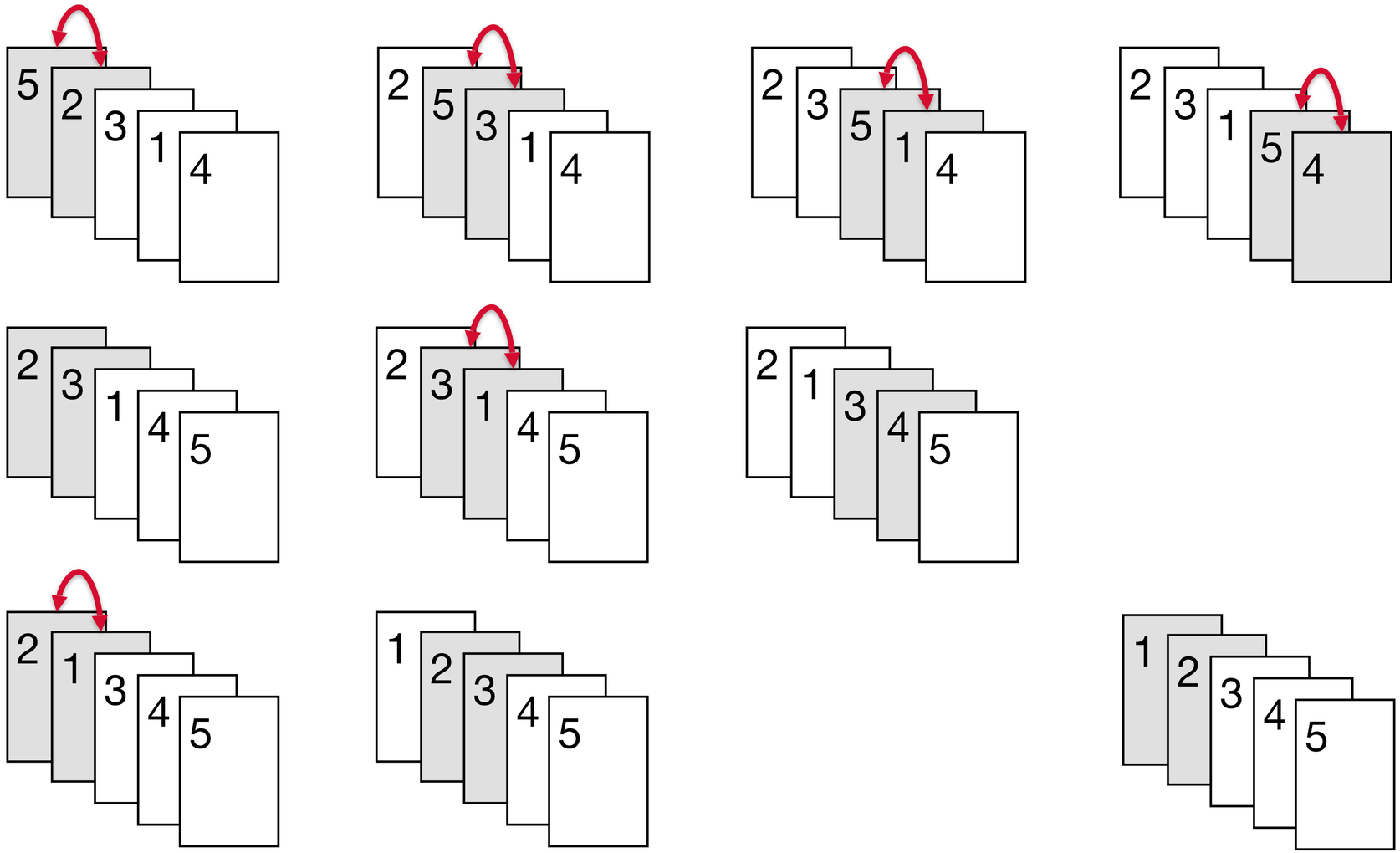


Universität Bremen

Sortieren durch Auswählen: Demo



Bubble Sort: Beispiel



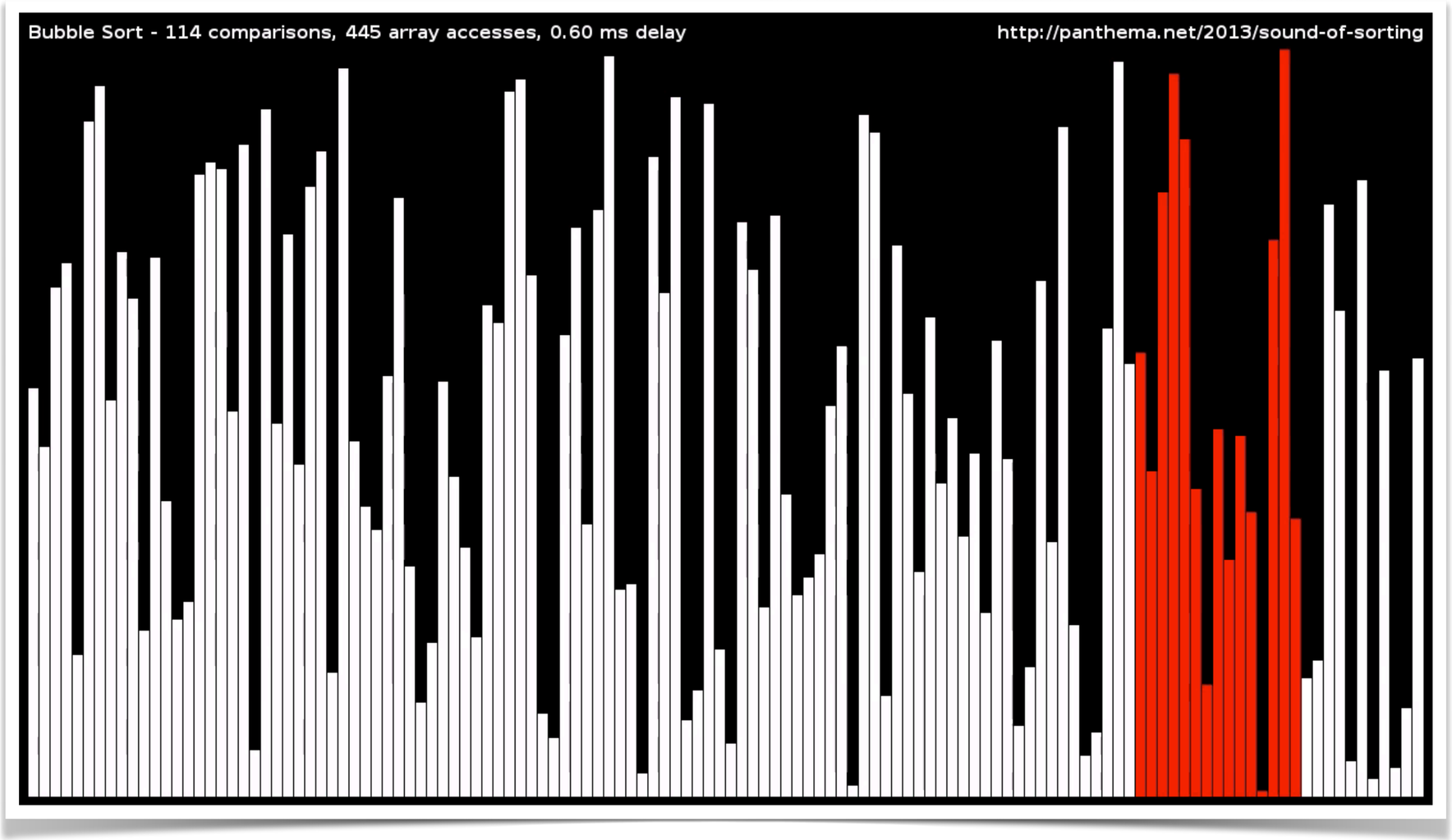
Universität Bremen

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

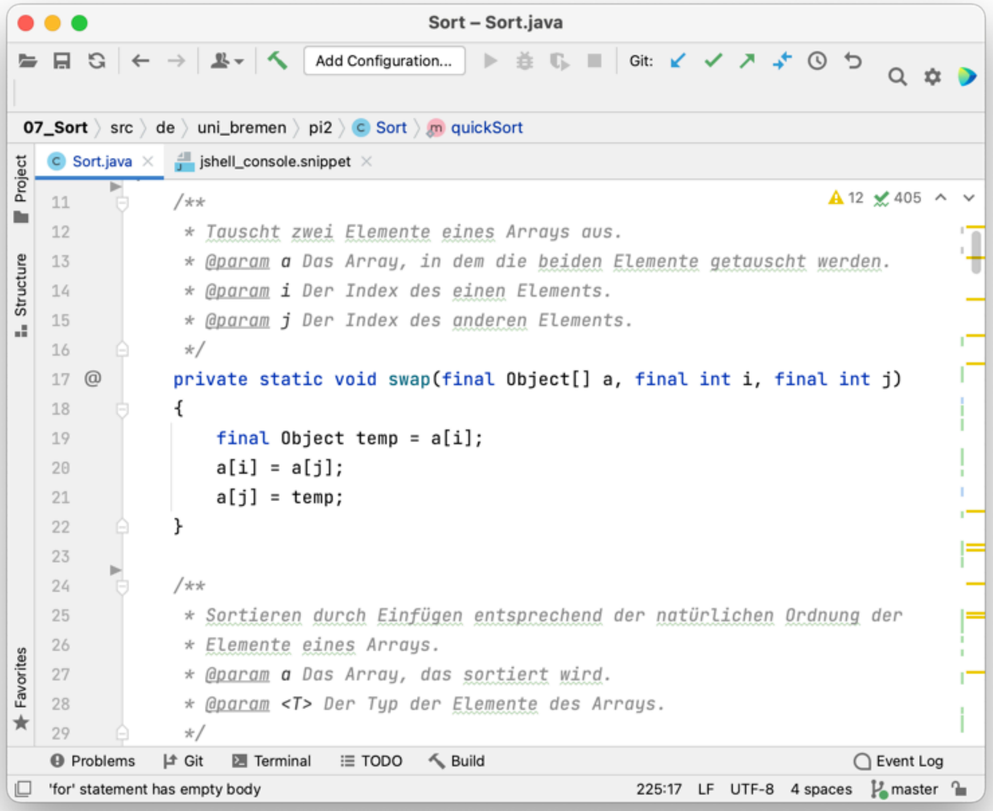
Universität Bremen

Bubble Sort: Demo



Universität Bremen

Bubble Sort: Demo

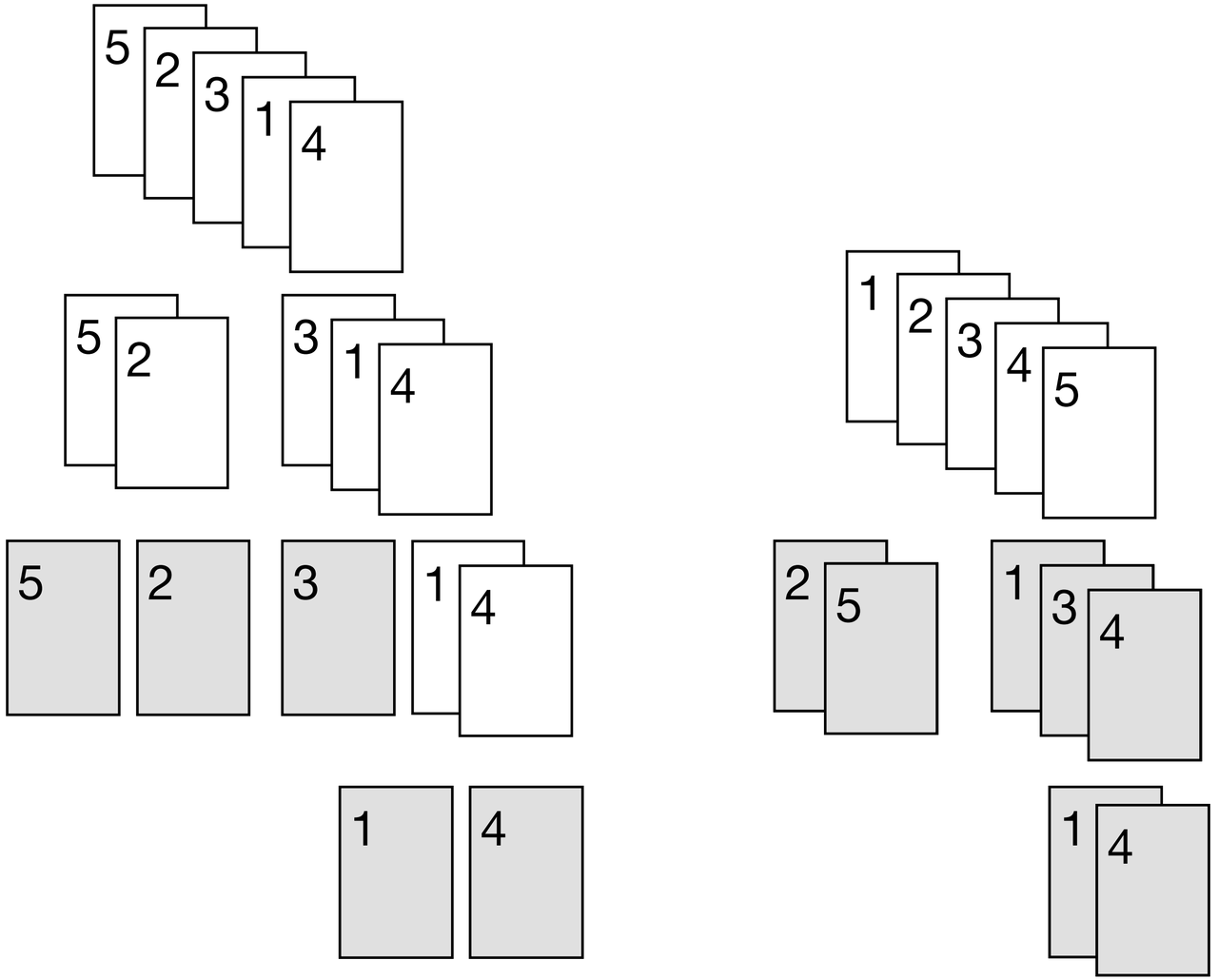


Universität Bremen

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



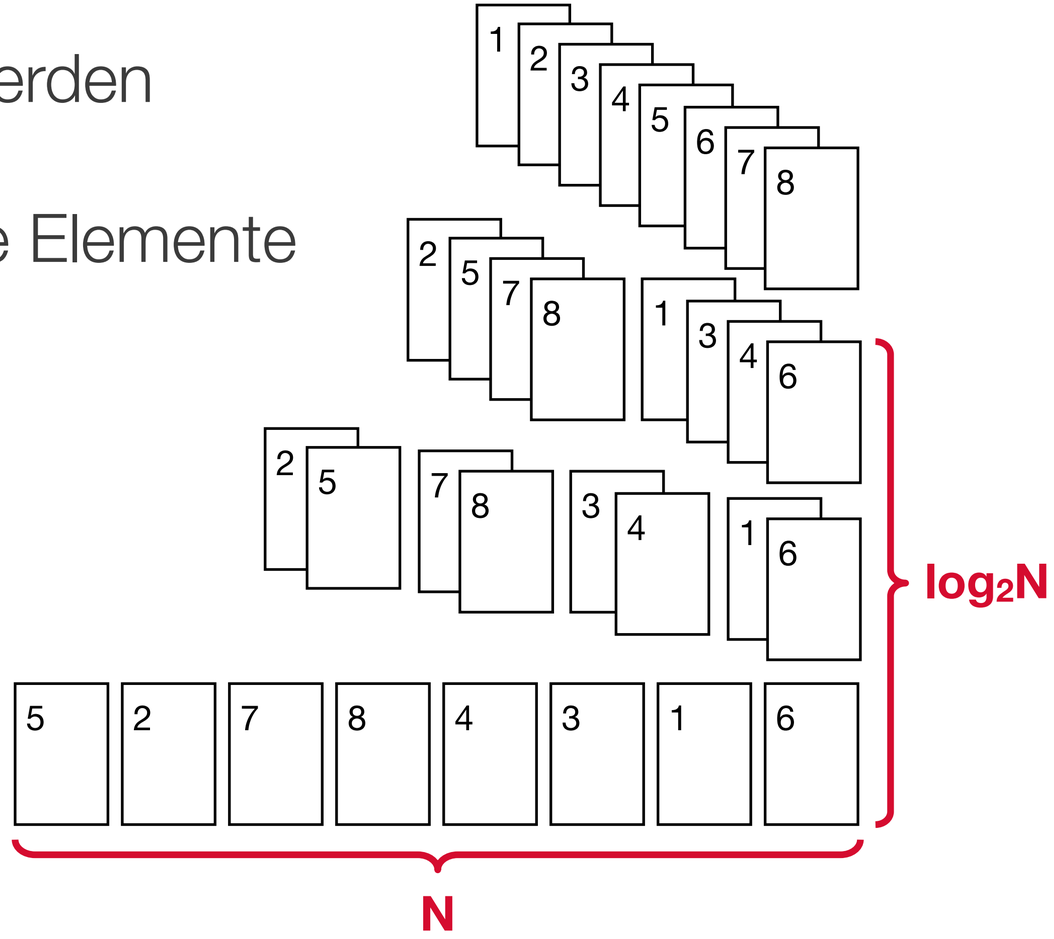
Universität Bremen

Sortieren durch Mischen

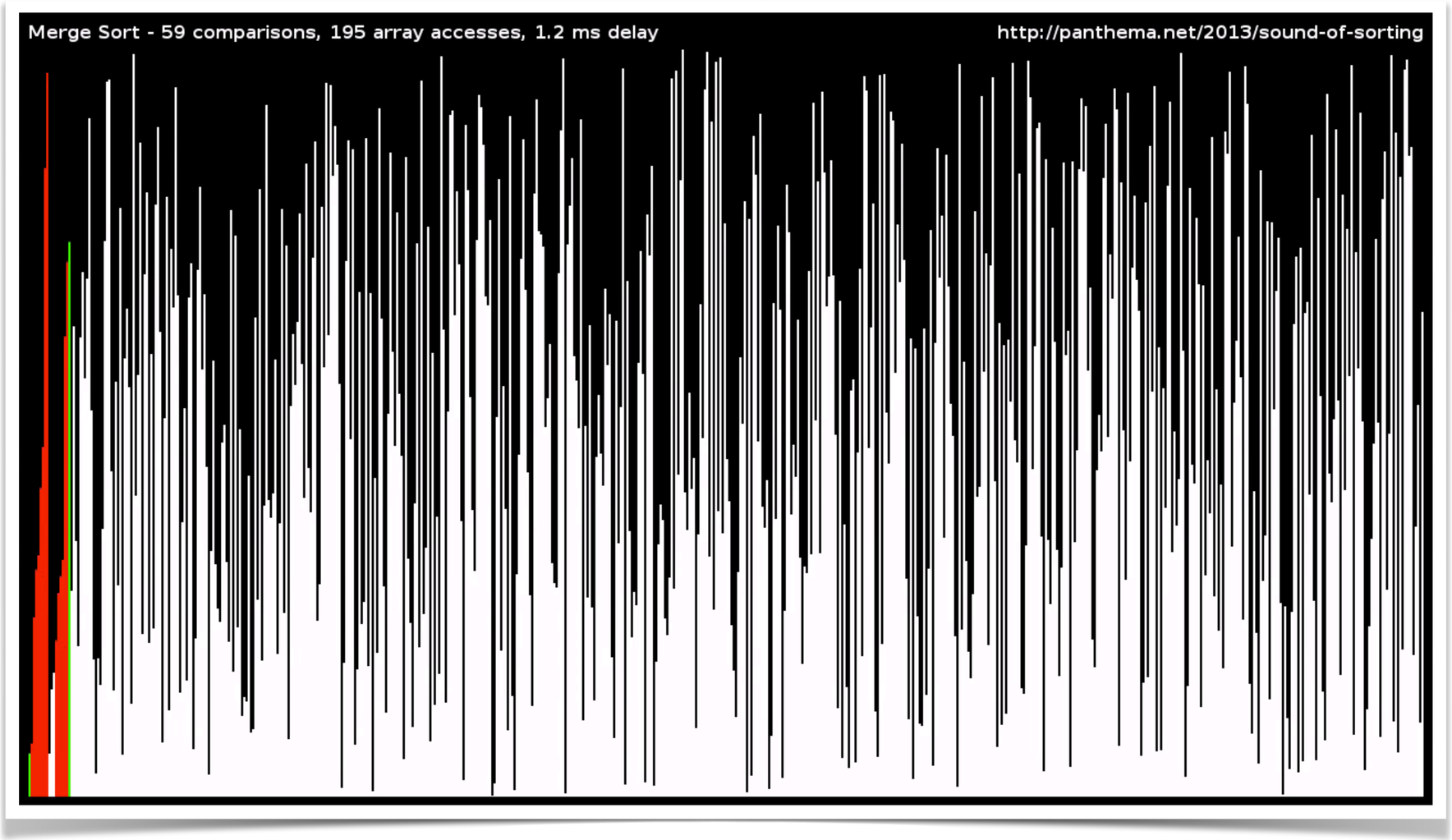
- Enthält die Folge kein oder nur ein Element, ist sie sortiert
- Ansonsten teile die Folge in zwei gleich große Hälften
- Sortiere die Hälften
- Mische die sortierten Hälften wieder zusammen
  - Vergleiche die beiden ersten Elemente der Hälften
  - Entnimm das kleinere von beiden und füge es dem Ergebnis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

Sortieren durch Mischen: Aufwand

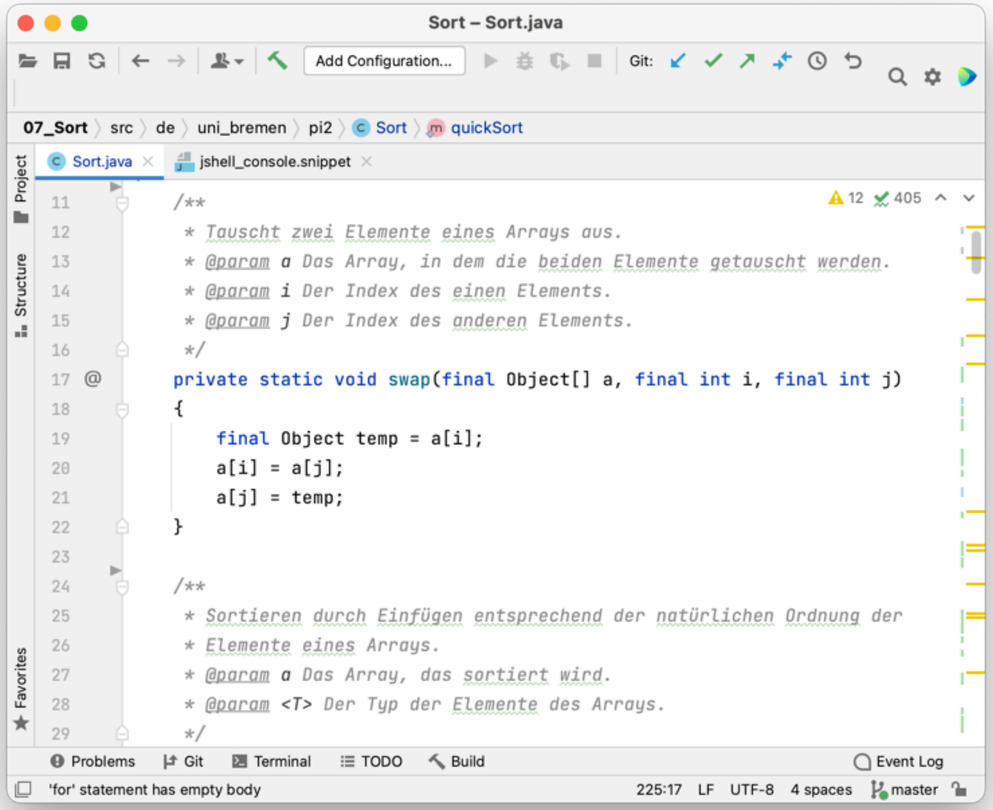
- Folge kann **$\log_2 N$**-mal in Hälften zerlegt werden
- Auf jeder Ebene der Zerlegung werden alle Elemente einmal gemischt: **O(N)**
- Insgesamt: **O(N log N)**
- Platzkomplexität: **2N** Elemente beim Mischen von Arrays



Merge Sort: Demo



Merge Sort: Demo



Universität Bremen

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)