

Bedienungsanleitung für JSort

Lars Hupel, Thomas Milde, Richard Tschirschnitz

19. Januar 2007

Inhaltsverzeichnis

1	Einleitung	1
2	Installation	1
3	Tour durch das Programm	1
3.1	Der Registerreiter „Sortieren“	1
3.2	Der Registerreiter „Laufzeitdiagramm“	1
3.3	Der Registerreiter „Visualisierung“	2
3.4	Der Registerreiter „Quiz“	2
3.5	Der Registerreiter „eigene Sortierverfahren“	2
4	Beschreibungen der Registerreiter	2
4.1	Die Startseite	2
4.2	Der Registerreiter „Sortieren“	3
4.3	Der Registerreiter „Laufzeitdiagramm“	3
4.4	Der Registerreiter „Visualisierung“	4
4.5	Der Registerreiter „Quiz“	5
4.6	Der Registerreiter „eigene Sortierverfahren“	5
5	Die Sortieralgorithmen	6
5.1	Bubblesort	6
5.2	Heapsort	6
5.3	Insertionsort	7
5.4	Mergesort	7
5.5	Quicksort	7
5.6	Selectionsort	7
5.7	Shakersort	8
5.8	Shellsort	8
6	Die Skriptsprache JSortScript	8

1 Einleitung

Die Tätigkeit, die jeder Computer am meisten ausführt, ist das Sortieren. Deswegen soll das Programm JSort jedem Benutzer ermöglichen, ein Grundverständnis dafür aufzubauen, wie im Computer sortiert wird. Hierzu stehen neben dem einfachen Sortieren von Zahlenfolgen Animationen und Laufzeitanalysen von bekannten Sortierverfahren zur Verfügung. Außerdem kann der Benutzer mithilfe einer einfachen Skriptsprache eigene Sortieralgorithmen implementieren und sein Wissen im Quiz überprüfen.

Dieses Programm wurde im Rahmen des Java Stars 2006 - Sun Microsystems Award durch das Team nlogn erstellt. Die Gruppenmitglieder sind Lars Hupel, Thomas Milde und Richard Tschirschnitz.

2 Installation

Das Programm selbst benötigt keine Installation, aber es ist notwendig, dass auf dem Computer, auf dem das Programm ausgeführt werden soll, das Java Runtime Environment (JRE) in der Version 1.5 oder höher installiert ist. Das aktuelle JRE kann auf der Seite java.sun.com heruntergeladen werden und das JRE in der Version 1.5 liegt dem Programm bei. Anweisungen zur Installation des JRE sind in der `readme.txt` auf der CD zu finden. Um das Programm auszuführen, muss die Datei `JSort.jar` mit der Java Virtual Machine geöffnet werden.

3 Tour durch das Programm

Dieses Kapitel beschreibt eine kurze Tour durch das Programm, in der man einige Funktionen ausprobiert. Die restlichen Kapitel beschreiben die Funktionen genauer.

Beim Öffnen des Programmes sieht man die Startseite. Diese bietet einen kurzen Überblick über die Funktionen des Programmes. Man klickt nun auf die Schaltfläche „Einfaches Sortieren“ und gelangt dadurch in den Registerreiter „Sortieren“.

3.1 Der Registerreiter „Sortieren“

Hier kann man Ganzzahlen sortieren, um zu überprüfen, dass die Sortieralgorithmen auch wirklich funktionieren. Hierzu betätigt man in der Mitte die Schaltfläche „Automatisch füllen“, um Zufallszahlen in das Feld auf der linken Seite einzufügen. Danach aktiviert man das Kästchen „Ausgabe der sortierten Elemente“ und betätigt die Taste „Sortieren“. Dadurch wird die links eingegebene Folge von Zufallszahlen mit dem Verfahren Bubblesort sortiert und auf der rechten Seite ausgegeben. Sie können sich nun selbst davon überzeugen, dass korrekt sortiert wurde und dann am linken Rand des Fensters auf „Laufzeitdiagramm“ klicken.

3.2 Der Registerreiter „Laufzeitdiagramm“

Hier klickt man auf die Schaltfläche „Sortieren“, um ein Laufzeitdiagramm des Algorithmus Bubblesort zeichnen zu lassen. Man erkennt deutlich, dass der Verlauf einer quadratischen Funktion ähnelt. Zum Vergleich betätigt man die Schaltfläche „Sortierverfahren“ und wählt im erscheinenden Fenster im Feld „Sortierverfahren“ den Eintrag „Heapsort“ aus und betätigt die Schaltfläche „Auswählen“. Nun betätigt man die Taste „Sortieren“

erneut und sieht, dass die Kurve hier deutlich flacher ist. Durch solche Diagramme kann man abschätzen, wie schnell ein Sortieralgorithmus ist. Um nun weitere Funktionen zu erkunden, klickt man auf „Visualisierung“.

3.3 Der Registerreiter „Visualisierung“

Diese Funktion bietet die Möglichkeit, sich anzuschauen, wie ein Sortierverfahren sortiert. Dazu aktiviert man zuerst die beiden Kästchen „horizontale Grenzen“ und „vertikale Grenzen“, die sich unten links befinden. Nun wählt man durch Betätigung der Taste „Sortierverfahren“ und Auswahl des Eintrages „Bubblesort“ im oberen Feld des erscheinenden Fensters das Sortierverfahren Bubblesort und betätigt die Taste „Sortieren“, um sich den Ablauf des Sortierverfahrens anzuschauen. Als nächstes erfolgt ein Klick auf „Quiz“.

3.4 Der Registerreiter „Quiz“

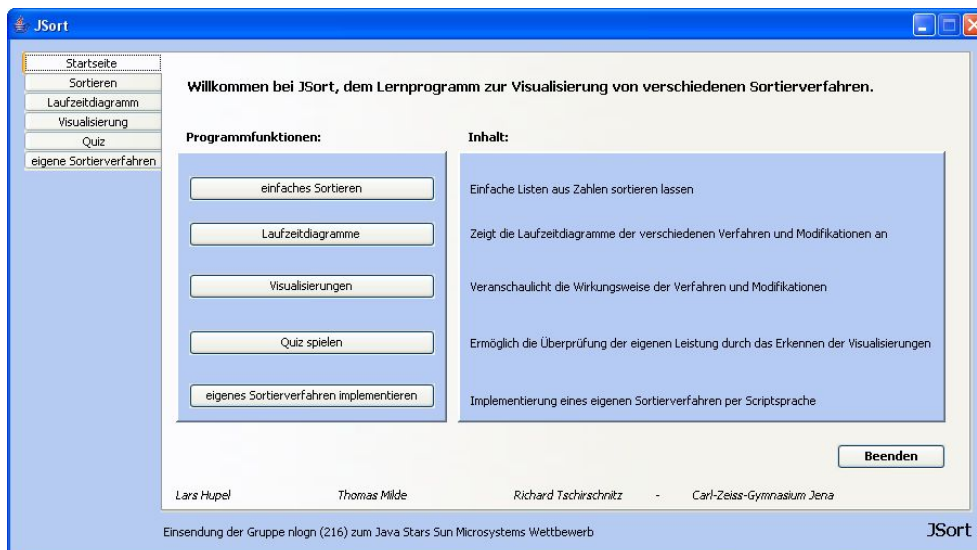
Hier können Sie überprüfen, wieviel Sie schon über Sortierverfahren wissen. Dazu betätigt man die Taste „Anzeigen“, um eine Visualisierung eines Algorithmus anzuschauen. Wenn man nun schon über größeres Wissen über Sortierverfahren verfügt, kann man versuchen zu erkennen, um welchen Sortieralgorithmus es sich handelt, diesen in der Auswahlliste selektieren und die Taste „Tipp abgeben“ betätigen. Daraufhin erhält man eine Bewertung, es wird also mitgeteilt, ob der Tipp richtig war und, falls der Tipp falsch war, wird die korrekte Antwort angezeigt. Um nun auch die letzte Funktion auszuprobieren, klickt man im linken Teil des Formulars auf „eigene Sortierverfahren“.

3.5 Der Registerreiter „eigene Sortierverfahren“

In diesem Registerreiter kann man seine eigenen Ideen für Sortierverfahren verwirklichen. Eine Beschreibung hierzu erfolgt in der ausführlichen Erläuterung dieses Registerreiters.

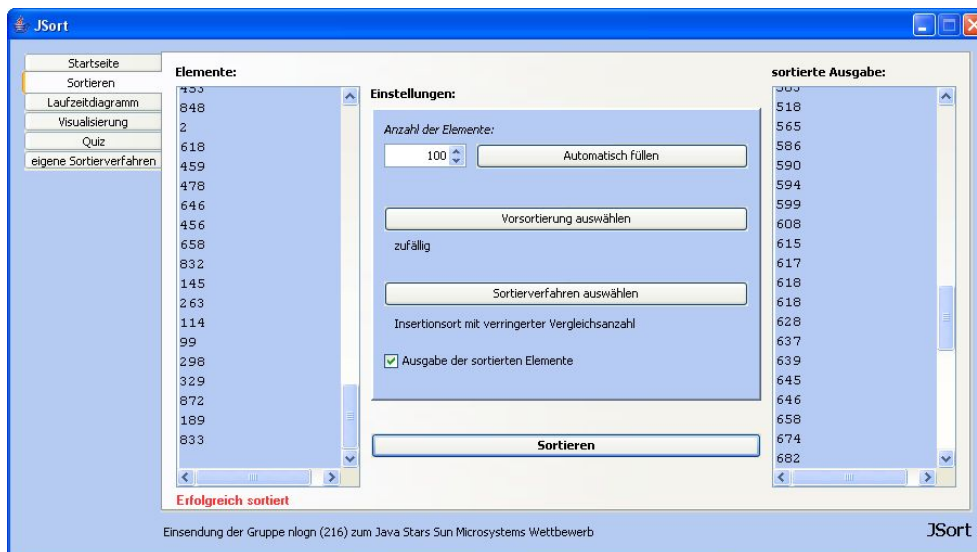
4 Beschreibungen der Registerreiter

4.1 Die Startseite



Dieser Registerreiter hat keine eigenen Funktionen, sondern gibt nur eine Übersicht über die Funktionen des Programmes.

4.2 Der Registerreiter „Sortieren“

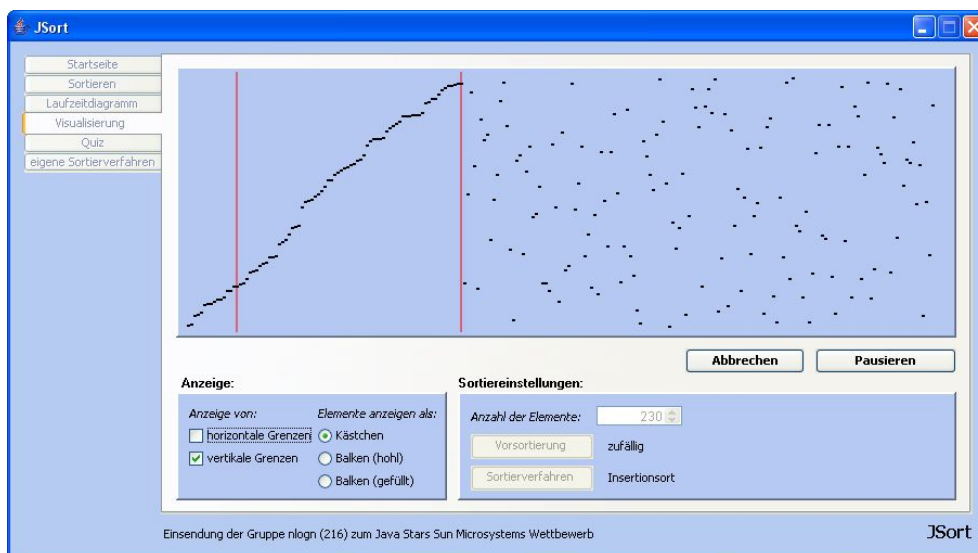
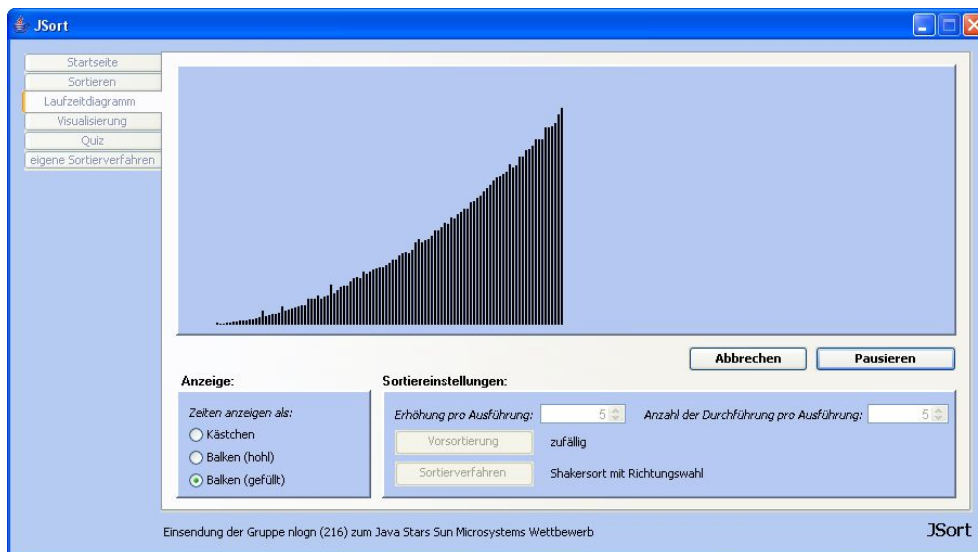


Dieser Registerreiter ist dazu gedacht, einfache Ganzzahlen zu sortieren, z.B. um zu überprüfen, dass die Algorithmen funktionieren.

Die Einstellung „Anzahl der Elemente“ kann genutzt werden, um einzustellen, wie viele Elemente bei Betätigung der Taste „Automatisch füllen“ in das linke Eingabefeld eingefügt werden sollen. Für dieses automatische Füllen ist auch die Einstellung der Vorsortierung relevant. Durch Betätigung der Taste „Sortieren“ werden die links eingegebenen Daten sortiert und, falls die Option „Ausgabe der sortierten Elemente“ aktiviert ist, im rechten Feld ausgegeben.

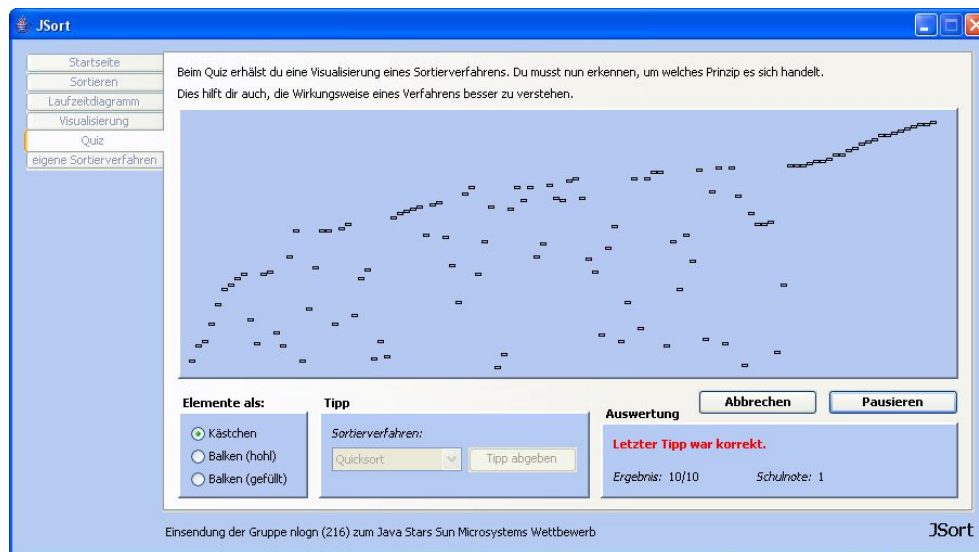
4.3 Der Registerreiter „Laufzeitdiagramm“

Dieser Registerreiter hat den Zweck, zu veranschaulichen, wie sich die Laufzeit der Algorithmen bei wachsender Elementanzahl verhält. Die Optionen im Bereich „Anzeige“ legen fest, in welcher Form das Diagramm erscheinen soll. Die Vorsortierung gibt an, ob das Array vor dem Starten des Sortierverfahrens schon sortiert, falschherum sortiert oder unsortiert sein soll. Die Einstellung „Erhöhung pro Ausführung“ gibt an, um wie viele Elemente länger das Array in jeder Stufe werden soll. „Anzahl der Durchführungen pro Ausführung“ gibt an, wie oft mit jeder Arraylänge sortiert werden soll. Wenn diese Zahl erhöht wird, wird der Verlauf der Kurve deutlicher, weil „Ausreißer“ dann weniger ins Gewicht fallen. Durch Betätigung der Schaltfläche „Sortieren“ wird damit begonnen, das Laufzeitdiagramm zu erzeugen. Hier wird jede Zeit eingetragen und immer, wenn die obere Grenze erreicht wird, wird das Diagramm gestaucht, sofern das noch möglich ist. Während dieser Vorgang läuft, kann dieser mit den entsprechenden Schaltflächen unterbrochen („Pausieren“) und abgebrochen („Abbrechen“) werden.



4.4 Der Registerreiter „Visualisierung“

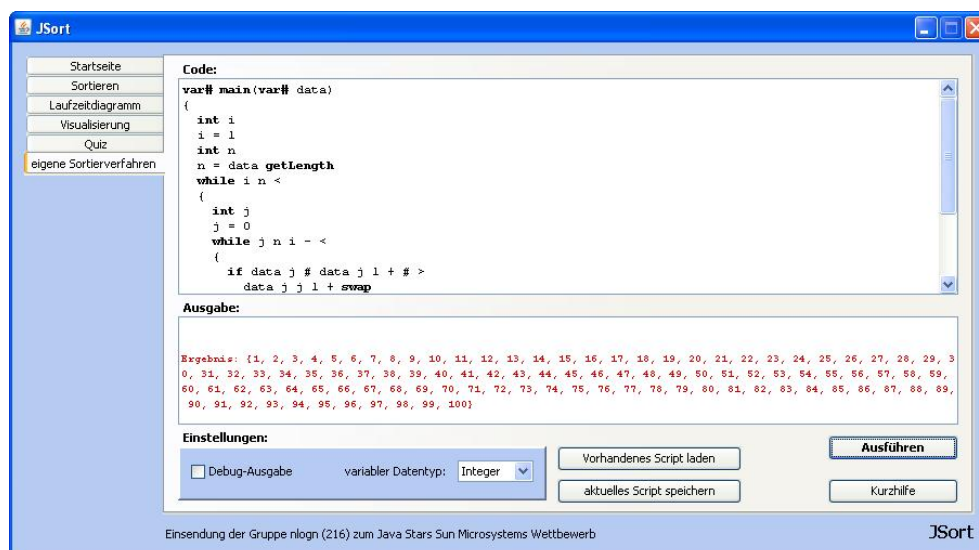
In diesem Registerreiter kann beobachtet werden, wie das Sortieren mit verschiedenen Sortieralgorithmen abläuft. Hierzu gibt es die Optionen „vertikale Grenzen“ und „horizontale Grenzen“, die festlegen ob die entsprechenden Linien sichtbar sein sollen, die bestimmte Stellen hervorheben, die aktuell für den Sortieralgorithmus entscheidend oder signifikant sind. Die Vorsortierung legt auch hier fest, ob das Array vor dem Sortieren mit dem gewählten Algorithmus unsortiert, sortiert oder umgekehrt sortiert sein soll. Das Feld „Anzahl der Elemente“ legt fest, wie viele Elemente sortiert werden sollen. Die Schaltfläche „Sortieren“ startet die Animation des Sortierverfahrens. Dieser Vorgang kann mit „Pausieren“ unterbrochen und mit „Abbrechen“ abgebrochen werden.



4.5 Der Registerreiter „Quiz“

Hier kann man sein Wissen über Sortieralgorithmen prüfen, indem man anhand der Animation des Verfahrens entscheidet, um welches Verfahren es sich handelt. Auch hier kann bei der Ansicht zwischen Kästchen und hohlen bzw. gefüllten Balken gewählt werden. Um das Quiz auszuführen, sollte man sich über die Taste „Anzeigen“ die Visualisierung anschauen, was man durch die Taste „Wiederholen“ auch mehrmals tun kann. Wenn man das Sortierverfahren erkannt hat, sollte man im Feld „Sortierverfahren“ einstellen, welches Verfahren man erkannt hat und die Taste „Tipp abgeben“ betätigen. Man erhält vom Programm immer eine Auswertung, die mitteilt, ob man richtig getippt hat und gegebenenfalls die Antwort korrigiert. Außerdem wird immer die Gesamtpunktzahl und die erreichte Punktzahl angezeigt und eine Schulnote vergeben.

4.6 Der Registerreiter „eigene Sortierverfahren“



In diesem Registerreiter kann man eigene Sortierverfahren implementieren. Dazu wird die Skriptsprache

JSortScript verwendet. Der Quelltext wird im oberen Feld eingegeben. Die Debug-Ausgabe gibt Statusmeldungen des Parsers aus, was Fehler im Skript unter Umständen leichter auffindbar macht. Der variable Datentyp ist der Datentyp, der für den Typ `var` eingesetzt wird. Im Ausgabefeld erscheint, sofern diese aktiviert ist, die Debug-Ausgabe. Außerdem erscheinen dort die Ausgabe des Skripts und Fehlermeldungen bei fehlerhaften Skripten. „Kurzhilfe“ zeigt einen kurzen Hilfetext zur Skriptsprache an. Die Ausführung des Skripts wird mit der Schaltfläche „Ausführen“ gestartet. Nach der erfolgreichen Durchführung des Skripts wird der Rückgabewert der Funktion `main` ausgegeben, weswegen es sich empfiehlt, das sortierte Array zurückzugeben.

5 Die Sortieralgorithmen



In vielen Registrierreitern steht die Schaltfläche „Sortierverfahren“ zur Verfügung. Diese öffnet ein Fenster mit zwei Auswahlfeldern. Das obere Feld gibt den grundlegenden Sortieralgorithmus an. Die untere Auswahl ermöglicht das Auswählen der Modifikation. Zum Beispiel sind dies Optimierungen des grundlegenden Algorithmus.

5.1 Bubblesort

Bubblesort durchläuft ein zu sortierendes Feld der Länge n n -mal. Dabei werden immer alle Paare von aufeinanderfolgenden Elementen verglichen und, falls sie noch nicht geordnet sind, getauscht. Die Optimierungen beziehen sich hier auf einen vorzeitigen Abbruch, falls das Array vor dem eigentlichen Ende des Algorithmus sortiert ist und auf das Verkleinern des zu durchlaufenden Bereiches bei jedem Durchlauf (es wird immer das größte Element nach hinten gebracht).

5.2 Heapsort

Bei Heapsort wird die zu sortierende Datenstruktur als Baum aufgefasst, indem jedes Element als ein Knoten aufgefasst wird, der maximal zwei Söhne hat. Eine Ebene des Baumes steht hier immer direkt aufeinanderfolgend in der Datenstruktur. Es wird nun, in der Mitte der Datenstruktur beginnend, jeder Knoten versickert. Das heißt, dass der Knoten so oft mit einem seiner Söhne getauscht wird, bis er entweder ein Blatt ist, also keine Söhne mehr hat, oder beide Söhne größer sind als der Knoten. Der Knoten wird dabei immer mit dem Sohn mit dem höchsten Wert getauscht. Anschließend wird vom Ende der Struktur beginnend jeder Knoten und jedes Blatt mit der Wurzel vertauscht (die Wurzel wird „abgehackt“) und die neue Wurzel versickert. Beim Versickern wird aber der Bereich der Datenstruktur außer Acht gelassen, der mit der soeben mit der Wurzel getauschten Stelle beginnt. Die Optimierung „trinäres Heapsort“ versteht die Datenstruktur nicht als binären, sondern als trinären Baum.

5.3 Insertionsort

Bei Insertionsort wird die Datenstruktur von links nach rechts durchlaufen und bei jedem Element der gesamte Bereich links neben dem aktuellen Element vom aktuellen Element zum Anfang durchlaufen. Dabei wird immer dann eine Vertauschung durchgeführt, wenn zwei aufeinanderfolgende Datenelemente nicht korrekt sortiert sind. Dadurch wird das aktuelle Element in diesen Bereich korrekt einsortiert. Die Optimierungen beziehen sich hier auf ein effizienteres Finden der Einfügeposition.

5.4 Mergesort

Mergesort arbeitet nach dem Prinzip „Teile und herrsche“. Zum Sortieren wird hier die zu sortierende Datenstruktur in eine linke und eine rechte Hälfte geteilt und der Sortieralgorithmus rekursiv aufgerufen. Wenn der Algorithmus mit nur einem Element aufgerufen wird, ist die Abbruchbedingung erreicht. Danach folgt in jeder Rekursionsstufe der Schritt des Zusammenführens der beiden Teilfelder. Hierzu wird immer das Datum ausgewählt, das das kleinere der beiden ist, dieses wird in ein neues Array eingefügt und ein Iterator, der die erste Stelle des Teilfeldes, aus dem das Datum genommen wurde kennzeichnet, wird um eins nach hinten verschoben. Zum Schluss wird das Array wieder auf das Ursprungsfeld übertragen. Da diese out-of-place-Implementation nur schlecht visualisierbar ist, gibt es auch die Möglichkeit, das Verfahren in-place zu realisieren. Dazu wird das Datum nicht in ein externes Array geschrieben, sondern an das Ende des sortierten Bereiches der Ursprungsstruktur gestellt und gegebenenfalls der linke Teil dieser Struktur um eine Stelle nach rechts verschoben. Die Optimierung „natürliches Mergesort“ ist darauf ausgerichtet, bereits sortierte Folgen zu finden und zusammenzufügen.

5.5 Quicksort

Der Algorithmus Quicksort arbeitet ebenfalls nach dem Prinzip „Teile und Herrsche“. Bei Quicksort wird in jeder Rekursionsstufe die zu sortierende Datenstruktur so aufgeteilt, dass ein bestimmten Element, das Pivotelement, die kleineren von den größeren Werten trennt. Alle Werte links vom Pivotelement sind kleiner oder gleich diesem und alle Elemente rechts des Pivotelements sind größer oder gleich dem Pivotelement. Diese beiden Teile werden über einen rekursiven Aufruf separat sortiert. Ein Zusammenführen ist nicht nötig, da bereits rechts die großen und links die kleinen Elemente stehen und durch die Sortierung der Teilstrukturen die komplette Folge sortiert ist.

Das Abtrennen der kleinen von den großen Elementen erfolgt, indem ein zufällig ausgewähltes Element als Pivotelement festgelegt und an das Ende der Struktur gestellt wird. Nun wird von links nach einem Element gesucht, das größer ist als das Pivotelement und von rechts wird nach einem Element gesucht, das kleiner als das Pivotelement ist. Diese beiden Elemente werden vertauscht und der Vorgang wird solange wiederholt, bis die beiden Suchen sich treffen. Zum Schluss wird das Pivotelement mit dem Element vertauscht, auf das beide Suchen gerade zeigen, um es an seine korrekte Position zu stellen. Die Optimierungen beziehen sich darauf, die Ineffizienz von Quicksort bei kurzen Folgen zu kompensieren sowie den schlimmsten Fall (das Pivotelement ist immer das erste oder letzte Element und somit ist die Laufzeit quadratisch) zu vermeiden.

5.6 Selectionsort

Bei Selectionsort wird der zu sortierende Datenbereich in einen sortierten und einen unsortierten Bereich geteilt. Der sortierte Bereich hat anfänglich immer die Größe Null. Es wird nun in einem Array der Größe n n -mal

das kleinste Element des unsortierten Bereiches sequentiell gesucht, mit dem ersten Element des unsortierten Bereiches getauscht und der sortierte Bereich wird um eins vergrößert. Dadurch wächst der sortierte Bereich immer weiter an und zum Schluss ist die Größe des unsortierten Bereiches null. Die Optimierungen beziehen sich darauf, das kleinste Element effizienter auszuwählen.

5.7 Shakersort

Shakersort, auch BiDiBubblesort genannt, beruht auf dem gleichen Prinzip wie Bubblesort, aber die Datenstruktur wird nicht nur von links nach rechts, sondern abwechselnd nach rechts und nach links durchlaufen. Dabei wird beim Durchlaufen nach rechts die rechte Grenze um eins nach links und beim Durchlaufen nach links die linke Grenze um eins nach rechts verschoben, denn es wird immer das größte bzw. kleinste Element an die entsprechende Grenze gebracht. Die Optimierungen beziehen sich darauf, immer die effizientere Richtung zu wählen und den Algorithmus dann abubrechen, wenn die Struktur sortiert ist.

5.8 Shellsort

Shellsort basiert auf dem gleichen Prinzip wie Insertionsort, aber es wird *h-sortiert*, das bedeutet, dass Shellsort auf der Datenstruktur ausgeführt wird, aber nur jedes *h*-te Element betrachtet wird. Die Elemente zwischen den *h*-ten Elementen werden ignoriert. Für dieses *h*-sortieren ist eine *h*-Folge nötig, die vorgibt, in welcher Reihenfolge *h* welche Werte annehmen soll. Diese *h*-Folgen sind auch der wichtigste Faktor für die Komplexität des Algorithmus und somit sind die verschiedenen *h*-Folgen hier auch die Optimierungen des Algorithmus.

6 Die Skriptsprache JSortScript

Die Skriptsprache JSortScript arbeitet auf Basis der Notationsform Postfix. Der Mittelpunkt eines jeden Skriptes ist die Funktion *main*, die ein Array des variablen Datentyps `var` übernimmt. Dies ist das Array, das sortiert werden soll. Als Beispiel für die Skriptsprache ist hier das Sortierverfahren Bubblesort definiert:

```
var# main(var# data)
{
  int i
  i = 1
  int n
  n = data.getLength
  while i n <
  {
    int j
    j = 0
    while j n i - <
    {
      if data j # data j 1 + # >
        data j j 1 + swap
      j inc
    }
  }
}
```

```

    i inc
  }
  data
}
```

Der Funktionskopf ist immer in der Form Rückgabety *Funktionsname*(*Parameterliste*). Die Funktion *main* muss immer als Parameter ein Array des Datentyps *var* übernehmen. In der Skriptsprache existieren die folgenden Datentypen:

Bezeichnung	Bedeutung
<i>int</i>	Ganzzahl
<i>float</i>	Fließpunktzahl (Dezimalzahl)
<i>string</i>	Zeichenkette
<i>bool</i>	Wahrheitswert
<i>var</i>	variabler Datentyp (kann bei jeder Ausführung mit einem der anderen Datentypen ersetzt werden)

In einer Funktion können Variablen deklariert werden. Dies geschieht in der Form *Datentyp Bezeichner*. Ein Bezeichner kann alle Buchstaben und Ziffern sowie Unterstriche, Bindestriche und Punkte enthalten. Das erste Zeichen darf jedoch keine Ziffer und kein Bindestrich sein. Außerdem dürfen Zuweisungen stattfinden. Eine Zuweisung auf eine normale Variable hat die Form *Bezeichner = Ausdruck*, wenn auf ein Array zugewiesen werden soll, muss die folgende Form vorliegen: *Arrayname Index # = Ausdruck*. Der Index ist dabei selbst auch wieder ein Ausdruck. Ein Ausdruck ist ein Funktionsaufruf oder die Anwendung eines Operators auf einen Ausdruck oder ein Bezeichner bzw. ein Literal (ein fester Wert, z.B. 1 oder „Hallo Welt“) und darf nicht nur in Zuweisungen, sondern auch als einzelne Anweisung vorkommen. Ein Funktionsaufruf hat die Form *Parameter1 Parameter2 Funktionsname*. Es müssen genau so viele Parameter vor dem Funktionsnamen stehen, wie in der Funktionsdefinition festgelegt wurden, diese Parameter werden in der Reihenfolge auf die Formalparameter (Parameter in der Funktionsdeklaration) angewendet, in der sie im Quelltext stehen. Jede Funktion wird genauso wie die *main*-Funktion definiert. Ein Operator wird angewendet, indem alle Operanden in der richtigen Reihenfolge notiert werden und danach der Operator steht. Abbildung 1 (Seite 11) gibt einen Überblick über alle Operatoren.

Außerdem gibt es noch die Möglichkeit, Schleifen und *if*-Anweisungen zu verwenden. Eine Schleife hat die Syntax

```

while Bedingung
  Block
```

Die Bedingung ist ein Ausdruck, der einen Wert vom Typ *bool* zurückgibt. Ein Block ist eine von geschweiften Klammern, die jeweils in separaten Zeilen stehen, umgebene Folge von Anweisungen oder eine einzelne Anweisung. Der Block wird solange ausgeführt, wie die Bedingung wahr ist. Eine *if*-Anweisung hat die Syntax

```

if Bedingung
  if-Block
else
```

else-Block

Der if-Block wird nur dann ausgeführt, wenn die Bedingung wahr ist. Ansonsten wird der else-Block ausgeführt. Der else-Block und das Schlüsselwort `else` können auch weggelassen werden, dann wird, falls die Bedingung nicht wahr ist keine Anweisung ausgeführt, sondern das Skript weiter ausgeführt.

Operator	Stellen	Funktion
Arrayoperatoren		
getLength	a	liefert die Länge des gegebenen Arrays
setLength	a,i	setzt die Länge des geg. Arrays auf den geg. Wert
swap	a,i,i	tauscht aus einem Array zwei Werte mit den geg. Indizes aus
toArray		wandelt alle Elemente des Stacks in ein einziges Array
Ausgabe		
print	?	schreibt das Argument in den Ausgabestrom
println	?	wie print, jedoch zusätzlich mit Zeilenumbruch
printStack		schreibt den Stapel in den Ausgabestrom
Arithmetik		
+	?,?	addiert beide Argumente, bei Arrays elementweise
-	(i;f;a),(i;f;a)	subtrahiert beide Argumente, bei Arrays elementweise
*	(i;f;a),(i;f;a)	multipliziert beide Argumente, bei Arrays elementweise
/	(i;f;a),(i;f;a)	dividiert beide Argumente, bei Arrays elementweise
^	(i;f;a),(i;f;a)	potenziert beide Argumente, bei Arrays elementweise
sum	?	kalkuliert die Arraysumme (bei bool die Bitsumme)
inc	(i;a)	inkrementiert das Argument, bei Arrays elementweise
dec	(i;a)	inkrementiert das Argument, bei Arrays elementweise
Boolesche Operatoren		
and	b,b	Und-Verknüpfung
or	b,b	Oder-Verknüpfung
!	(b;a)	Negation
neg	(b;a)	Negation
Vergleichsoperatoren		
<	n,n	kleiner als
>	n,n	größer als
<=	n,n	kleiner als oder gleich
>=	n,n	größer als oder gleich
==	n,n	gleich
!=	n,n	ungleich
Konvertierungsoperatoren		
toInt	(f;a)	konvertiert ein float in ein int, bei Arrays elementweise
toFloat	(i;a)	konvertiert ein int in ein float, bei Arrays elementweise
Zufallszahlen		
random		float-Zufallszahl aus dem Wertebereich [0, 1]
randomInt	i,i	int-Zufallszahl aus dem gegebenen Bereich
randomShuffle	a	mischt das gegebene Array
sonstige Stapeloperatoren		
inv		kehrt den Stapel um: das oberste Element zu unterst etc.
top		legt das aktuelle Stapelement nochmals auf den Stapel
pop		entkellert das aktuelle Stapelement

Zeichenerklärung:

i... int, f... float, s... string, b... bool

(i;f)... Angabe verschiedener Möglichkeiten

a... Array (Datentyp erschließt sich aus den anderen Angaben)

?... beliebiger Datentyp (jedoch pro Operation immer derselbe)

i,i... zwei Stellen (bei Angabe versch. Möglichkeiten jeweils derselbe)

n... (i;f;s;b)

Abbildung 1: Übersicht über die Operatoren