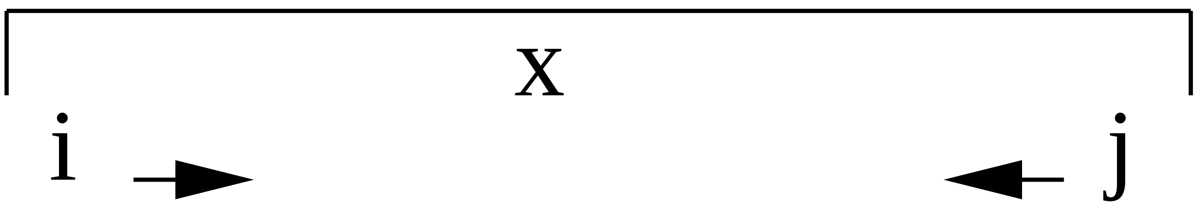
Was: Sortierung

Wie: Schnellsortierung
Rekursives Aufteilung in zwei Felder, die dann wieder sortiert werden.
Jedes Element der linken Hälfte des Feldes soll kleiner sein als alle Elemente der rechten Hälfte.

Warum: Performanztests

Schnellsortierung

Element x in der Mitte, Zähler i am Anfang des Feldes, Zähler j am Ende.



- Wenn das i-te Element kleiner ist als x, rückt der Zeiger weiter vor, maximal bis j.
- Ist eines größer als x, beginnt j das Feld hinunterzulaufen.
- Wenn das j-te Elemente größer ist als x, läuft j weiter, maximal bis i.
- Ist es kleiner als x, dann werden die Inhalte, auf die i und j zeigen, vertauscht.
- Treffen sich i und j, wird die Sortierung mit den beiden Feldern links und rechts des Felds, an dem sich i und j getroffen haben, wieder aufgerufen.

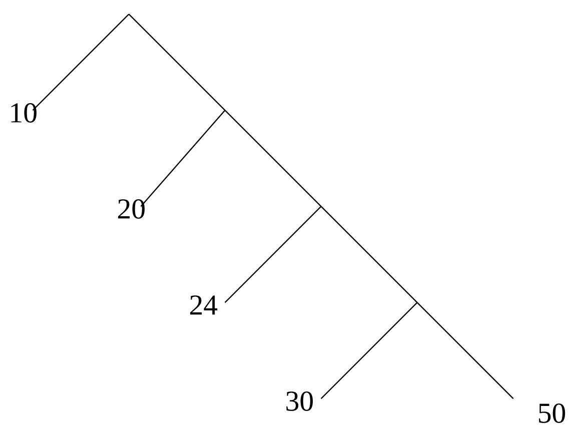
Aufwand der Schnellsortierung

Wenn sich i und j immer genau in der Mitte treffen, dann gibt es keinen Unterschied in der Laufzeitabschätzung zwischen Schnellsortierung und Sortieren durch Mischen. Der Aufwand ist in diesem Falle $O(nlogn)$ bei n Elementen.

Im Gegensatz zur Mischsortierung kann es hier aber vorkommen, daß nur ein Element in die untere Hälfte getan wird und j bis Feldende enthält alle anderen.

Feld: 20,30,10,24,50

x=10



Der schlimmste Fall der Laufzeitabschätzung ist $O(n^2)$.

```
public class AnimatedQuickSort {
    public static void sort (int [] a) {
        sort (a, 0, a.length - 1);
        ShowArray.show (a);
    }
    public static void sort (int[] a, int unten, int oben) {
        int tmp;
        int i = unten;
        int j = oben;
        int x = a[ (unten+oben) / 2];          // Pivotelement
        do {
            while (a[i] < x) {
                i++;                       // x fungiert als Bremse
                ShowArray.show (a, i, j);
            }
            while (a[j] > x) {
                j--;                       // x fungiert als Bremse
                ShowArray.show (a, i, j);
            }
            if ( i<=j ) {
                tmp = a[i];                        // Hilfsspeicher
                a[i] = a[j];                             // a[i] und
                a[j] = tmp;                 // a[j] werden getauscht
                i++;
                j--;
            }
        } while (i <= j);
```

```
            // alle Elemente der linken Haelfte sind kleiner
                   // als alle Elemente der rechten Haelfte
      if (unten < j) sort (a, unten, j);      // sortiere linke Haelfte
      if (i < oben ) sort (a, i, oben );      // sortiere rechte Haelfte
    }
}
```

Performanztest

Verfahren werden auf

- Effektivität, Korrektheit
- Effizienz, Laufzeit

hin untersucht.

Untersuchungsmethoden: Induktionsbeweis

Implementationen werden mit systematischen Experimenten getestet.

Laufzeit vs. Laufzeitabschätzung

- Wenn ein Programm exponentiell viel Zeit verbraucht, obwohl das Problem eigentlich in polynomieller Zeit lösbar sein sollte, dann ist es schlecht programmiert.

- Wenn es in den meisten Fällen polynomiell viel Zeit verbraucht, obwohl es im schlimmsten Falle exponentiell viel Zeit benötigt, ist das kein Widerspruch.

Systematisches Testen

- Was wollen Sie testen?
- Welche Daten brauchen Sie, um den Test durchzuführen?
- Wie erfassen Sie die Ergebnisse?
- Durchführen und Revidieren der Testläufe
- Interpretation und Kommunikation der Ergebnisse

Typische Fehler

Falsch:

"Es gibt einen Aufruf und Datensatz, bei dem mein Programm tut, was es soll."

"Es gibt einen Aufruf und Datensatz, bei dem mein Programm schnell ist."

Besser:

"Mein Programm tut immer, was es soll."

"Mein Programm liefert sein Ergebnis immer innerhalb von 10 Minuten."

Was soll getestet werden?

Benchmark Wollen Sie testen, ob das Programm eine festgelegte Anzahl von Fällen in der ebenfalls festgelegten Weise behandelt?

Zeitrahmen Wollen Sie testen, ob das Programm den Zeitrahmen einhält, der für verschiedene Aufgaben vorgegeben ist? So soll ein Roboter beispielsweise bei der Hindernisvermeidung in Sekundenbruchteilen ausweichen, darf bei der Wegeplanung ruhig 2 Minuten (aber nicht mehr!) verbrauchen und muß seine Fahrt neben einem Menschen her (dem er z.B. den Koffer trägt) auf das variierende Gehtempo des Menschen abstimmen können.

Neuer Benchmark Wollen Sie testen, ob Ihr Programm mehr Fälle bearbeiten kann als ein anderes (z.B. das bisher weltbeste Programm zur selben Aufgabe)? Achten Sie darauf, daß Sie nicht einfach nur andere Fälle bearbeiten!

Vergleich Wollen Sie testen, ob Ihr Programm schneller ist als ein anderes?

Laufzeit Wollen Sie das Laufzeitverhalten eines Programms in Bezug zur Größe der bearbeiteten Datenmenge testen? Definieren Sie präzise, was Sie unter der Größe verstehen!

Kriterium

- Ein binäres Kriterium stellt einfach fest, ob das tatsächliche Verhalten mit dem festgelegten Verhalten identisch ist – ja oder nein. Wir können dann feststellen, bei wieviel Prozent der Beispiele für einen Fall, das tatsächliche Verhalten verschieden vom festgelegten war. In der Statistik gibt es Resultate darüber, wieviele Beispiele für einen Fall notwendig sind, um diese Prozentzahl als Erwartungswert für einen Fehler zu interpretieren.

- Es gibt aber auch gleitende Kriterien, bei denen die Abweichung des tatsächlichen vom festgelegten Verhalten in eine reellwertige Zahl zwischen 0 und 1 abgebildet wird.

- Sie können auch komplexere Kriterien formulieren, wie etwa, daß der Erwartungswert für einen Fehler bei den selteneren Fällen höher ist, als bei den häufigen, oder daß die Abweichung bei den häufigen Fällen stets niedriger ist als bei den selteneren.

Beispiel: Sortieren

Wir wollen testen, unter welchen Umständen welches unserer Sortierprogramme das schnellste ist.

Als **Umstand** definieren wir die Sortiertheit der Eingabe, angegeben durch eine Zahl zwischen 0 und 1:

1 völlig ungeordnet

0 völlig geordnet.

Als **Schnelligkeit** definieren wir die Anzahl der tatsächlich ausgeführten Schritte. Unser Kriterium ist also die kleinste von 3 Anzahlen von Schritten.

Die Daten lassen wir uns generieren (Zufallsgenerator).

Die Ergebnisse werden ausgegeben.

Wir vergleichen die Ergebnisse aller drei Lernverfahren.

Wir fassen die Ergebnisse zusammen.

Experimentdesign

Wir variieren Größe der Eingabe und Sortiertheit und testen jedes Sortierprogramm. Dann vergleichen wir die Ergebnisse.

Größe: 1 .. 1000

Schrittweite: etwa logarithmisch (2,32,128,256,512,1000) 6

Sortiertheit: 0 .. 1

Schrittweite: (0, 0.25, 0.5, 0.75, 1) 5

Das sind $1 + 5 \cdot 5$ Läufe des Testprogramms für ein Sortierprogramm. Von der Analyse her wissen wir, daß sich Misch- und Selektionssortierung durch die Geordnetheit nicht beirren lassen. Deshalb führen wir nur für die Schnellsortierung $1 + 5 \cdot 5$ Läufe, für Misch- und Selektionssortierung nur 6 Läufe durch.

Ergebnis

Testdaten		Quick	Merge	Selection
Größe	Ordnung			
2	0.5	1	2	2
32	0	104	160	527
32	0.25	101	160	527
32	0.5	85	160	527
32	0.75	84	160	527
32	1	86	160	527
128	0	650	896	8255
128	0.25	595	896	8255
128	0.5	566	896	8255
128	0.75	557	896	8255
128	1	595	896	8255
256	0	1547	2048	32895
256	0.25	1419	2048	32895
256	0.5	1249	2048	32895
256	0.75	1540	2048	32895
256	1	1265	2048	32895
512	0	3596	4608	131327
512	0.25	3203	4608	131327
512	0.5	3328	4608	131327
512	0.75	2955	4608	131327
512	1	4056	4608	131327
1000	0	7988	9976	500499
1000	0.25	7464	9976	500499
1000	0.5	7327	9976	500499
1000	0.75	7850	9976	500499
1000	1	6213	9976	500499

Interpretation und Verfeinerung

Wir sehen, daß Quicksort immer das schnellste Verfahren ist. Allerdings variiert die Anzahl der Schritte innerhalb eines *Umstands* der Messung. Daher ist es nicht korrekt, nur die Schrittzahl eines Laufes anzugeben.

128 Elemente, 1 Varianz:

595, 594, 646, 475, 651, 523, 552, 538, 485, 739

- Mittelwert (579.8)
- tiefster und höchster Wert (475, 739)
- Median (zwischen 552 und 594)

Wir müssen also unser Experiment verfeinern: für Quicksort werden für alle Varianzen (außer 0) 10 Läufe durchgeführt.

Wir können dann untersuchen, ob die Varianz der Eingabe einen Einfluß auf die Varianz der Ergebnisse hat.

Varianz

Zufallsvariable Y – hier: Laufzeit in Schritten

mit gemessenen Werten $y_1, y_2, ..., y_n$

Erwartungswert E[Y] $= \sum_{i=1}^{n} y_i p(Y = y_i)$

Varianz V[Y] $= E[(Y - E[Y])^2]$

Standardabweichung $\sigma = \sqrt{E[(Y - E[Y])^2]}$

Beispiel:

Y=595, 594, 646, 475, 651, 523, 552, 538, 485, 739

E[Y]= 579,8

V[Y]= $\sum_{i=1}^{n} (y_i - 579,8)^2 0,1 =$ 6093,86

Standardabweichung 78,06

```
import AlgoTools.IO;

class TestSort {
    static public void main (String [] argumente) {
        int verfahren;
        int anzahl;
        double pv;

        IO.println ("1: SelectionSort");
        IO.println ("2: QuickSort");
        IO.println ("3: MergeSort");

        do {
            verfahren = IO.readInt ("[1..3]? ");
        } while (verfahren < 1 || verfahren > 3);

        do {
            anzahl = IO.readInt ("Anzahl zu sortierender
Elemente (>1)? ");
        } while (anzahl < 1);

        do {
            pv = IO.readDouble ("Pseudo-Varianz [0..1]? ");
        } while (pv < 0 || pv > 1);

        int [] testArray = new int [anzahl];
```

```
for (int i = 0; i < testArray.length; i++) {
    testArray [i] = ((int) (i + pv * (Math.random ()
− 0.5) * testArray.length)
    + testArray.length) % testArray.length;
}

ShowArray.display = IO.readBoolean ("Animation
anzeigen (=> keine Zeitmessung)? ");

IO.println ("Bitte warten...");

double startTime = System.currentTimeMillis ();

switch (verfahren) {
    case 1:   AnimatedSelectionSort.sort
(testArray); break ;
    case 2:   AnimatedQuickSort.sort (testArray);
break ;
    case 3:   AnimatedMergeSort.sort (testArray);
break ;
}

if (ShowArray.display) {
    ShowArray.showArray.repaint (); // Anzeige
Endzust. erzwingen}
else {   // zeit nur anzeigen, wenn nicht anim.
IO.println ("Laufzeit: "
    + (System.currentTimeMillis () − startTime) /
```

```
1000 + "Sek.");
          }

          IO.println ("Schritte: "+ ShowArray.counter);
     }
 }
```