Tagesprogramm

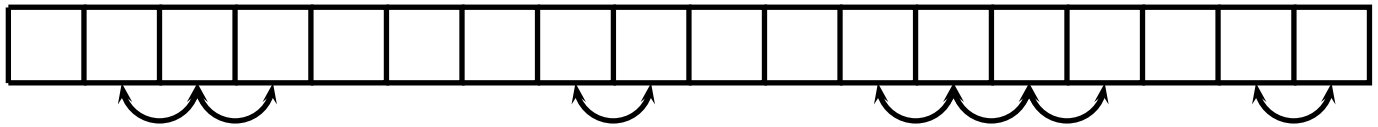
Bubblesort

Tree-Sort

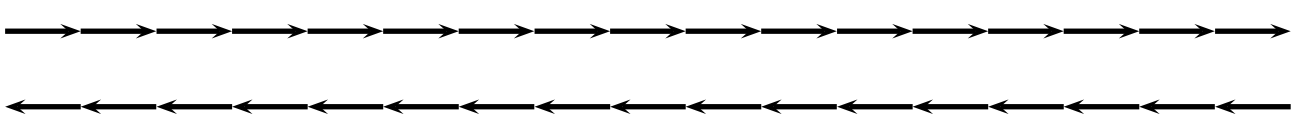
Quicksort

Bubblesort

ein Arraydurchlauf (von links nach rechts):



Grenzfälle:



1 Arraydurchlauf

n – 1 Arraydurchläufe

nötige Suchschritte:

$\approx n^2$ da gesamtes Array bis zu n Mal durchlaufen

Termination:

ja da nach n-tem Arraydurchlauf alle Positionen erreicht

Aufgabe: Bubblesort mit Zufallszahlen

Bubblesort war mit vorgegebenen sortierten Zahlen schnell fertig,
hat mit Zufallszahlen aber nicht in vernünftiger Zeit terminiert.

Warum ist das so?

A Fehler in der Implementierung, der sich nicht immer auswirkt

B Zufallszahlen sind durchschnittlich größer

C Abbruchbedingung trifft bei Zufallszahlen vielleicht nie zu

D Laufzeit hängt sehr stark von Sortierung ab

Aufgabe: Bubblesort ohne boolesche Variable

Schreiben Sie Bubblesort so um, dass der Algorithmus nach einer bestimmten Zahl an Arraydurchläufen terminiert (egal ob ein Durchlauf etwas geändert hat).

```
private void sort(int[] a) {
    boolean done;
    do { done = true;
         for (int i = 1; i < a.length; i++) {
             if (a[i] < a[i-1]) {
                 int h = a[i];  a[i] = a[i-1];  a[i-1] = h;
                 done = false;
             }
         }
    } while (! done);
}
```

Welche dieser Varianten von Bubblesort (ursprüngliche oder veränderte) optimiert den durchschnittlichen Fall, welche den schlechtesten?

Ausnutzen der Baumstruktur

Methode in Baumknoten:

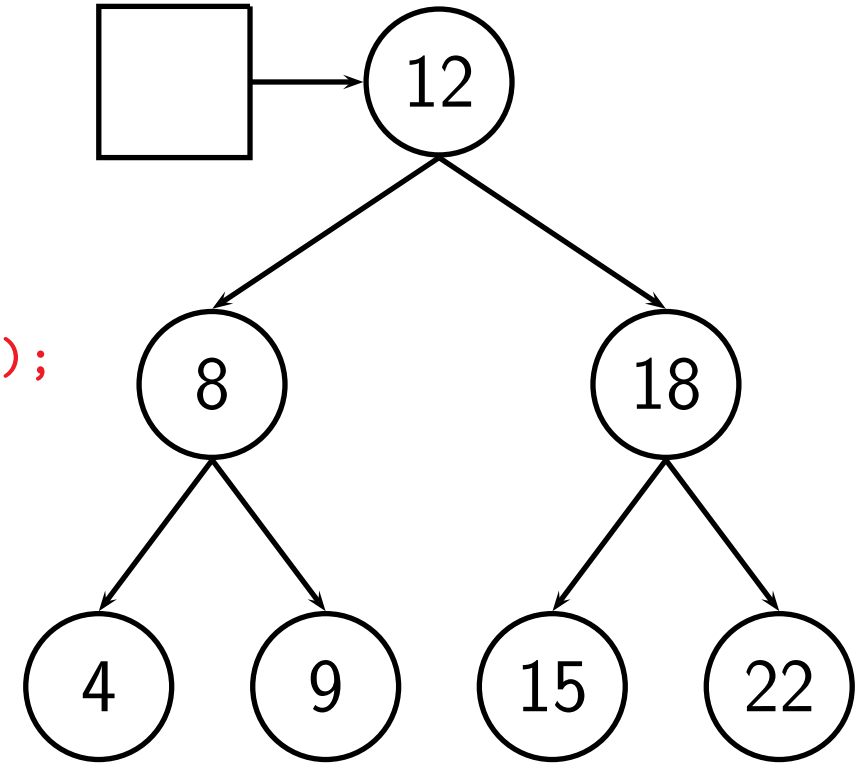
```
void printUp() {

    if (left != null) {
        left.printUp();
    }

    System.out.println(this.elem);

    if (right != null) {
        right.printUp();
    }

}
```



Aufgabe: Manipulationsaufwand

Die besprochene Variante von Tree-Sort ist sehr umständlich:
Zahlen in Array geschrieben – in Baum kopiert – wieder zurückkopiert.

Warum hat das Sortieren samt Umkopieren trotzdem nicht viel Zeit gebraucht?

A Im Vergleich zum Sortieren braucht das Umkopieren nur wenig Zeit

B Das Sortieren im Baum ist gratis, nur das Umkopieren wird bemerkt

C Wir haben einen schnellen Rechner, da ist Zusatzaufwand egal

Aufgabe: Tree-Sort mit sortierten Zahlen

Tree-Sort war mit Zufallszahlen schnell fertig,
hat mit sortierten Zahlen aber zum Absturz geführt (weil System-Stack zu klein).

Warum ist das so?

A System-Stack zu klein gewählt, Vergrößern nötig

B Lineare Suche da Baum zu Liste entartet ist

C Wahrscheinlichkeit hoch, dass eine Zufallszahl zum Abbruch führt

D Wahrscheinlich ein Implementierungsfehler

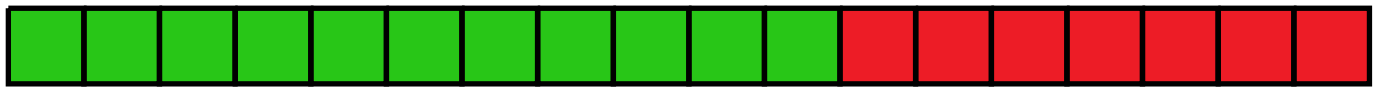
Divide and Conquer

Prinzip:

1. Teile Problem in kleinere Probleme auf
2. löse die kleineren Probleme (meist rekursiv)

Quicksort:

1. wähle ein Element als Pivot-Element p,
   teile Array in zwei Teile, sodass
   ein Teil Elemente <= p enthält
   und anderer Teil Elemente >= p enthält
2. mache dasselbe mit den beiden Teilen



Aufgabe: Kriterien für Sortierverfahren

Ordnen Sie die folgenden Kriterien von Sortierverfahren nach ihrer Wichtigkeit:

A auch ein kleiner System-Stack muss dafür reichen

B Abhängigkeit von vorheriger Sortierung soll klein sein

C im Durchschnitt soll die Laufzeit kurz sein

D das Verfahren soll einfach verständlich sein

die Wichtigkeit der Kriterien ist subjektiv,
es gibt keine richtige oder falsche Antwort (wohl aber bessere und schlechtere)