Sortierverfahren

Ein Leitprogramm von Susanne Kasper und Barbara Keller

Inhalt und Ziel:
Die Schülerinnen und Schüler lernen fünf verschiede Sortierverfahren der Informatik kennen und anwenden.

Unterrichtsmethode:
Das Leitprogramm ist ein Selbststudienmaterial. Es enthält alle notwendigen Unterrichtsinhalte, Übungen, Arbeitsanleitungen und Tests, die die Schüler/innen brauchen, um ohne Lehrperson lernen zu können.

Fachliches Review:
Juraj Hromkovic, Departement für Informatik, ETH Zürich

Fachdidaktisches Review:
Juraj Hromkovic, Departement für Informatik, ETH Zürich

Redaktion:
Regina Leufgen

Publiziert auf EducETH:
6. November 2006

Eidgenössische
Technische Hochschule
Zürich
Institut für Verhaltensforschung

Departement für Informatik

Leitprogramm
Sortierverfahren

Fach
Informatik

Stufe
Gymnasium (ca. 15-Jährige)

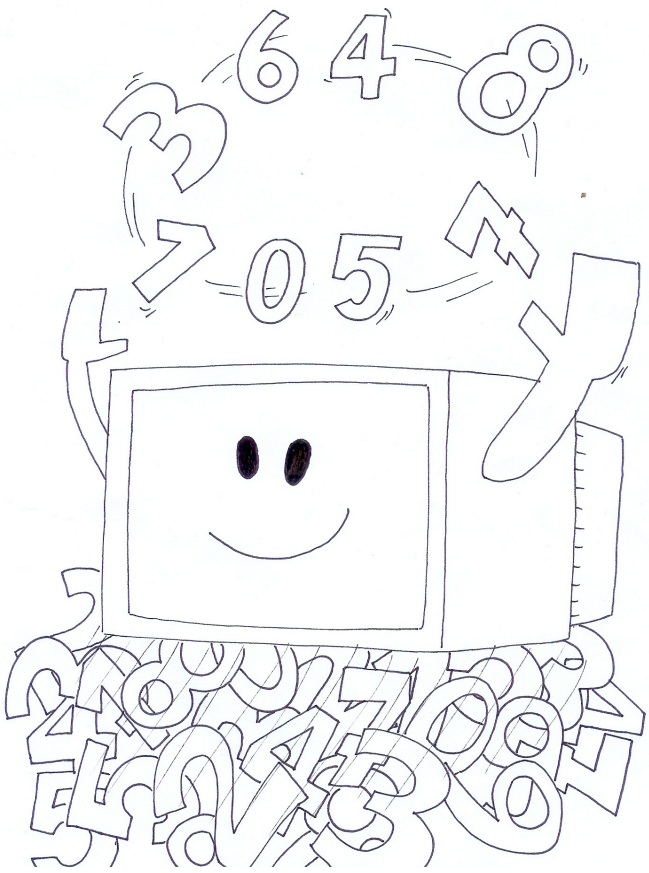
Bearbeitungsdauer
ca. 8 Lektionen

Autoren
Susanne Kasper
Barbara Keller

Betreuer
Prof. Juraj Hromkovič

Fassung
vom 06.11.2006

Kapitel
1 und 3: Susanne Kasper
2 und 4: Barbara Keller
Übrige: Kasper/ Keller



Illustrationen
Roman Haefeli

Schulerprobung
noch keine

Einleitung

Um was geht es?
Im Alltag gibt es viele Dinge, die sortiert werden. In der Schule werden Arbeitsblätter nach Fächern geordnet, zu Hause werden die CDs sortiert, damit man die Lieblingsmusik schneller findet.

Angenommen, du suchst eine Telefonnummer eines Bekannten und alle Nummern wären in einer beliebigen Reihenfolge aufgelistet. Bei einer grossen Stadt wie Zürich könnte es Tage dauern, bis man alle Nummern durchgesehen und die richtige gefunden hätte. Telefonbücher machen die Einträge übersichtlich, denn sie sind dort nach Namen geordnet. Man kann so einen bestimmten Eintrag viel einfacher und schneller finden.

Weil das Sortieren wichtig, aber auch eintönig und mühsam ist, gibt man diese Arbeit gerne dem Computer. Da weltweit ca. ein Viertel aller Rechenzeit für das Sortieren von Daten verwendet wird, lohnt sich der Blick auf die verschiedenen Verfahren und ihre Eigenschaften.

Wie kann ein Computer sortieren? In diesem Leitprogramm werden fünf Sortierverfahren vorgestellt. Darunter Methoden mit witzigen Namen wie Bubble-Sort, aber auch sehr ausgetüftelte Strategien. Am Anfang betrachten wir einfache, langsame Verfahren, später werden sie schneller und komplexer.

Tauch ein in die Welt des Sortierens: Viel Spass dabei!

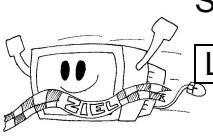
Inhaltsverzeichnis

Arbeitsanleitung

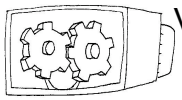
Wie gehst du mit diesem Leitprogramm am besten um?

In den nächsten Lektionen wirst du weitgehend selbstständig lernen. Alles was du zum Schluss wissen musst, kannst du alleine und in deinem eigenen Tempo mit diesem Leitprogramm erarbeiten.

Zur besseren Übersicht ist jedes Kapitel gleich aufgebaut. Die folgenden Symbole zeigen, was dich als Nächstes erwartet.

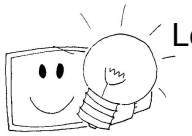


Lernziele — Sie beschreiben, was du nach dem Bearbeiten des Kapitels kannst.



Verfahren — Hier wird das konkrete Sortierverfahren Schritt für Schritt erklärt.

Lernkontrolle — Die Aufgaben der Lernkontrolle sind ein Selbsttest. Du alleine kontrollierst die Ergebnisse.



Lösungen — Am Ende des Kapitels stehen die Lösungen der Lernkontrolle. Vergleiche sie mit deinen Antworten.

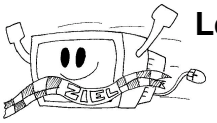
Kapiteltest beim Lehrer — Wenn du dich sicher fühlst und dir die Lernkontrolle keine Mühe bereitet, kannst du zum Kapiteltest antreten.

Falls du an einem Punkt im Leitprogramm noch Unsicherheiten oder Unklarheiten hast, dann geh zu dem Punkt zurück, an dem du noch alles verstanden hast. Beginne nochmals von dort. Manchmal gibt es noch hilfreiche Quellen, die du in diesem Fall bearbeiten kannst. Zusätzliches Material in Form von Power-Point Präsentationen gehören ebenfalls zum Leitprogramm. Sie sind entweder auf der beiliegenden CD oder als Download verfügbar. Frage deinen Lehrer, wo du sie abrufen kannst.

Einige Grundbegriffe sind im Kapitel 6 erklärt. Ein Begriff, der dort erklärt wird ist ***fett und kursiv*** geschrieben. Man kann diese Begriffe jederzeit nochmals nachschlagen.

Die Kapitel 1 bis 4 sind obligatorisch, müssen also bearbeitet werden. Das Kapitel 5 ein so genanntes Additum. Du kannst es freiwillig bearbeiten, wenn dir noch Zeit bleibt.

1. Kapitel: Bubble-Sort



Lernziele:

- Du kannst das Sortieren umgangssprachlich beschreiben.
- Du kannst die Grundlagen eines Sortierverfahrens aufzählen.
- Ein erstes, einfaches Verfahren kannst du anhand einer Zahlenfolge durchspielen und erklären.

1.1 Einführung

Um überhaupt sortieren zu können, braucht man einige Daten oder Dinge, die sortieren werden sollen. Dieses Leitprogramm benutzt dazu der Einfachheit halber Zahlenreihen mit kleinen, übersichtlichen Zahlen, z. B.:

5 3 4 2 1 6

Welche Schritte muss man machen, um die obere, unsortierte Reihe in folgende Form zu bringen? Überlege kurz.

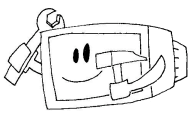
1 2 3 4 5 6

Lernkontrolle:

Aufgabe 1.1 a)
Welche Schritte hast du gemacht, um (5, 3, 4, 2, 1, 6) zu sortieren?

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Du hast die Folge (5, 3, 4, 2, 1, 6) nun intuitiv sortiert. Das ist gut so, aber man möchte vielleicht nicht nur eine bestimmte Folge, sondern auch eine beliebige Zahlenfolge sortieren. Dazu braucht man ein Verfahren oder Schema.



Lernkontrolle:

Aufgabe 1.1 b)
Verallgemeinere die Lösung der vorherigen Aufgabe so, dass man damit eine beliebige Zahlenfolge mit sechs Elementen sortieren kann.

.....
.....
.....
.....
.....

Du hast jetzt wahrscheinlich ein Verfahren beschrieben, welches in diesem Leitprogramm besprochen wird. Auf welche grundsätzlichen Probleme kann man beim Sortieren stossen? Besprich die Lösungen von Aufgabe 1.1 a) und 1.1 b) kurz mit deinem Banknachbarn/deiner Banknachbarin.

1.2 Wichtige Begriffe und Begriffserklärungen

Um die Aufgaben und Probleme genau zu beschreiben, wird nachfolgend zwischen diesen Aufgabenstellungen unterschieden:

- Sortiere eine bestimmte Reihe, die sechs Einträge hat.
- Sortiere irgendeine Reihe mit beliebig vielen Einträgen.

Der erste Fall „sortiere eine bestimmte, vorgegebene Reihe, die sechs Einträge hat“ entspricht der Aufgabe 1.1 a). Eine solche Aufgabe wird ***Problemfall*** genannt.

Die zweite Aufgabe „sortiere irgendeine Reihe mit beliebig vielen Einträgen“ wird nachfolgend ***Problem*** genannt und entspricht der Aufgabe 1.1 b).

Ein Problem besteht aus unendlich vielen Problemfällen.

Mögliche Problemfälle für das Problem „sortiere irgendeine Reihe mit beliebig vielen Einträgen“ sind z. B.: (5, 4, 2, 1, 6),
(4, 6, 8, 5, 7, 2, 9, 1),
(235, 7, 5278, 2, 11, 42),
(1, 2, 3, 4, 5, 6, 7, 8)
und viele mehr…

Lernkontrolle:

Aufgabe 1.2 a)
Erkläre die beiden neuen Begriffe „Problem“ und „Problemfall“.

Bei allen Verfahren müssen die Zahlen angeschaut, miteinander verglichen und verschoben werden. Weil man diese einfachen Tätigkeiten nicht weiter unterteilen kann, nennt man sie ***elementare Operationen***.

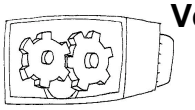
Man benötigt zum Sortieren folgende elementaren Operationen:

- **Vergleichen von zwei Zahlen**
  Eine Vergleichsoperation (grösser als „>“ oder kleiner als „<“) muss vorhanden sein. Damit kann man bestimmen, welche Zahl kleiner und welche grösser ist. Wenn man das weiss, kann man die Überlegungen fortsetzen. Um Spezialfälle zu vermeiden, werden hier keine identischen Zahlen in einer Zahlenfolge benutzt.

- **Das Verschieben von einzelnen Zahlen**
  Ist bekannt, welche Zahl von zwei betrachteten Zahlen grösser oder kleiner ist, kann man diese Zahl dem jeweiligen Verfahren entsprechend verschieben. Man kann sie auch an eine bestimmte neue Position schreiben, die nicht zwingend die endgültige Position sein muss.

1.3 Bubble-Sort

Ein erstes, einfaches ***Verfahren*** ist Bubble-Sort. Das bedeutet „Blubber-Sortierung“ oder „Sortierung durch Austauschen der Nachbarn“. Bei diesem Verfahren werden immer zwei Zahlen verglichen, die direkt nebeneinander stehen. Wenn du nicht weisst, was ein ***Verfahren*** ist, dann lies die Bedeutung in Kapitel 6 nach.



Verfahren:

1. Schritt: Betrachte die ersten beiden Zahlen.
2. Schritt: Vergleiche sie.
3. Schritt Falls sie in der richtigen Reihenfolge stehen, dann wird nichts unternommen. Falls sie in der falschen Reihenfolge stehen, werden die beiden Zahlen vertauscht.
4. Schritt Betrachte nun das nächste Zahlenpaar und wiederhole die Schritte 2 bis 4 bis zum letzten Zahlenpaar.

Wurden alle Schritte einmal auf die ganze Zahlenreihe angewendet, ist der erste ***Durchgang*** beendet. Ein Durchgang ist also abgeschlossen, wenn alle Zahlenpaare einmal betrachtet, verglichen und wenn nötig vertauscht wurden. Man siehst, dass die Zahlen schon ihrer Endposition näher gekommen sind. Die Folge ist aber noch nicht sortiert.
Die ***Schritte*** 1 bis 4 werden so lange wiederholt, bis in einem ganzen Durchgang keine Vertauschung mehr notwendig ist. Die Zahlenreihe ist dann sortiert.

Bubble-Sort am Beispiel:

Betrachte die folgende Zahlenreihe:

3 7 5 2

Zeichenerklärung:

[9] Zahlen mit einem Rahmen werden gerade bearbeitet.

 Dieser Doppelpfeil kennzeichnet eine Vertauschung der Zahlen.

1. Durchgang

[3]	[7]	5	2	Die ersten beiden ***Elemente*** werden betrachtet. Da sie in der richtigen Reihenfolge sind, bleiben sie an ihrer aktuellen Stelle.
3	[7] 	[5]	2	Die nächsten beiden Zahlen werden miteinander verglichen. Da sie nicht in der richtigen Reihenfolge stehen, werden sie vertauscht.
3	5	7	2	So sieht die neue Folge aus.
3	5	[7] 	[2]	Die letzten beiden Elemente werden verglichen. Auch sie müssen vertauscht werden.
3	5	2	7	So sieht die Folge nach dem ersten Durchgang aus.

2. Durchgang

[3]	[5]	2	7
3	[5] 	[2]	7
3	2	[5]	[7]

3. Durchgang

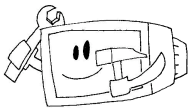
2 3 5 7

2 3 5 7 Nach dem dritten Durchgang ist die Folge sortiert.

Ein Computer merkt das aber nach diesem dritten Durchgang noch nicht. Für ihn ist eine Folge erst dann sortiert, wenn er in einem kompletten Durchgang keine Vertauschung mehr machen muss. Deswegen vergleicht er nochmals die benachbarten Elemente der ganzen Folge miteinander. Erst am Ende des vierten Durchgangs ist die Sortierung abgeschlossen, weil er im ganzen Durchgang keine Vertauschung mehr machen musste.

Zusätzliche Informationen:
Im Internet wird dieses Verfahren demonstriert. Du kannst es dir vorführen lassen, wenn du es noch nicht verstanden hast. Rufe dazu den untenstehenden Link auf. Drücke entweder auf den Knopf `go` (engl.: Start), dann wird das Verfahren komplett durchgeführt. Wählst du `step` (engl.: Schritt), dann wird nur ein Schritt ausgeführt. Du kannst die Demonstration auch `stop`pen und sie auf schrittweise oder auf `fast` (engl.: schnell) umstellen.

Link:
www.cs.pitt.edu/~kirk/cs1501/animations/Sort3.html



Lernkontrolle:

Aufgabe 1.3 a)
Führe Bubble-Sort für (3, 8, 5, 2, 7, 1) durch und schreibe mindestens die Reihen nach den einzelnen Durchgängen auf.

Aufgabe 1.3 b)
Wie viele Vergleiche muss man für die Zahlenfolge aus Aufgabe 1.3 a) machen? Überlege zuerst, wie viele Vergleiche für einen Durchgang benötigt werden und danach, wie viele Durchgänge das Verfahren braucht um die ganze Reihe zu sortieren?

Die Lösungen stehen am Ende des Kapitels.

1.4 Spezielle Eigenschaften von Bubble-Sort

Betrachtet man die Folgen nach den einzelnen Durchgängen, dann fällt auf, dass nach jedem Durchgang ein Element mehr an seiner endgültigen Position steht.

3	7	5	2	
3	5	2	**7**	nach dem 1. Durchgang
3	2	**5**	**7**	nach dem 2. Durchgang
2	**3**	**5**	**7**	nach dem 3. Durchgang

Die fettgedruckten Ziffern stehen an ihren endgültigen Positionen. Man kann also sagen, dass die grossen Zahlen wie Blasen (Bubbles) nach oben steigen. Deshalb wird dieses Verfahren „Bubble-Sort" genannt.

Lernkontrolle:

Aufgabe 1.4 a)
Wie viele Durchgänge muss man für eine beliebige Zahlenreihe machen?

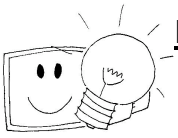
Aufgabe 1.4 b)
Welche Ausgangszahlenfolge würde bis zur sortierten Reihe am meisten Positionsvertauschungen benötigen? Überlege anhand der Folge (1, 2, 3, 4) und verallgemeinere dann.

Zusammenfassung

Du hast in diesem Kapitel Grundlegendes über Sortierverfahren und einige neue Begriffe kennengelernt. Um eine Zahlenfolge zu sortieren, muss man die Zahlen miteinander vergleichen und gegebenenfalls ihre Positionen verändern.

Im Bubble-Sort Verfahren wurden immer zwei benachbarte Zahlen betrachtet. Wenn die vordere Zahl grösser war als die Hintere, wurden die Positionen vertauscht. Dadurch bewegt sich im ersten Durchgang die grösste Zahl an den letzten Platz in der Zahlenreihe. Nach dem zweiten Durchgang stehen die grösste und die zweitgrösste Zahl an ihrem endgültigen Platz, usw.
Kommen in einem Durchgang keine Positionswechsel mehr vor, dann ist die Folge sortiert.

Wenn du alle Aufgaben lösen konntest, dann geh jetzt für den Kapiteltest zu deinem Lehrer/ deiner Lehrerin.



Lösungen zum Kapitel 1

Lösung 1.1 a) und 1.1 b)

Diese Lösungen sind individuell und es gibt dazu keine eindeutigen Antworten. Schaue dir den Rest des Leitprogramms an, dort findest du einige mögliche Lösungen.

Lösung 1.2 a)

Problemfall: Eine genau bestimmte Aufgabe, die man lösen muss.
Wenn die Reihe (3, 5, 2) gegeben ist, kann man einfach die 2 nach vorne schieben und man hat den Problemfall gelöst.

Problem: Die Gesamtheit der Problemfälle werden zu einem übergeordneten Problem zusammengefasst.
Will man z. B. alle Fälle lösen, in denen drei Elemente zu sortieren sind, reicht es nicht aus zu sagen: „Man nimmt die 2 nach vorne." Stattdessen müssen alle beliebigen Reihen mit drei Elementen mit einem Verfahren lösbar sein. Ein Problem ist also die Verallgemeinerung eines bestimmten Problemfalls auf alle denkbaren Fälle.
Um ein Sortierproblem mit drei Elementen zu lösen, könnte man sagen: „Suche das Minimum, schreibe es an die erste Stelle. Suche das Maximum und verschiebe es an die letzte Stelle." Mit diesem Verfahren werden alle Reihen mit drei Elementen sortiert. Dabei ist es unwichtig, in welcher Reihenfolge die Zahlen zu Beginn waren.

Lösung 1.3 a)

3	8	5	2	7	1	Ausgangsfolge
3	5	2	7	1	**8**	
3	2	5	1	**7**	**8**	
2	3	1	**5**	**7**	**8**	
2	1	**3**	**5**	**7**	**8**	
1	**2**	**3**	**5**	**7**	**8**	Ende des 5. Durchgangs

Man braucht fünf Durchgänge, bis die Folge sortiert ist. Der Computer merkt aber erst im sechsten Durchgang, dass er aufhören kann, weil er so lange vergleicht, bis keine Vertauschungen mehr nötig sind.

Lösung 1.3 b)

Es müssen 30 Vergleiche gemacht werden.

Es gibt sechs Zahlen in der zu sortierenden Folge. Da immer die benachbarten Zahlen verglichen werden, werden je Durchgang fünf Vergleiche gemacht. Weil man fünf Durchgänge braucht, bis die Folge sortiert ist und noch einen zur Bestätigung, braucht man sechs Durchgänge.

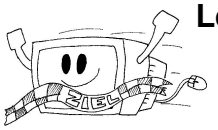
Es werden also 5 × 6 = **30 Vergleiche** angestellt.

Lösung 1.4 a)
Es müssen maximal so viele Durchgänge gemacht werden, wie es zu sortierende Zahlen gibt, weil nach jedem Durchgang eine Zahl mehr an ihrer endgültigen Position steht.

Strenggenommen braucht man einen Durchgang weniger. Denn wenn alle Zahlen bis auf eine an der richtigen Position sind, dann steht auch die Letzte richtig. Der Computer braucht jedoch die Extrarunde, um sicher zu sein, dass keine Vertauschungen mehr vorkommen.

Lösung 1.4 b)
Am meisten Vertauschungen benötigt die absteigend sortierte Folge, die aufsteigend sortiert werden soll.
Will man (4, 3, 2, 1) mit dem Bubble-Sort Verfahren in die Form (1, 2, 3, 4) bringen, benötigt man die meisten Vertauschungen.

2. Kapitel: Selection-Sort und Insertion-Sort



Lernziele:

- Du weisst, wie der Computer das Minimum (= kleinstes Element) und das Maximum (= grösstes Element) einer Zahlenreihe sucht.
- Du kannst das Verfahren „Selection-Sort“ in eigenen Worten erklären.
- Du kannst einen Stapel Karten mittels Insertion-Sort sortieren.

2.1 Minimumsuche

Die Suche nach dem Minimum in einer Zahlenreihe ist einfach. Meistens muss man sie nur ansehen und findet den kleinsten Wert schnell. Aber wie sucht ein Computer das kleinste Element?

Die Grundidee der Minimumsuche basiert auf dem Ko-System von Wettkämpfen: Zwei Gegner treten gegeneinander an – einer gewinnt, einer verliert. Der Gewinner bleibt im Wettkampf und tritt gegen den Nächsten an. Wer in der letzten Runde siegt, hat den kompletten Wettkampf gewonnen.

Der Computer setzt das Prinzip so um: Er beginnt mit der ersten Zahl in einer Zahlenfolge. Diese Zahl ist von den bisher betrachteten Zahlen die Kleinste, weil er noch keine andere analysiert hat. Nun betrachtet er zusätzlich eine zweite Zahl. Die Kleinere geht als Sieger hervor. Der Computer speichert diese Zahl und vergleicht sie mit der Nächsten. Wiederum gewinnt die Kleinere der beiden.

Diese Vergleiche werden so lange fortgesetzt, bis auch die letzte Zahl betrachtet und mit einer anderen aus der Zahlenreihe verglichen wurde. Die Zahl, die am Schluss übrig bleibt, ist das gesuchte Minimum der ganzen Zahlenreihe.

Lernkontrolle:

Aufgabe 2.1 a)
Suche das Minimum der folgenden Zahlenreihe mit dem vorgestellten Verfahren. Notiere die einzelnen Schritte.

3 7 5 2

Aufgabe 2.1 b)
Wie viele Vergleiche führt ein Computer in der Zahlenreihe aus der Aufgabe 2.1 a) durch, bis er das Minimum findet?

Aufgabe 2.1 c)
Muss ein Computer mehr oder weniger Vergleiche machen, wenn die Zahlenreihe zu Beginn so lautet?

2 3 7 5

Aufgabe 2.1 d)
Schreibe in wenigen Sätzen auf, wie ein Computer das Maximum einer Zahlenfolge bestimmen kann.

Aufgabe 2.1 e)
Wie viele Vergleiche benötigt ein Computer, wenn er das Maximum von fünf, sieben und 2000 Zahlen finden möchte? Formuliere eine allgemeingültige Aussage dazu.

2.2 Die Grundidee von Selection- und Insertion-Sort

In diesem Kapitel werden zwei Sortierverfahren vorgestellt, die eine Gemeinsamkeit haben: Beide gehen davon aus, dass der Anfang einer Zahlenfolge sortiert ist. In jedem Schritt wird anschliessend diese Anfangsfolge um ein Element erweitert.

Kann man voraussetzen, dass der Anfang schon sortiert ist?
Ja, man kann. Man muss ihn nur klein genug wählen. Wenn man eine Zahlenfolge mit Selection- und Insertion-Sort bearbeitet, wird die Folge in zwei Teilfolgen aufgeteilt. Eine Teilfolge ist leer, die andere enthält alle Elemente. Weil die leere Zahlenfolge keine Elemente enthält, kann sie nicht unsortiert sein und gilt daher als sortiert.

Zur Veranschaulichung eine Illustration:

Zeichenerklärung:

Die Quadrate stellen beliebige Zahlen einer Zahlenfolge dar.

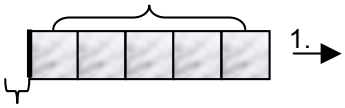

Ein Quadrat mit dieser Färbung gehört zur unsortierten Teilfolge.


Ein Quadrat mit dieser Färbung gehört zur sortierten Teilfolge.


Die Zahlen über den Pfeilen bezeichnen die Durchgänge der Verfahren.

Dieser Strich trennt die sortierte und die unsortierten Teilfolge.

Hier sieht man die unsortierte Teilfolge. Vor dem ersten Durchgang gehören alle Elemente noch zur unsortierten Teilfolge.

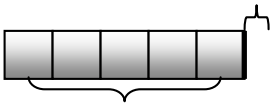


Die als sortiert geltende, leere Teilfolge wird eingefügt.

Nach dem ersten Durchgang bekommt die sortierte Teilfolge ein Element hinzu. Die unsortierte Teilfolge ist damit um ein Element kleiner als zu Beginn.

Nach jedem weiteren Durchgang ist die sortierte Teilfolge um ein Element grösser und die unsortierte Teilfolge um ein Element kleiner.

Nach dem fünften Durchgang ist die unsortierte Teilfolge leer,…



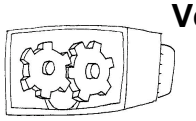
…und alle Elemente sind in der sortierten Teilfolge.

Somit ist die ganze Folge sortiert. Nach jedem Durchgang gibt es ein Element mehr in der sortierten Teilfolge. Folglich braucht man so viele Durchgänge, wie es Elemente in der Zahlenfolge gibt.

Das dargestellte Prinzip wird sowohl von Selection- als auch von Insertion-Sort verwendet. Sie unterscheiden sich jedoch in der Umsetzung.

2.3 Selection-Sort

„Selection“ bedeutet auf Englisch „Auswahl“. Selection-Sort ist also das Sortieren durch Auswahl. Aus einer Zahlenfolge wählt man das kleinste Element des unsortierten Teils aus und fügt es an der richtigen Position ein. Es funktioniert so, wie die meisten Menschen intuitiv sortieren.



Verfahren:

1. Schritt Suche das Minimum.
2. Schritt Vertausche das Minimum mit dem Element an der ersten Position der Zahlenfolge.
3. Schritt Suche das nächstkleinste Element.
4. Schritt Tausche es gegen das Element aus, welches an seiner Position in der Sortierung steht.

Schritt 3 und 4 werden so lange wiederholt, bis alle Elemente betrachtet wurden und damit die Zahlenreihe sortiert ist. In Zukunft werden hier diese beiden Schritte zusammengefasst und als ein Durchgang bezeichnet.

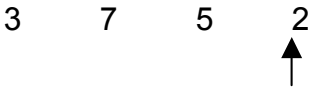
Selection-Sort am Beispiel:

Zeichenerklärung:

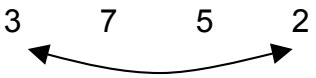
Die fett gedruckten Zahlen sind bereits sortiert.

Mit dem Pfeil markiert der Computer das Minimum der unsortierten Zahlenfolge analog des beschriebenen Verfahrens für die Minimumsuche.

1. Durchgang



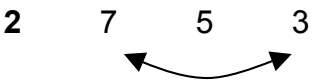
Der Computer sucht das Minimum und markiert es.



Dann vertauscht er das Minimum mit dem Element an der ersten Position.

2. Durchgang

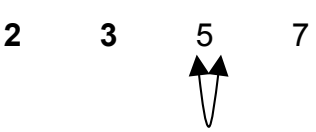
Nun sucht er das zweitkleinste Element.



Das Minimum der unsortierten Folge wird direkt hinter das letzte Element der sortierten Folge positioniert.

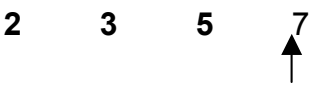
3. Durchgang

Das nächste Minimum wird gesucht.

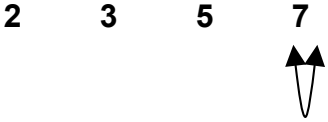


Da die 5 schon am richtigen Platz ist, wird sie mit sich selbst vertauscht.

4. Durchgang



Das nächste Minimum wird gesucht.

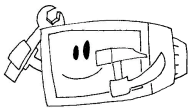


Die 7 ist auch am richtigen Platz und wird ebenfalls mit sich selbst vertauscht.

Fertig!

Zusätzliche Informationen:
Falls du dich bei diesem Thema noch nicht sicher fühlst, ist unter folgendem Link ein anschauliches Beispiel dargestellt. Lies dazu den Abschnitt „Das Sortierverfahren Selection-Sort“ auf der ersten Seite und betrachte das Beispiel auf der gegenüberliegenden Seite. Das Wort „Array“ ist in diesem Zusammenhang gleichzusetzen mit dem Begriff „Zahlenfolge“.

Link:
www.educeth.ch/lehrpersonen/informatik/unterrichtsmaterialien_inf/programmieren/array/uebung_computer.pdf



Lernkontrolle:

Aufgabe 2.3 a)
In einer Variante von Selection-Sort wird nicht das Minimum, sondern das Maximum ausgewählt. Trotzdem soll das kleinste Element am Schluss links stehen. Wie funktioniert das? Spiele dieses Verfahren anhand von folgendem Beispiel durch:

3 7 5 2

Aufgabe 2.3 b)
Wie viele Vergleiche benötigt Selection-Sort für das Beispiel in 2.3.a) im ersten Durchgang?
Überlege auch, wie viele Vergleiche man im zweiten und dritten Durchgang benötigt. Findest du eine Regelmässigkeit?

Aufgabe 2.3 c)
Wie viele Vergleiche benötigt Selection-Sort mit 23 Karten im fünften Durchgang?

Aufgabe 2.3 d)
Es werden nicht nur Elemente verglichen, sondern auch vertauscht. Wie viele Vertauschungen finden im ersten, zweiten und fünften Durchgang statt?

2.4 Insertion-Sort

„Insertion“ ist Englisch und bedeutet „Einfügen“. Insertion-Sort sortiert also durch Einfügen. Bevor das Verfahren vorgestellt wird: Überlege, wie man in einem Kartenspiel soeben aufgenommene Karten sortiert.

Öffne danach den untenstehenden Link.
Wenn man auf `Start` klickt, beginnt die Demonstration.
Die verwendeten Farben bedeuten Folgendes:
Der unsortierte Teil der Zahlenreihe ist rot dargestellt.
Der sortierte Teil der Zahlenreihe ist dunkelblau dargestellt.
Die Elemente, die gerade betrachtet werden, sind hellblau.

Klicke anschliessend auf `Continue` (engl.: weiter).
Ab jetzt wird das Verfahren Schritt für Schritt vorgeführt. Klicke nach jedem Schritt auf `Next`. Um nur das Ergebnis ohne die einzelnen Durchgänge anzusehen, drücke auf `Finish` (engl.: beenden).

Link:
www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/insertionSort/insertionSort.html

Schreibe in wenigen Sätzen auf, wie Insertion-Sort funktioniert. Besprich deine Lösung mit jemandem, der diese Aufgabe schon bearbeitet hat.

.....
.....
.....
.....
.....
.....
.....
.....
.....

In der Einführung von Insertion-Sort im Kapitel 2.2 wurde erläutert, dass der Computer zuerst die leere Menge betrachtet, da diese bereits sortiert ist. Denkt man einen Schritt weiter, so ist auch eine einzelne Zahl bereits sortiert. In den folgenden Beispielen wird daher nicht mit der leeren Menge, sondern direkt mit der ersten Zahl begonnen. Man spart also den ersten Schritt aus.

Für den Fall offener Fragen zu diesem Verfahren befindet sich auf der CD die PowerPoint Demonstration „Insertion Sort“. Frage deinen Lehrer danach.
Alternativ steht folgendes Beispiel zur Verfügung:

Insertion-Sort am Beispiel:

Zeichenerklärung:

9	2			Die fett gedruckten Zahlen sind bereits sortiert.

1. Durchgang

3	7	5	2	Der Computer betrachtet die Zahl 3. Da sie die erste Zahl ist, gilt sie bereits als sortiert. Jedoch befindet sie sich noch nicht an der endgültigen Position.

2. Durchgang

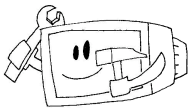
3	7	5	2	Nun betrachtet er die 7. Er vergleicht sie mit der Zahl direkt vor ihr. Da 7 grösser als 3 ist, sind diese Zahlen schon in der richtigen Reihenfolge. Der Computer muss sie nicht vertauschen.

3. Durchgang

3	**7**	5	2	Der Computer betrachtet die 5 und vergleicht sie mit der Zahl direkt vor ihr. Weil die 5 kleiner ist, muss sie mit der 7 getauscht werden.
3	5	**7**	2	5 wird mit 3 verglichen. Da die 5 grösser ist, hat sie ihren vorläufigen Platz gefunden.

4. Durchgang

3	**5**	**7**	2	Nun wird 2 mit 7 verglichen. Da 2 kleiner ist als 7, werden diese beiden Zahlen vertauscht.
3	**5**	2	**7**	Die 2 wird wieder mit der Zahl direkt vor ihr verglichen. Weil die 2 kleiner ist, werden die Plätze getauscht.
3	2	**5**	**7**	Dann wird 2 mit 3 verglichen. Wieder ist die 2 kleiner; es wird also erneut getauscht.
2	**3**	**5**	**7**	Weil links von der 2 keine Zahlen mehr sind, steht sie jetzt an ihrer endgültigen Position. Zudem war sie die letzte unsortierte Zahl. Die Sortierung ist beendet.



Lernkontrolle:

Aufgabe 2.4 a)
Wieso wird das Verfahren Insertion-Sort genannt?

Aufgabe 2.4 b)
Wie sieht diese Sequenz nach fünf Durchgängen von Insertion-Sort aus?

45 21 3 8 47 59 35 65

Aufgabe 2.4 c)
Wie viele Vergleiche benötigt man im zweiten, dritten und achten Durchgang mindestens? Wie muss die Zahlenfolge lauten, damit man möglichst wenige Vergleiche benötigt?

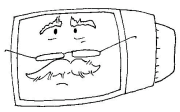
Aufgabe 2.4 d)
Wie oft muss man eine beliebige Zahl höchstens verschieben, bis sie an der vorläufig richtigen Position im zweiten, dritten respektive 42. Durchgang steht?
Wie sieht die Zahlenfolge hierfür aus? (Überlegen, nicht nachzählen!)

Zusammenfassung

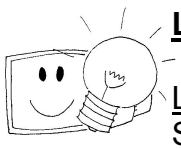
Um in einer Zahlenreihe ein Minimum zu finden, analysiert der Computer sie schrittweise. Er speichert das vorläufige Minimum und vergleicht es mit der nächsten Zahl. Ist diese kleiner, wird sie das neue Minimum. Anderenfalls bleibt die bisherige Zahl das Minimum.

Selection-Sort: Das kleinste Element wird gesucht und an die erste Position der Zahlenfolge gesetzt. Danach wird das zweitkleinste Element an der zweiten Stelle positioniert. Das Verfahren dauert an, bis die Zahlenfolge sortiert ist.

Insertion-Sort: Die nächste Zahl der unsortierten Zahlenfolge wird in die bereits sortierte Zahlenfolge eingefügt. Es ist vergleichbar mit der Aktion beim Kartenspielen, wenn man eine neue Karte vom Stapel nimmt und in vorhandene Karten einfügt.



Nun kannst du dich für den Kapiteltest melden.



Lösungen zum Kapitel 2

Lösung 2.1 a)
Suche das Minimum in folgender Zahlenreihe:

3 7 5 2

Zeichenerklärung:

6
↑ Der Pfeil kennzeichnet das bisherige Minimum. Es entspricht dem Element, das sich der Computer gemerkt hat. Übrige Zeichen analog zum letzten Kapitel.

Minimumsuche anhand der Aufgabe:

3 7 5 2
↑ Der Computer merkt sich die 3 und vergleicht sie mit der 7. Weil 3 kleiner ist als 7, ist das vorläufige Minimum weiterhin die 3.

3 7 5 2
↑ Das bisher kleinste Element wird mit dem nächsten Element verglichen. Da 3 kleiner ist als 5, merkt sich der Computer wieder die 3.

3 7 5 2
↑ Jetzt vergleicht der Computer die 3 mit der 2.
Weil 2 kleiner ist als 3, merkt er sich neu die 2.

3 7 5 2
↑ Die Zahlenreihe wurde komplett betrachtet.
2 ist das endgültiges Minimum.

Lösung 2.1 b)
Der Computer benötigt drei Vergleiche (siehe Antwort 2.1a).

Lösung 2.1 c)
Er braucht genau gleich viele Vergleiche, denn der Computer erkennt am Anfang nicht, dass er bereits das kleinste Element betrachtet.

Lösung 2.1 d)
Um das Maximum zu finden, wendet ein Computer das gleiche Verfahren an wie bei der Minimumsuche. Er speichert jedoch immer die Grössere von zwei Zahlen.

Lösung 2.1 e)
Ein Computer braucht immer einen Vergleich weniger als die Anzahl der Elemente in der Zahlenfolge – wie bei der Minimumsuche auch.
Mit fünf Elementen braucht er vier, bei sieben also sechs und bei 2000 demnach 1999 Vergleiche.

Lösung 2.3 a)
Selection-Sort mit Maximumsuche entspricht Selection-Sort mit Minimumsuche, wie bisher beschrieben. Unterschied: Man sortiert von hinten nach vorne.

Verfahren anhand der Aufgabe:

3 7 5 2	Der Computer sucht das Maximum und markiert es.
	
3 7 5 2	Er tauscht das Maximum gegen das Element an der letzten Stelle aus.
	
3 2 5 **7**	Nun sucht er das zweitgrösste Element, also das Maximum der unsortierten Zahlenreihe.
3 2 5 **7**	Weil die 5 schon am richtigen Platz ist, wird sie mit sich selbst vertauscht.
	
3 2 **5** **7**	Das nächste Maximum wird gesucht.
	
3 2 **5** **7**	Das Maximum der unsortierten Folge bekommt den Platz direkt vor dem ersten Element der sortierten Folge.
	
2 **3** **5** **7**	Nun wird nochmals das Maximum gesucht.
	
2 **3** **5** **7**	Die 2 ist am richtigen Platz und wird mit sich selbst getauscht.
	
2 **3** **5** **7**	Fertig!

Ein anderes Beispiel wird unter folgendem Link demonstriert. Du kennst die Seite schon von Bubble-Sort. Bedienhinweise zum Nachschlagen am Ende von Kapitel 1.3. Wähle diesmal „Selection Sort“ aus.

Link:
www.cs.pitt.edu/~kirk/cs1501/animations/Sort3.html

Lösung 2.3 b)
Das Verfahren braucht im ersten Durchgang drei Vergleiche. Analog der Minimumsuche werden die „Anzahl der Zahlen minus 1“, benötigt, hier also
4 - 1 = 3 Vergleiche.
Im zweiten Durchgang braucht man zwei Vergleiche. Da die vorderste Zahl schon endgültig sortiert ist, muss man das Minimum nur noch im restlichen, unsortierten Teilstück suchen.
Um das Minimum von drei Zahlen zu finden, werden genau zwei Vergleiche benötigt. Nach jedem Schritt wird eine Zahl weniger betrachtet, daher braucht man in jedem weiteren Durchgang einen Vergleich weniger.

Lösung 2.3 c)
Im ersten Durchgang mit 23 Karten benötigt man 22 Vergleiche. In jedem Durchgang braucht man einen Vergleich weniger. Im fünften Durchgang braucht man also 18 Vergleiche.

Lösung 2.3 d)
Das gefundene Minimum wird in jedem Durchgang mit der Zahl vertauscht, die an dessen Platz steht. In jedem Durchgang findet genau ein Tausch statt, unabhängig davon, ob es sich um den ersten, zweiten oder fünften Durchgang handelt.

Lösung 2.4 a)
Das Verfahren wird Insertion-Sort genannt, weil ein Element nach dem anderen in den bereits sortierten Teil eingefügt wird. (engl. to insert: einfügen)

Lösung 2.4 b)

nach 1. Durchgang:	**45**	21	3	8	47	59	35	65
nach 2. Durchgang:	**21**	**45**	3	8	47	59	35	65
nach 3. Durchgang:	**3**	**21**	**45**	8	47	59	35	65
nach 4. Durchgang:	**3**	**8**	**21**	**45**	47	59	35	65
nach 5. Durchgang:	**3**	**8**	**21**	**45**	**47**	59	35	65

Man findet die Lösung auch nur durch Überlegen. Nach dem fünften Durchgang sind die ersten fünf Zahlen in der richtigen Reihenfolge und die restlichen Zahlen noch unsortiert.

Lösung 2.4 c)
Im zweiten Durchgang benötigt man mindestens einen Vergleich. Gleiches gilt für den dritten und achten Durchgang. Das gilt immer dann, wenn die einzufügende Zahl die Grösste der bisher sortierten Zahlen ist.

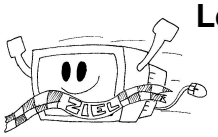
Ein Beispiel dafür ist in Aufgabe 2.4 b) im fünften Durchgang zu finden. 47 ist grösser als 45 und somit grösser als alle bisher sortierten Zahlen.

Weil der Computer konsequent das Verfahren anwendet, vergleicht er die 47 zuerst noch mit der 45. Wenn die Zahlenfolge schon sortiert ist, benötigt man die wenigsten Vergleiche.

Lösung 2.4 d)
Falls eine Zahl die bisher Kleinste ist, muss man jede andere bereits sortierte Zahl um einen Platz weiter nach rechts verschieben, bis die kleinste Zahl am Anfang der Zahlenfolge steht. Man vertauscht sie also mit allen bereits sortierten Zahlen.

Im zweiten Durchgang ist es eine Zahl, die verschoben werden muss. Im dritten Durchgang sind es bereits zwei Zahlen und im 42. Durchgang sind es 41 Zahlen. Ein Beispiel dafür befindet sich in Aufgabe 2.3 b) im dritten Durchgang.
Ist die Zahlenfolge absteigend sortiert, z. B. (4 3 2 1), werden die meisten Vertauschungen benötigt.

3. Kapitel: Merge-Sort



Lernziele:

- Du kannst das Konzept von Merge-Sort in eigenen Worten erklären.
- Du kannst eine Zahlenreihe mit dem neuen Verfahren sortieren.
- Du kennst die Vor- und Nachteile des neuen Verfahrens.

3.1 Einführung

Angenommen man muss ein sehr grosses und schwieriges Problem lösen. Wie geht man vor? Es gibt die Möglichkeit, einige Lösungsansätze einfach auszuprobieren oder das Problem in einfache Teilprobleme aufzuteilen. Falls du die zweite Variante wählst, machst du automatisch das, was man unter dem Konzept Teile-und-Herrsche versteht: Man teilt das Ursprungsproblem in kleinere Probleme auf, die man schnell und einfach lösen kann. Anschliessend kombiniert man die Teillösungen wieder zu einem Ganzen.

Die kleinsten Teile eines grossen Problems nennt man elementare Probleme. Ein ***elementares Problem*** ist so klein, dass man es nicht mehr weiter zerlegen kann. Es ist schnell und einfach zu lösen.

Beispiel 1:
Wie viel ergibt **163 × 4852**?
Nimm an, es ist kein Taschenrechner verfügbar und du musst die Aufgabe schriftlich lösen, wie machst du das?

Zu Beginn legt man ein elementares Problem fest. In vorliegenden Fall ist eine Multiplikation von einer Ziffer der ersten Zahl mit einer Ziffer der zweiten Zahl ein elementares Problem. Die zweite Zahl kann keine, eine oder mehrere anschliessende Nullen haben.

Elementare Probleme sind zum Beispiel: 2 × 5, 7 × 90 oder 3 × 4000.

Man schreibt die beiden Zahlen 163 × 4852 auf und multipliziert von der ersten Zahl eine Ziffer nach der anderen mit der zweiten Zahl.

Man teilt den Problemfall 163 × 4852 somit in drei Teilprobleme auf:
3 × 4852, 60 × 4852 und 100 × 4852. Die drei Teilprobleme löst man nach dem gleichen Verfahren wie das Ursprungsproblem.

Lösung eines Teilproblems:
Man erkennt im ersten Teilproblem 3 × 4852, dass es auf die gleiche Weise gelöst werden kann wie das Ursprungsproblem.
Man zerlegt die zweite Zahl in ihre Ziffern und erhält dadurch die elementaren Probleme 3 × 2, 3 × 50, 3 × 800 und 3 × 4000.

Diese einfach zu lösenden elementaren Probleme werden nacheinander berechnet. Danach werden die Zwischenergebnisse addiert. Das Ergebnis ist die erste Teillösung des grossen Problems.

```
3 × 4852
       6      3 ×    2
 +   15.      3 ×   50 (auf die Zehnerstelle achten!)
 +  24..      3 ×  800
 + 12...      3 × 4000
 = 14556      → Teillösung 1
```

Achtung auf die Zehnerstellen, denn man rechnet in der zweiten Zeile 3 × 50 und nicht 3 × 5! Die Punkte als Platzhalter verdeutlichen das.

Man löst die anderen beiden Teilprobleme analog und erhält drei Teillösungen. Sie müssen nun richtig miteinander kombiniert werden:

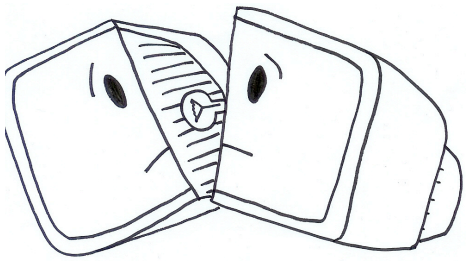
- Die Teillösung der Einerstelle (3 von 163) wird mit 1 gewichtet.
- Die Teillösung der Zehnerstelle (6 von 163) wird mit 10 gewichtet.
- Die Teillösung der Hunderterstelle (1 von 163) mit 100 gewichtet.

Danach addiert man die Ergebnisse und erhält die Gesamtlösung.

Die Gewichtung entspricht den Punkten im untenstehenden Beispiel.

1. Teilproblem	2. Teilproblem	3. Teilproblem	**Kombination**
16**3** × 4852	1**6**3 × 4852	**1**63 × 4852	163 × 4852
14556	14556	14556	14556
	29112.	29112.	**+** 29112.
		4852..	**+** 4852..
			= 790876
			Gesamtlösung

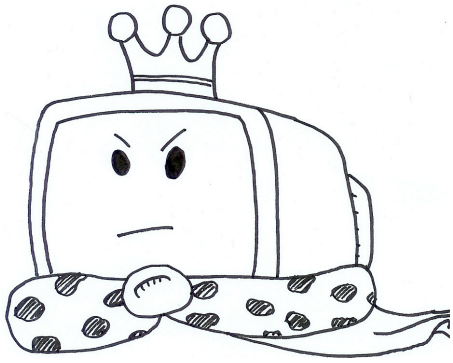
Das Problem wurde zuerst in Teilprobleme und anschliessend in elementare Probleme geteilt, die man nacheinander auf die gleiche Weise löst. Die Lösungen dieser elementaren Probleme wurden erst zu Teillösungen, dann zu einer Gesamtlösung kombiniert. Damit ist das Ursprungsproblem gelöst.



Dieses Konzept ist in der Informatik als

„Teile-und-Herrsche“

(engl.: *„Divide and Conquer“)*
bekannt.
Es ist die Grundlage von Merge-Sort.



Was genau heisst „Teile-und-Herrsche“?

Falls ein Problem für eine direkte Lösung zu umfangreich ist, dann:

- teile das Problem in mindestens zwei, ungefähr gleich grosse Teilprobleme (divide).
- löse die kleineren, einfacheren Teilprobleme (elementare Probleme) auf die gleiche Art (conquer).
- füge die Teillösungen zu einer Gesamtlösung zusammen (merge).

Lernkontrolle

Aufgabe 3.1 a)

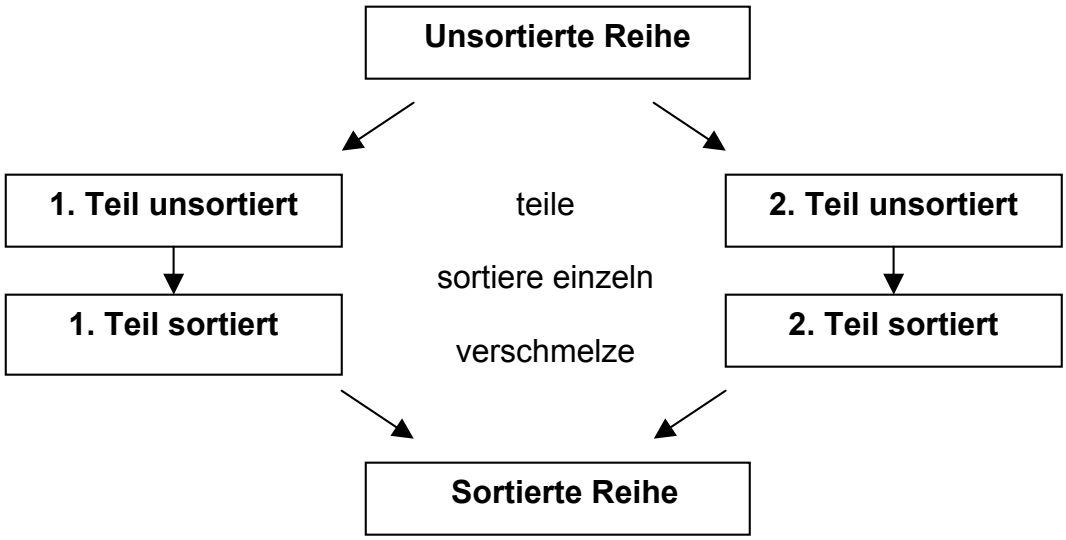
Finde ein Beispiel für ein Alltagsproblem, welches mit dem Teile-und-Herrsche-Prinzip gelöst werden kann. Verwende für die Suche maximal fünf Minuten. Findest du keins, schau dir die Beispiele in der Lösung an.

3.2 Allgemeines zu Merge-Sort

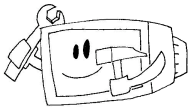
Hauptbestandteil von Merge-Sort ist das Verschmelzen von kleineren Teilreihen, deshalb wird es „Sortieren durch Zusammenfügen“ genannt. „Merge“ ist Englisch und bedeutet „vereinigen“, „zusammenfügen“ oder „verschmelzen“.

Man zerlegt die ursprüngliche Zahlenreihe in zwei etwa gleich grosse Teilreihen, die man einzeln sortiert. Wenn beide Teilreihen sortiert sind, fügt man sie im Reissverschlussverfahren zusammen und erhält eine sortierte Gesamtreihe.

Illustration zu Merge-Sort:



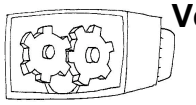
Quelle: http://de.wikipedia.org/wiki/Mergesort



Lernkontrolle:

Aufgabe 3.2 a)
Wie kann man am einfachsten zwei sortierte Zahlenreihen zu einer gemeinsamen sortierten Zahlenreihe verschmelzen?

Aufgabe 3.2 b)
Was könnte ein elementares Problem für ein Sortierverfahren sein, das auf Teile-und-Herrsche beruht?



Verfahren:

1. Schritt: Falls die betrachtete Reihe mehr als ein Element hat, teile sie in zwei Teilreihen auf.
2. Schritt: Sortiere nun die beiden Teilreihen getrennt. Wende das gleiche Verfahren an wie für die grosse Reihe. Beginne also für jede Teilreihe wieder bei Schritt 1. (Selbstaufruf des Verfahrens für beide Teile)
3. Schritt: Vereinige die zwei sortierten Teilreihen zur gemeinsamen Reihe. Vergleiche nacheinander die vorderen Elemente der beiden Reihen. Füge das Kleinere der beiden ersten Elemente in die gemeinsame Reihe ein. Mach es so lange, bis eine Teilreihe leer ist. Die restlichen Elemente der noch vollen Reihe fügt man an das Ende der gemeinsame Reihe an.

3.3 Selbstaufruf

Ein Teile-und-Herrsche-Verfahren hat mehrere Hierarchiestufen. Das ursprüngliche Problem wird in zwei Teilprobleme geteilt, die dann separat mit dem gleichen Verfahren gelöst werden. Man spaltet die Teile auf, bis sie so klein sind, dass einfach lösbare elementare Probleme vorliegen. Wegen dieser Praxis spricht man von einem selbstaufrufenden Verfahren, das auch ***rekursives Verfahren*** genannt wird.

Die Anzahl der Rekursionsstufen ist abhängig von der Grösse des ursprünglichen Problems. Je grösser das Anfangsproblem ist, desto mehr Rekursionsstufen gibt es.

Die Strategie des Selbstaufrufs kann zu sehr eleganten Lösungen für mathematische Probleme führen. Wenn ein Programm sich immer wieder selbst aufruft, braucht es eine gute Abbruchbedingung. Eine Abbruchbedingung definiert den Punkt, an dem ein elementares Problem besteht und nicht mehr weiter aufgeteilt wird. Das elementare Problem wird dann gelöst. Damit verhindert man, dass das Verfahren endlos läuft.

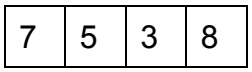
Das elementare Problem unterliegt folgenden Regeln:

- Wenn nur noch ein Element in der Reihe enthalten ist, dann versuche nicht weiter zu teilen. Ein Element alleine ist immer sortiert, daher ist auch Schritt 2 schon vollzogen.
- Da in Schritt 1 nichts geteilt wurde, müssen keine Teile zusammengefügt werden. Man darf also bei nur einem Element in einer Reihe das Verfahren abkürzen und alle drei Schritte überspringen. Man beginnt wieder bei Schritt 1, als das Verfahren rekursiv aufgerufen wurde.

Die Bedeutung von Rekursion ist im nächsten Beispiel nochmals veranschaulicht.

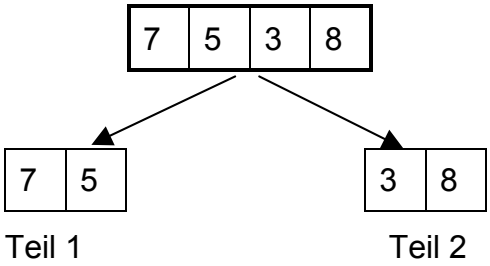
3.4 Beispiel zu Merge-Sort

Ausgangsreihe:



Schritt 1 für die gesamte Reihe:
Teile die Reihe in zwei gleich grosse Teilreihen auf. Hat man eine ungerade Anzahl von Elementen zu sortieren, ist die erste Teilreihe um ein Element grösser als die Zweite.

Schritt 1 für die Ausgangsreihe: Teilen



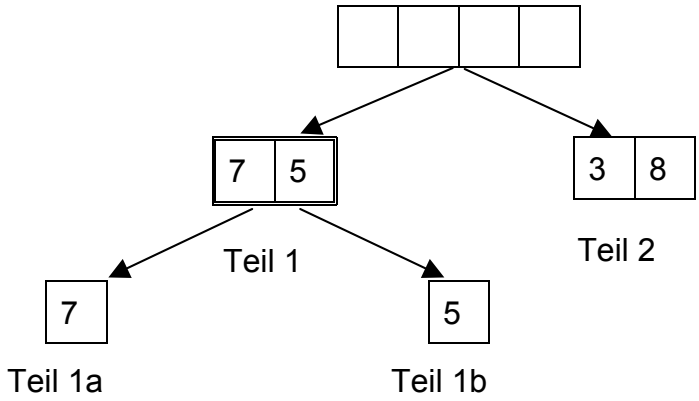
1. Rekursionsstufe

Schritt 2 für die Ausgangsreihe: Sortieren, jeden Teil einzeln

Sortiere nun Teil 1 und Teil 2 rekursiv, d.h. mit dem gleichen Verfahren.

Betrachtung von Teil 1:
Man sortiert Teil 1 mit dem gleichen Verfahren wie die Ausgangsreihe.
Schritt 1 wird nur für den Teil 1 ausgeführt.

Schritt 1 für Teil 1: Teile!

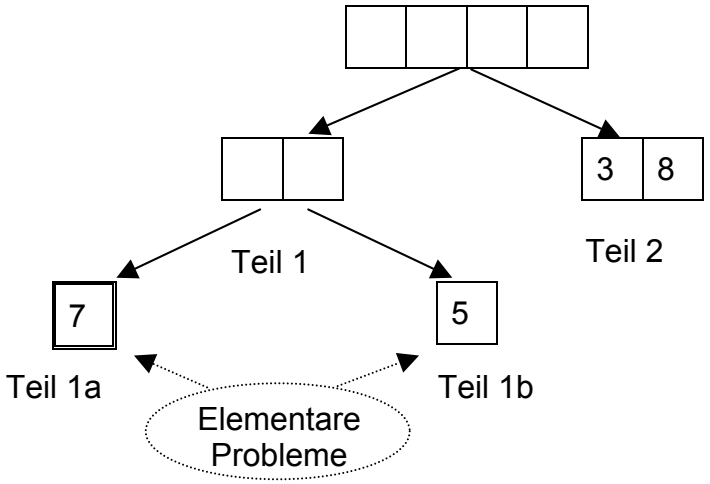


1. Rekursionsstufe

2. Rekursionsstufe

Schritt 2 für die Reihe in Teil 1 lautet:
Sortiere Teil 1a und 1b separat mit dem gleichen Verfahren.

Betrachtung von Teil 1a:
Man sortiert Teil 1a nach dem gleichen Verfahren wie die Ausgangsreihe.



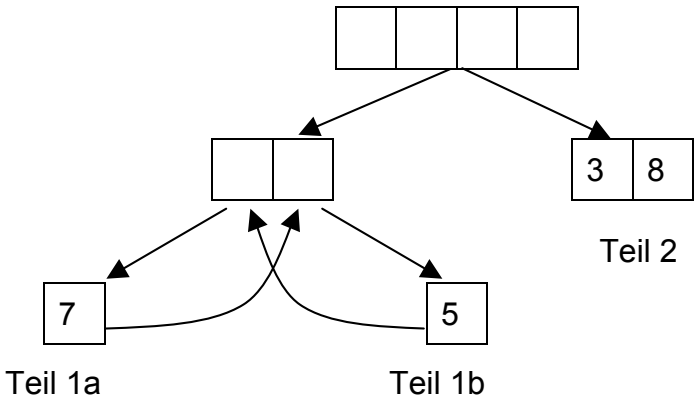
Teil 1a hat nur ein Element, daher ist es ein elementares Problem. Eine Reihe mit einem Element ist immer sortiert, daher kann man mit der Anwendung des Verfahrens auf Teil 1 fortfahren.

Teil 1a ist fertig. Teil 1b muss noch sortieren werden.

Da Teil 1b ein Element enthält, ist es auch schon sortiert. Man fährt eine Hierarchiestufe höher mit Teil 1 fort.

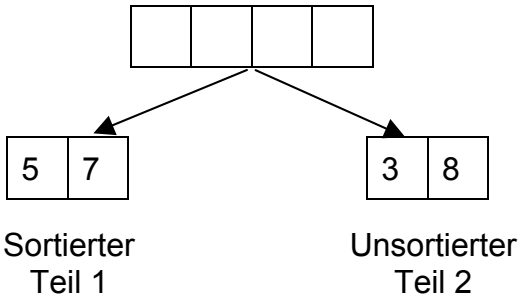
Für Teil 1 wurde Schritt 2 angewendet. Beide Teile sind nun sortiert, und es folgt Schritt 3 für Teil 1. Darin werden die Elemente der zwei Teilreihen 1a und 1b verglichen und als sortierte Reihe Teil 1 vereinigt.

7 < 5? → Nein → Deshalb wird 5 an die erste Stelle von Teil 1 geschrieben.

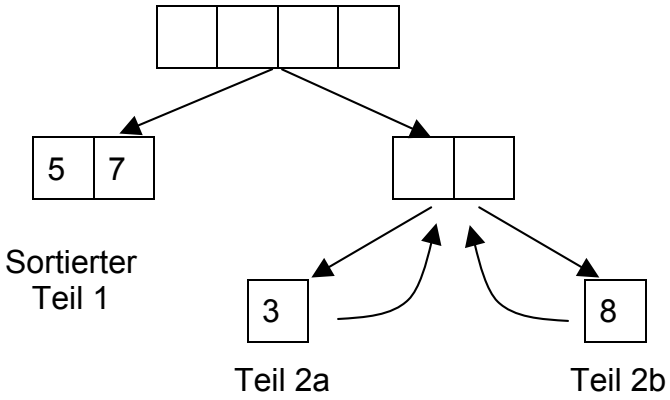


Man hat somit Teil 1 sortiert. Auf der obersten Stufe stehen noch die Sortierung von Teil 2 und die Verschmelzung zum Ganzen aus.

Die neue Situation sieht so aus:

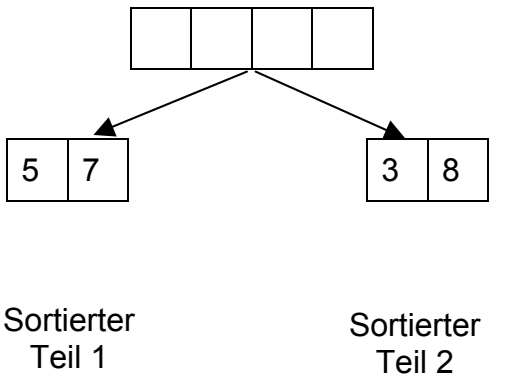


Teil 2 unterliegt dem gleichen Verfahren; er wird in zwei kleinere Teile 2a und 2b getrennt.



Teil 2a und 2b können mit einem Element nicht mehr weiter aufgeteilt werden. Sie sind elementare Probleme und schon sortiert. Nun fügt man Teil 2a und 2b wieder zu einem sortierten Teil 2 zusammen.

Die neue Situation:



Der jetzige Status der Sortierung ist nicht neu, sondern der Situation vor dem Sortieren von Teil 2 gleich. Ein Computer kann bei diesem Verfahren nicht erkennen, ob eine Teilfolge schon sortiert ist. Für ihn ist es weniger aufwendig die Folge zu teilen, als zu prüfen, ob eine Teilfolge schon sortiert ist. Nachdem Teil 1 und 2 sortiert sind, müssen sie vereinigt werden.

Vereinigung der beiden Teile = Schritt 3 für die Ausgangsfolge

5	7	und	3	8

Man vergleicht die ersten Elemente der beiden Teilfolgen.
5 < 3? → Nein → 3 kommt an die erste Stelle.

3			

5 < 8? → Ja → 5 kommt an die nächste Stelle.

3	5		

7 < 8? → Ja → 7 ist die nächste Zahl.

3	5	7	

8 ist die letzte Ziffer in der Gesamtfolge.

3	5	7	8

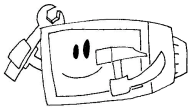
Für die Ausgangsfolge wurden alle drei Schritte durchgeführt. Man ist wieder auf der obersten Hierarchiestufe angekommen. Die Sortierung ist beendet.

Was wurde gemacht?
Man hat eine Zahlenreihe aufgeteilt, bis nur noch elementare Probleme vorlagen. Diese elementaren Probleme hat man gelöst und sie schrittweise mit anderen Teillösungen kombiniert, um eine Gesamtlösung zu erhalten. Der ursprüngliche Problemfall ist gelöst.

Zusätzliche Informationen:
Das Verfahren wird nochmals vorgeführt. Gehe bei Bedarf auf den untenstehenden Link.
Drücke auf `START`.
Die Farben sind codiert:
Der unsortierte Teil der Zahlenreihe ist rot dargestellt.
Der sortierte Teil der Zahlenreihe ist dunkelblau dargestellt.
Die Elemente, die betrachtet werden, sind hellblau gekennzeichnet.
Drücke auf `CONTINUE`.
Gib im hellblauen Feld eine `Zahl` zwischen 2 und 12 ein. Damit wird die Anzahl der Elemente in der Zahlenreihe festgelegt.
Jetzt kannst du entweder mit `NEXT` den nächsten Schritt oder mit `FINISH` das ganze Verfahren ohne Zwischenhalt ausführen lassen.

Link:
www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html

Auf der CD sind der Begriff „Rekursion“ und der Aufbau des Verfahrens in der Datei „Mergesort.ppt“ erläutert.



Lernkontrolle:

Aufgabe 3.4 a)
Was bedeutet Rekursion? .....
.....

Beschreibe kurz in eigenen Worten die drei Schritten von Merge-Sort:
Schritt 1: .....
.....
Schritt 2: .....
.....
Schritt 3: .....

Aufgabe 3.4 b)
Führe Merge-Sort für die Zahlenfolge (3, 8, 5, 2, 7, 1) durch und schreibe die Zwischenschritte auf.

Aufgabe 3.4 c)
Zähle die Vergleiche, die Merge-Sort für die Zahlenreihe aus Aufgabe 3.6 a) benötigt.

Aufgabe 3.4 d)
Die Zahlenreihe aus 3.6 a) ist identisch mit der in Kapitel 1 Aufgabe 1.3 a). Betrachte die benötigten Vergleiche zur Lösung der Aufgaben. Was stellst du fest, wenn du die beiden Verfahren gegenüberstellst?

3.5 Teile-und-Herrsche an Beispielen

Nach der Veranschaulichung durch ein konkretes Beispiel und mit dem Grundlagenwissen über rekursive Verfahren wird das Beispiel 1 aus Kapitel 3.1 nochmals genauer betrachtet.

Im Beispiel 1, dem Multiplizieren der Zahlen 163 × 4852, liegt eine zweistufige Rekursion vor. Das bedeutet, man ruft das angewendete Verfahren zweimal auf – einmal für die erste, einmal für die zweite Zahl. Es ist unabhängig von der Zahlengrösse. Das entspricht nicht ganz dem Teile-und-Herrsche-Konzept. Im folgenden Beispiel 2 wird eine vollständige Rekursion durchgeführt.

Beispiel 2:
Das vorliegendende Problem wird Min-Max-Problem genannt. Aufgabe ist es, in der Zahlenreihe das Minimum und das Maximum mit Hilfe eines Teile-und-Herrsche-Verfahrens zu bestimmen.

Ablauf des Verfahrens:

1. Schritt Teile die Zahlenreihe in zwei Teile.
2. Schritt Bestimme für die erste und zweite Teilreihe gesondert das Minimum und das Maximum.
3. Schritt Vergleiche die Minima der beiden Teilreihen und betrachte das Kleinere als Gesamtminimum. Mach das Gleiche mit den zwei Maxima.

Zu Beginn jedes Teile-und-Herrsche-Verfahrens muss man das elementare Problem festlegen.

Überlege, was das elementare Problem sein könnte und schreib es auf.

.....
.....
.....

Es gibt zwei Möglichkeiten für das elementare Problem:

1. Eine Reihe mit zwei Elementen, in der eine Zahl das Minimum und die andere Zahl das Maximum ist.
2. Eine Reihe mit einem Element, das gleichzeitig sowohl Minimum als auch Maximum ist.

Die erste Möglichkeit ist günstiger, da man die Reihe nur in Zweiergruppen aufteilen muss. Dadurch bricht die Rekursion schneller ab. Würde man aber z. B.

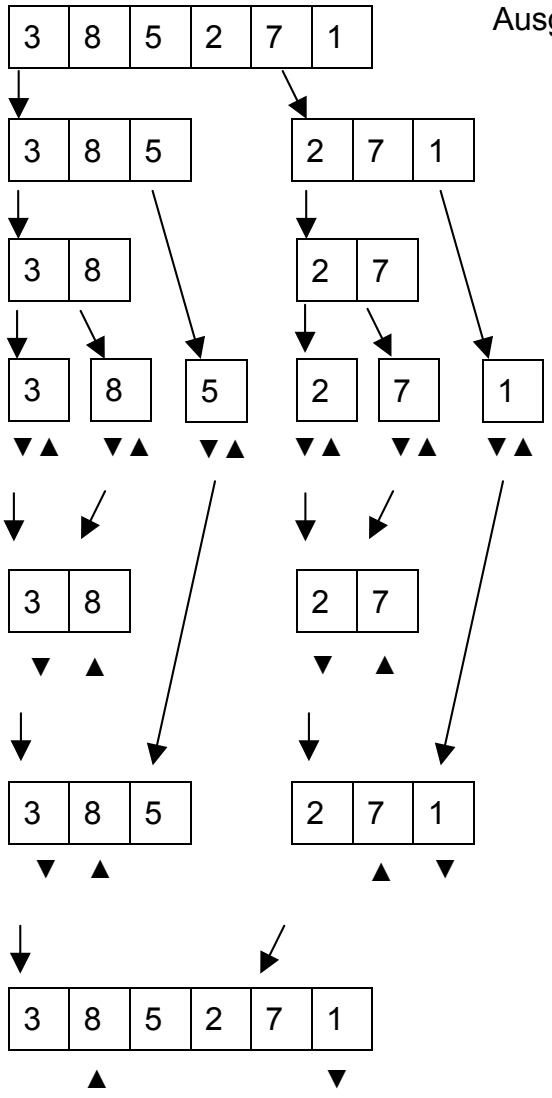
eine Reihe mit fünf Elementen betrachten, läge ein ungünstiger Spezialfall vor. In diesem Fall wählt man die zweite Möglichkeit und teilt in Reihen mit nur einem Element auf.

Nachdem man die Voraussetzungen, also den Ablauf des Verfahrens und das elementare Problem bestimmt hat, kann man es an einer Zahlenfolge durchspielen.

Gesucht wird in der folgenden Zahlenreihe das Minimum und das Maximum, ohne die Reihe zu sortieren. Das elementare Problem ist hier die Reihe mit einem Element.

Zeichenerklärung:

▼ Steht für das kleinste Element, also das Minimum.
▲ Steht für das Maximum.



Ausgangsreihe

Teilen, Rekursionsstufe 1

Teilen, Rekursionsstufe 2

Nochmals teilen, Stufe 3

Elementare Probleme:
Alle Elemente gleichzeitig
Minimum und Maximum.

Vergleiche: (3 mit 8) und (2 mit 7)

Vergleiche die Minima:
(3 mit 5) und (2 mit 1)
Vergleiche die Maxima:
(8 mit 5) und (7 mit 1)

Vergleiche die Minima: (3 mit 1)
Vergleiche die Maxima: (8 mit 7)

Am Schluss erhält man das Minimum (1) und das Maximum (8) der ganzen Zahlenreihe.

Dieses Beispiel benötigt mehrere Rekursionsstufen. Deshalb wird es vollständig durch das Teile-und-Herrsche-Prinzip gelöst. Die Anzahl der Rekursionsstufen ist abhängig von der Grösse der Ausgangsreihe. Je grösser die Ausgangsreihe, desto mehr Rekursionsstufen benötigt man. Das steht im Gegensatz zu Beispiel 1, in dem nur zwei Rekursionsstufen vorlagen.

3.6 Spezielle Eigenschaften von Merge-Sort

Nachteil:
Sobald sich ein Verfahren etwas „merken“ muss, braucht es einen Zwischenspeicher und somit auch mehr Platz. Weil Merge-Sort die Reihen immer wieder aufteilt und zwischenspeichert, braucht es viel mehr Speicherplatz als die bisher vorgestellten Verfahren. Die in Kapitel 1 und 2 genannten Verfahren haben lediglich zwei Zahlen miteinander verglichen und sie ggf. ausgetauscht. Daher brauchen sie nur einen Zwischenspeicher für ein Element und keinen zusätzlichen Speicherplatz für Teilreihen.

Bei völlig unsortierten Reihen ist das Merge-Sort Verfahren recht schnell. Bereits vorsortierte Reihen können jedoch mit anderen Verfahren schneller bearbeitet werden.

Vorteil:
Merge-Sort benötigt im Gegensatz zu den anderen Verfahren nur sehr wenige Vergleiche. Diese Eigenschaft macht Merge-Sort sehr schnell, da Vergleiche zwischen Zahlen sehr zeitaufwendig sind.

Lernkontrolle:

Aufgabe 3.6 a)
Merge-Sort braucht grundsätzlich sehr wenige Vergleiche. Wie kann man noch mehr Vergleiche einsparen?

Aufgabe 3.6 b)
Überlege, welche Reihe am wenigsten Vergleiche benötigt.

Aufgabe 3.6 c)
Gehe auf den bekannten Link:
www.cs.pitt.edu/~kirk/cs1501/animations/Sort3.html.

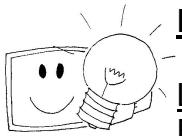
Wähle „Merge Sort“ aus und beobachte, wie das Verfahren durchgespielt wird. Welche Unterschiede oder Gemeinsamkeiten zum hier vorgestellten Merge-Sort Verfahren kannst du feststellen?

Zusammenfassung

In diesem Kapitel wurde ein fundamentales Konzept der Informatik erörtert: Teile-und-Herrsche. Ein Verfahren, das auf diesem Konzept beruht, geht immer mit einem rekursiven Programmaufbau einher.
Man teilt dabei das Ursprungsproblem in kleinere Teile und führt für alle Teilprobleme nochmals das gleiche Verfahren durch. Man teilt so lange, bis nur noch elementare Probleme vorliegen und löst diese anschliessend. Die elementaren Probleme sind die Abbruchbedingung für das Teilen. Die Lösungen der elementaren Probleme fügt man zu Lösungen der Teilprobleme zusammen und vereinigt sie dann zu einer Gesamtlösung.

Merge-Sort teilt in einem ersten Schritt die Sortieraufgabe in zwei Teile. Im zweiten Schritt werden beide Teile nacheinander separat sortiert. Zum Schluss werden die beiden sortierten Teilfolgen miteinander zu einer gemeinsamen sortierten Reihe verschmolzen.

Gehe nun zum Kapiteltest.



Lösungen zum Kapitel 3

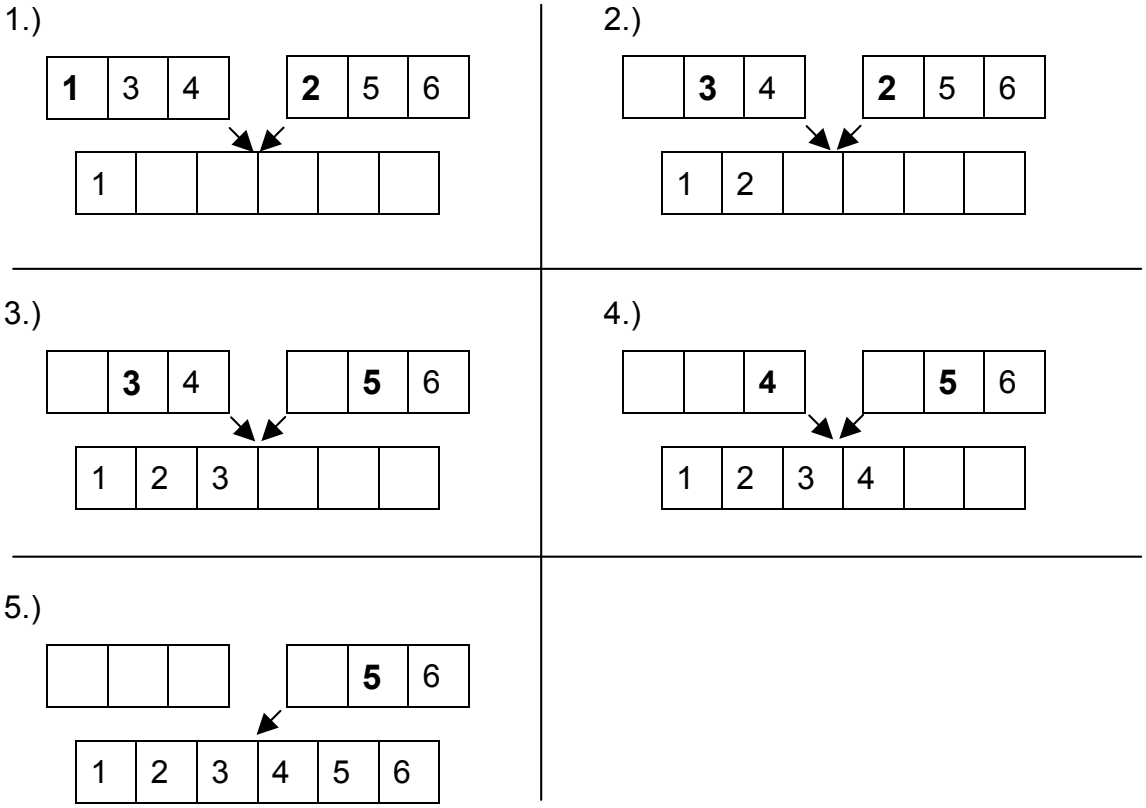
Lösung 3.1 a)

Einige Beispiele:

- zwei grosse Zahlen multiplizieren
- eine Reihe sortieren
- eine Wohnung putzen:
  Um eine Wohnung zu putzen, muss man z. B. Staub saugen, aufräumen und wischen. Man kann nun die Aufgabe „ganze Wohnung auf Vordermann bringen“ in verschiedene Teile, die Zimmer, aufteilen und in jedem Staub saugen, aufräumen und wischen. Die Zimmer kann man bei Bedarf weiter unterteilen. Hier kommt es auf die Definition des elementaren Elements an. Wenn man alle Zimmer bearbeitet hat, ist auch die ganze Wohnung wieder sauber und die Aufgabe gelöst.
  Man könnte die Aufgabe auch anders lösen, indem man in der ganzen Wohnung zuerst Staub saugt, aufräumt und dann wischt. Diese Aufteilung widerspricht aber dem Konzept, da sich die einzelnen Teilaufgaben nicht auf die gleiche Weise lösen lassen.
- ein 5-Gang-Menü kochen:
  Möchte man ein mehrgängiges Menü essen, dann bereitet man einen Gang nach dem anderen zu, denn sonst wird es unübersichtlich in der Küche. Für jeden Gang stellt man zuerst die Zutaten griffbereit hin, bereitet dann den Gang zu und serviert ihn anschliessend.
- etc.

Lösung 3.2 a)
Hat man zwei sortierte Zahlenfolgen, z. B. (1,3,4) und (2,5,6), muss man jeweils die beiden ersten Elemente der Folgen vergleichen und kann das kleinere Element in die grosse Reihe einfügen. Das macht man, bis eine Reihe leer ist. Den Rest der anderen Reihe kann man dann ohne weitere Vergleiche hinten anfügen.

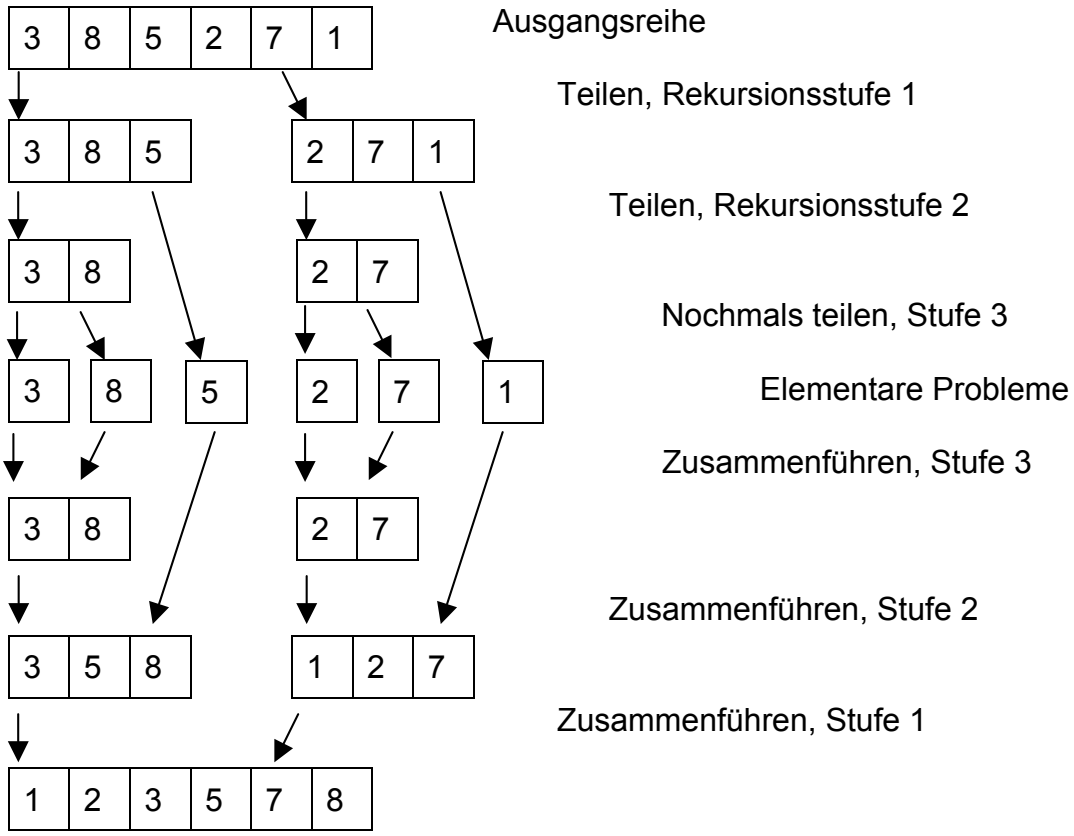
Die fett gedruckten Elemente werden miteinander verglichen. Das Kleinere wird in die grosse Reihe geschrieben.



Lösung 3.2 b)
Das elementare Problem beim Sortieren einer Zahlenfolge ist eine Reihe, die aus einem einzigen Element besteht.

Lösung 3.4 a)
Rekursion bedeutet, dass etwas auf sich selbst verweist oder sich selbst aufruft. Im vorliegenden Fall drückt es die Anwendung des immer gleichen Verfahrens aus. Nach jedem Aufruf wird der zu lösende Problemfall durch die Aufteilung kleiner und somit besser und schneller lösbar.

Lösung 3.4 b)



Endergebnis ist die sortierte Reihe.

Lösung 3.4 c)
Vergleiche werden nur beim Zusammenführen durchgeführt.
Man vergleicht in der untersten Stufe (Stufe 3) die Ziffer (3) mit (8) und (2) mit (7).
→ **2** Vergleiche
In der Stufe 2 muss man für den ersten Teil (3, 8) mit (5) vergleichen. Um die richtig sortierte Reihe zu erhalten, muss man (3 mit 5) und (8 mit 5) vergleichen.
→ **2** Vergleiche
Im zweiten Teil werden (2, 7) mit (1) verglichen, d. h. (2) mit (1).
→ **1** Vergleich
In Stufe 1 muss man noch (3, 5, 8) mit (1, 2, 7) vergleichen.
→ **5** Vergleiche

2 + 2 + 1 + 5 Vergleiche = **10 Vergleiche** für diese Reihe.

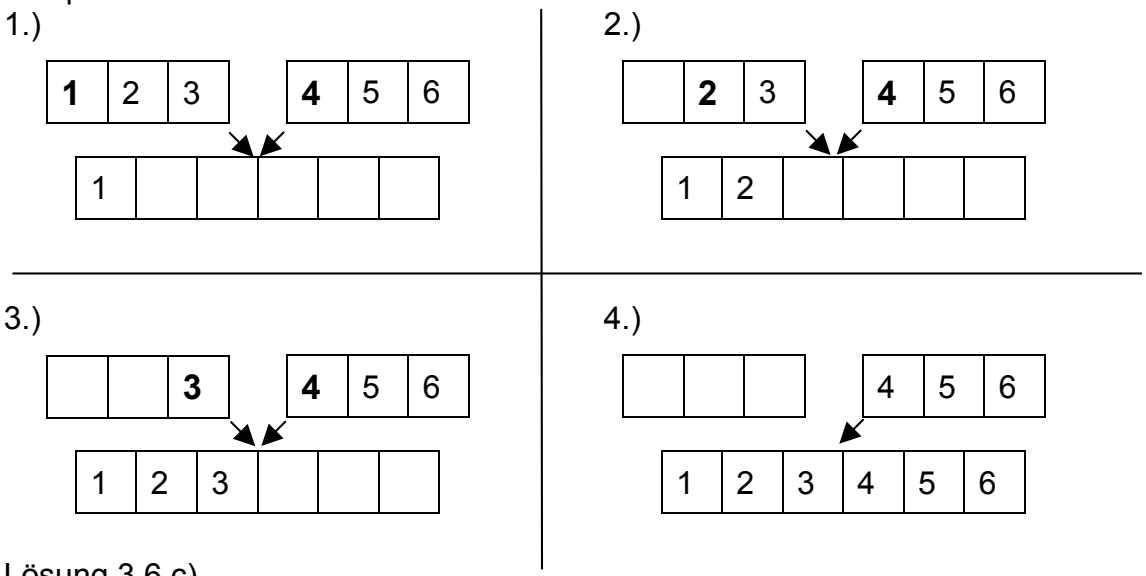
Lösung 3.4 d)
Im Gegensatz zu Bubble-Sort, mit dem man 30 Vergleiche machen muss, braucht Merge-Sort nur zehn Vergleiche.

Das weist darauf hin, dass Merge-Sort schneller und effizienter ist als Bubble-Sort.

Lösung 3.6 a)
Man kann zusätzliche Vergleiche sparen, wenn eine Teilfolge beim Zusammenführen leer ist, während die andere noch mehr als ein Element hat, s. Lösung 3.3 a). Dort kann man am Schluss die Zahlen 5 und 6 ohne einen weiteren Vergleich in die grosse Folge einfügen.

Lösung 3.6 b)
Eine bereits sortierte Folge braucht am wenigsten Vergleiche. Man schreibt beim Zusammenführen immer zuerst eine Teilfolge in die grössere Reihe. Dabei werden alle Elemente der ersten Teilfolge mit dem ersten Element der zweiten Teilfolge verglichen. Sobald eine Teilreihe leer ist, darf man die Elemente der anderen Teilfolge ohne weitere Vergleiche anfügen.

Beispiel:

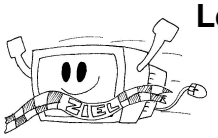


Lösung 3.6 c)
Die Elemente werden im ersten Durchgang in Zweiergruppen aufgeteilt. Es wird innerhalb dieser Zweiergruppen sortiert. Im nächsten Durchgang betrachtet man Vierergruppen. Man fügt die sortierten Zweiergruppen in die Vierergruppen ein. Die Gruppen werden so lange zusammengezogen, bis die ganze Reihe aus einer grossen Gruppe besteht.

Der Grundgedanke bleibt gleich: Man teilt die Folge in kleinere Teilfolgen auf, sortiert diese und kombiniert die Teile zu einem Ganzen.

Anders als das in diesem Leitprogramm vorgestellte Verfahren wird in dem Programm keine komplizierte Struktur aufgebaut, sondern innerhalb der ursprünglichen Zahlenreihe mit Hilfe einer zweiten Zahlenreihe sortiert.

4. Kapitel: Quick-Sort



Lernziele:

- Du kannst Quick-Sort an einem Beispiel erklären.
- Du kennt den Vorteil von Quick-Sort gegenüber den bisherigen Verfahren.
- Du kannst drei verschiedene Arten von Quick-Sort benennen und ihre Unterschiede erklären.

4.1 Die Grundidee von Quick-Sort

Quick-Sort ist ein weiteres Sortierverfahren, das nach dem Prinzip „Teile-und-Herrsche“ funktioniert. Quick-Sort teilt die zu sortierende Zahlenreihe in zwei Teile und sortiert sie einzeln und rekursiv. Das bedeutet, dass auch diese zwei neuen Teile mit Quick-Sort sortiert werden.

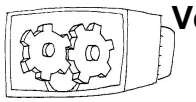
Oberflächlich betrachtet ähnelt das Verfahren Merge-Sort. Die Grundidee von Quick-Sort gleicht jedoch Selection-Sort aus Kapitel 2.

Erinnerst du dich noch? Selection-Sort platziert das kleinste Element an den richtigen Platz. Dann sortiert es das zweitkleinste Element usw., bis die ganze Zahlenfolge sortiert ist. Selection-Sort sucht in jedem Schritt das Minimum der unsortierten Folge. Das ist aufwendig, da man für die Minimumsuche alle Elemente betrachten muss.

Wäre es nicht einfacher, das vorderste Element zu nehmen und es an die richtige Position zu stellen? So macht es Quick-Sort.

4.2 Quick-Sort

Betrachte zuerst die PowerPoint-Demonstration „Quicksort“ auf der CD.
Hier werden die einzelnen Schritte von Quick-Sort zusammengefasst:



Verfahren:

1. Wähle ein Element als Pivot-Element.
2. Bestimme den Platz für das Pivot-Element. Vertausche dazu alle dunkleren Elemente der linken Seite mit allen helleren Elementen der rechten Seite.
3. Platziere das Pivot an der richtigen Stelle.
4. Sortiere die linke Hälfte nach dem gleichen Schema.
5. Sortiere die rechte Hälfte nach dem gleichen Schema.

Soll ein einzelnes Element oder eine leere Zahlenfolge sortiert werden, muss man nichts weiter unternehmen. Beides sind bei der Anwendung

von Quick-Sort ***elementare Probleme,*** die gleichzeitig auftreten können. In beiden Fällen muss Quick-Sort nichts tun, denn sowohl ein einzelnes Element als auch eine leere Zahlenfolge sind bereits sortiert.

Lernkontrolle:

Aufgabe 4.2 a)
Sortiere die folgende Zahlenreihe mit Quick-Sort:

3 8 5 2 7 1

Aufgabe 4.2 b)
Soll mit Quick-Sort eine bereits sortierte Reihe geordnet werden, läuft der gleiche Prozess wie bei einem anderen Sortierverfahren ab. Welches Verfahren ist gemeint? Warum ist das so?

4.3 Wie quick ist Quick-Sort?

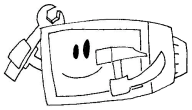
Betrachtet man die Anzahl der notwendigen Vergleiche, stellt man fest, dass Quick-Sort im ersten Schritt jedes Element mit dem Pivot-Element vergleichen muss. Daher benötigt es im ersten Schritt fast gleich viele Vergleiche wie Selection-Sort oder Bubble-Sort. Da sich die zwei Pfeile überkreuzen müssen, braucht man sogar zwei Vergleiche mehr.

Quick-Sort hat jedoch nach dem ersten Durchgang zwei verkleinerte Aufgaben zu lösen, die im Idealfall nur noch halb so gross sind wie bei den anderen Verfahren.
Während Selection-Sort im zweiten Durchgang Vergleiche in der Menge „Anzahl Elemente minus 2“ machen muss, benötigt Quick-Sort nur noch die Hälfte der Vergleiche. Im dritten Schritt halbiert sich der Aufwand mit Quick-Sort nochmals. Man braucht also nur noch einen Viertel der Vergleiche. Bei Fortsetzung dieser Logik wird deutlich, dass Quick-Sort den Sortieraufwand deutlich reduziert.

Ein möglicher Einwand gegen die Effektivität wäre: „Ja, Quick-Sort muss nur die Hälfte der Vergleiche machen, aber das zweimal, weil es zwei neue Teile erzeugt. Der Vorteil gegenüber den anderen Verfahren besteht also nicht.“
Dieser Einwand wäre falsch, denn nachdem Quick-Sort den zweiten Schritt in der linken Seite ausgeführt hat, ist bereits ein Element wieder am richtigen Platz. Die andere Seite ist ein eigener Durchgang.
Ebenso verhält es sich mit den Anzahl der notwendigen Vertauschungen. Weil die zu sortierenden Zahlenfolgen geringer werden, gibt es weniger Elemente, die vertauscht werden müssen.

Oben wurde vom Idealfall gesprochen. Wie verhält sich Quick-Sort jedoch, wenn nicht der Idealfall eintritt?

Im schlechtesten Fall teilt Quick-Sort die Zahlenfolge in eine leere Folge und in eine weitere Folge, die um ein Element kleiner ist als die bisherige Folge. Dadurch wird die mit Quick-Sort zu sortierende Folge wie bei Selection-Sort in jedem Schritt ein Element kleiner. Im schlechtesten Fall ist Quick-Sort etwa so schnell wie Selection-Sort. Wenn du mehr zu diesem Thema wissen möchtest, findest du zusätzliche Informationen im Additum.



Lernkontrolle:

Aufgabe 4.3 a)
Ordne die Zahlen 1 bis 7 so an, dass Quick-Sort möglichst wenige Vergleiche und möglichst wenige Vertauschungen machen muss. Wie viele Vergleiche und Vertauschungen sind es dann?

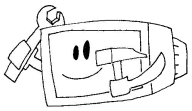
Aufgabe 4.3 b)
Ordne nun die Zahlen 1 bis 7 so an, dass Quick-Sort möglichst viele Vergleiche machen muss. Wie viele Vergleiche sind es?

4.4 Verschiedene Versionen von Quick-Sort

In der PowerPoint-Demonstration war man bei der Bestimmung des Pivot-Elements nicht wählerisch – man hat einfach das Erstbeste genommen. Es existieren jedoch Versionen von Quick-Sort, die immer das letzte Element einer Zahlenfolge als Pivot-Element definieren.
Im Unterschied zum PowerPoint-Beispiel gibt es dann eine kleine Änderung: Wenn man das Pivot-Element an seinen richtigen Platz stellen will, muss man es mit dem grösseren der beiden Elemente tauschen. Mit dem Element also, auf das der grüne Pfeil zeigt.
Die beiden Versionen unterscheiden sich nicht stark voneinander. Es ist Geschmackssache, welche man wählt.

Quick-Sort ist wenig effektiv, wenn man eine bereits sortierte Zahlenfolge damit ordnen will. Ursache dafür ist, dass Quick-Sort die Zahlenfolge in jedem Schritt in eine leere und in eine um 1 reduzierte Zahlenfolge teilt, wenn das Pivot entweder das grösste oder das kleinste Element der Zahlenfolge ist.
Um dieses Problem zu vermeiden, wird die Auswahl des Pivot-Elements geändert. Man betrachtet sowohl das erste Element als auch das Element in der Mitte und das letzte Element der Zahlenfolge. Wenn man das Element mit dem mittleren Wert als Pivot-Element auswählt, ist man sicher, dass man weder das kleinste noch das grösste Element bestimmt hat. Es existiert also mindestens noch ein grösseres und ein kleineres Element als das Pivot-Element.

Lernkontrolle



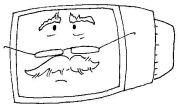
Aufgabe 4.4 a)
Sortiere diese Zahlenfolge mit dem Quick-Sort Verfahren, welches das letzte Element als Pivot-Element auswählt.

6 3 9 5 8

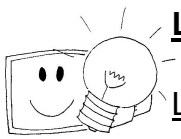
Aufgabe 4.4 b)
Welches ist der schlechteste Fall bei der Quick-Sort Variante, die drei verschiedene Elemente betrachtet, bevor es ein Pivot-Element wählt?

Zusammenfassung

Quick-Sort beruht auf dem Konzept „Divide and Conquer“. Als erstes wählt es ein Pivot-Element aus. Die Zahlenfolge wird damit in zwei Teile geteilt: In einen linken Teil mit allen Zahlen, die kleiner sind als das Pivot und in einen rechten Teil mit allen grösseren Zahlen. Anschliessend wird erst der linke Teil, dann der rechte Teil mit Quick-Sort sortiert.
Der Vorteil des Verfahrens liegt darin, dass die neuen Teile kleiner sind und deshalb weniger Vergleiche und Vertauschungen notwendig werden. Die Auswahl des Pivot-Elements beruht auf verschiedenen Strategien. Man kann dafür das erste oder das letzte Element auswählen. Die beste Strategie besteht darin, sich das erste, das letzte und das mittlere Element anzusehen und den mittleren Wert als Pivot-Element zu definieren.



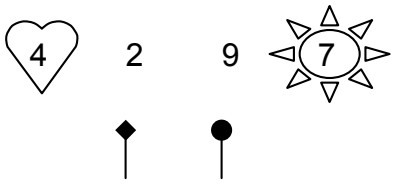
Wenn du alles verstanden hast, wartet der Kapiteltest auf dich.



Lösungen zu Kapitel 4

Lösung 4.2 a)

Zeichenerklärung:

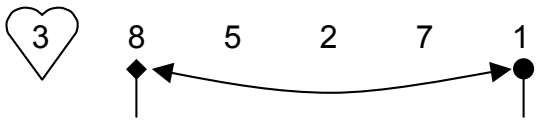


Das Herz kennzeichnet das aktuelle Pivot- Element.

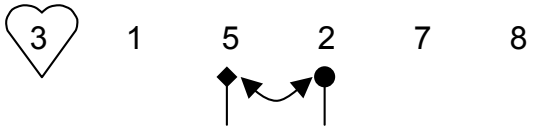
Die Sonne kennzeichnet ein bereits sortiertes Element, also ein ehemaliges Pivot-Element.

Der eckige Pfeil entspricht dem grünen Pfeil der PowerPoint-Demonstration. Er startet von links. Der runde Pfeil entspricht dem blauen Pfeil der PowerPoint-Demo. Er startet von rechts.

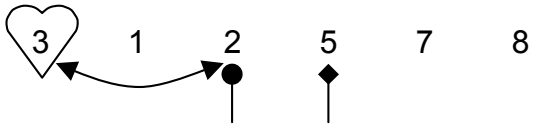
Los geht ‘s!



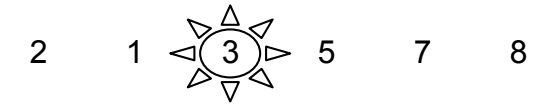
Die 3 steht an erster Stelle und ist das erste Pivot-Element.
Da der eckige Pfeil schon auf ein grösseres Element und der runde Pfeil auf ein kleineres Element zeigt, kann man diese zwei Zahlen tauschen.



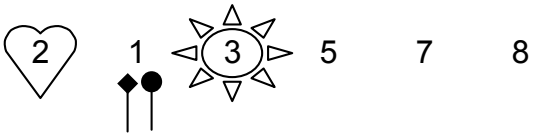
Als Nächstes bleibt der eckige Pfeil auf der 5 stehen und der runde Pfeil auf der 2. Auch diese zwei Elemente werden getauscht.



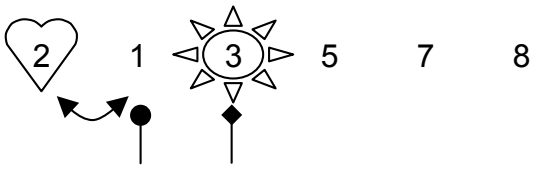
Nun haben sich die zwei Pfeile überkreuzt. Damit das Pivot-Element an den richtigen Platz kommt, tauscht man es mit der 2.



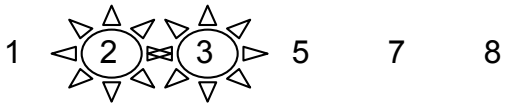
Die 3 ist am richtigen Platz. Man sortiert zuerst die neu entstandene linke Seite. Die 2 wird zum neuen Pivot-Element.



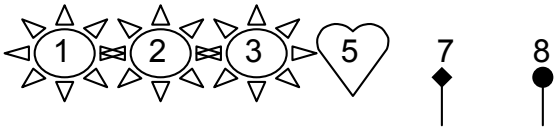
Weil die 1 kleiner ist als die 2, wandert der eckige Pfeil eine Position weiter nach rechts, während der runde Pfeil wartet.



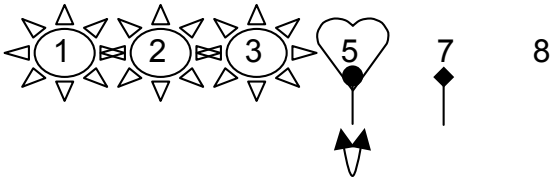
Da sich die Pfeile bereits gekreuzt haben, ist der richtigen Platz für die 2 gefunden. Man tauscht sie mit der 1.



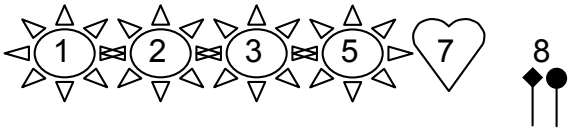
Die 2 ist jetzt auch an der richtigen Stelle. Man muss zuerst die linke Seite, also die 1, und danach die rechte Seite mit der leeren Zahlenfolge zwischen der 2 und der 3 sortieren.
Weil es sich bei der linken Seite um ein einzelnes Element handelt, ist sie schon sortiert.
Die leere Folge ist bereits sortiert. Man hat also die komplette Seite links von der 3 sortiert und kann sich der rechten Seite zuwenden. Hier wird die 5 zum Pivot.



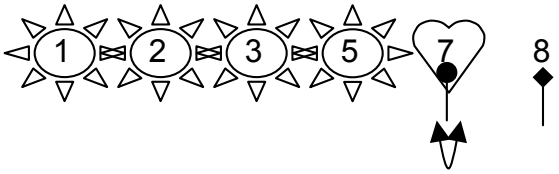
Da die 7 grösser ist als die 5, muss der eckige Pfeil warten, während der runde Pfeil bis zur 5 nach vorne wandern kann.



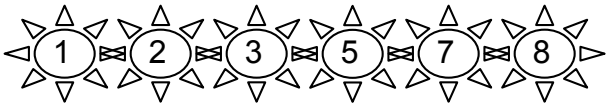
Nun haben sich die Pfeile gekreuzt und die 5 wird mit sich selbst vertauscht. Die so entstandene linke Seite ist leer, weil die 5 das kleinste Element war. Man kann direkt mit der rechten Seite fortfahren. Die 7 wird zum nächsten Pivot-Element.



Weil 8 grösser ist als 7, wartet der eckige Pfeil, während der runde Pfeil zur 7 wandert.



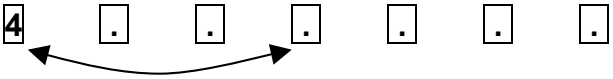
Nun haben sich die Pfeile wieder gekreuzt. Die 7 wird mit sich selbst vertauscht. Die neu entstandene linke Seite ist wieder die leere Folge und somit schon sortiert. Die rechte Seite besteht nur aus einem einzigen Element und ist deshalb auch schon sortiert.



Fertig!

Lösung 4.2 b)
Wenn man Quick-Sort eine schon sortierte Reihe gibt, verhält es sich wie Selection-Sort. Es definiert das erste Element als Pivot-Element und vergleicht alle anderen Elemente damit. Weil das erste Element das Kleinste ist, ist es sortiert. Die neu entstandene linke Teilfolge ist somit die leere Folge. Die neue rechte Teilfolge ist die gleiche Folge ohne das erste Element. Mit der leeren Folge passiert nichts, deshalb macht Quick-Sort mit der rechten Teilfolge weiter. Hier ist das vorderste Element wiederum das Kleinste. Quick-Sort vergleicht es mit den restlichen Elementen, es bleibt aber an seiner Stelle stehen.
Die Anzahl Vertauschungen entspricht denen von Selection-Sort. Man benötigt jedoch jedes Mal zwei Vergleiche mehr, weil sich die Pfeile überkreuzen müssen. Die Zahlenreihe sieht jedoch nach jedem Schritt so aus, als hätte man Selection-Sort angewendet.

Lösung 4.3 a)
Man beginnt mit der Überlegung, dass die Reihe in zwei gleich grosse Teile geteilt werden soll. Um das zu erreichen, muss das mittlere Element das erste Pivot-Element sein. Deshalb muss die 4 vorne stehen, um möglichst wenige Vergleiche zu machen.



Im nächsten Schritt wird die 4 mit dem Element aus der Mitte vertauscht, weil das der richtige Platz für die 4 ist. Im Umkehrschluss steht das mittlere Element jetzt auf dem ersten Platz.
Das Element am ersten Platz ist nichts anderes als das Pivot-Element der linken Teilreihe. Es soll diese Teilreihe in zwei gleich grosse Stücke teilen. Deswegen bestimmt man als Pivot die 2, denn sie ist die Mitte von (1, 2, 3). Die 2 muss also vorher in der Mitte der ganzen Reihe gestanden haben.

4 . . **2** . . .

Die Plätze zwischen 2 und 4 füllt man am besten mit der 1 und der 3 auf. Wenn es möglich ist so, dass nur wenige Elemente getauscht werden müssen.

4 **1** **3** **2** . . .

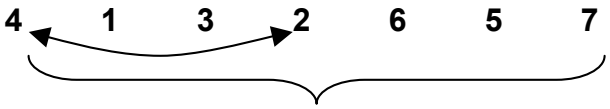
Mit der rechten Seite verfährt man gleich.
Als Pivot wünscht man sich wieder das mittlere Element. Das ist die 6.
Sie kommt an die erste Stelle der rechten Teilfolge.

4	1	3	2	6	.	.

Damit man wieder möglichst wenige Vertauschungen machen muss, setzt man die 5 und die 7 sortiert ein.

4	1	3	2	6	5	7

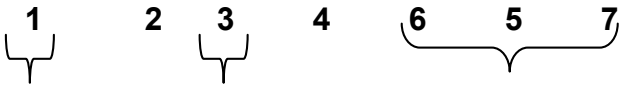
Wenn man hier Quick-Sort anwendet, benötiget man:



6 Vergleiche, denn jedes Element wird mit dem Pivot-Element verglichen.
+ 2 Vergleiche, weil sich die Pfeile überkreuzen.
1 Vertauschung, damit das Pivot-Element den richtigen Platz erhält.
Weil alle kleineren Elemente bereits auf der linken Seite stehen, benötigt man keine weiteren Vertauschungen.

Nun muss man diese zwei Teilreihen sortieren. Man beginnt mit der linken Seite.

Man braucht:
2 Vergleiche, weil die 1 und die 3 mit dem Pivot verglichen werden.
+ 2 Vergleiche, da sich die Pfeile kreuzen.
1 Vertauschung, damit das Pivot am richtigen Platz steht.

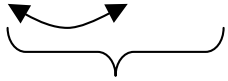


Es sind erneut zwei kleinere Teilreihen entstanden.

Um die 1 zu sortieren benötigt man:
0 Vergleiche, weil man ein einzelnes Element mit nichts vergleichen kann.
0 Vertauschungen, denn ein einzelnes Element steht immer am richtigen Ort.

Ebenso für die 3:
0 Vergleiche
0 Vertauschungen

1 2 3 4 6 5 7



Jetzt sortiert man die rechte Seite.

Man braucht:
2 Vergleiche, denn die 5 und die 7 werden mit dem Pivot verglichen.
+ 2 Vergleiche, da sich die Pfeile überkreuzen. Das heisst, die 5 und die 7 werden durch jeden Pfeil einmal mit dem Pivot-Element verglichen.
1 Vertauschung, denn das Pivot kommt an seinen Platz.

1 2 3 4 5 6 7

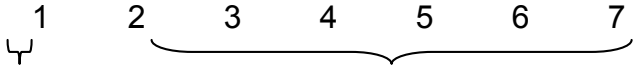
Erneut sind zwei kleinere Teile entstanden.
Für die einzeln stehenden Elemente 5 und 7 benötigt man:
0 Vergleiche
0 Vertauschungen.

Insgesamt braucht man:
8 + 4 + 0 + 0 + 4 + 0 + 0 = 16 Vergleiche
1 + 1 + 0 + 0 + 1 + 0 + 0 = 3 Vertauschungen

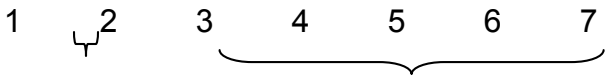
Lösung 4.3 b)
Zu dieser Aufgabe gibt es mehrere richtige Lösungen. Damit Quick-Sort möglichst viele Vergleiche macht, muss man verhindern, dass es von kleineren Teilen profitiert. Immer dann, wenn Quick-Sort ein Pivot-Element auswählt, muss dieses entweder das grösste oder das kleinste Element der zu sortierenden Zahlenfolge sein. Dadurch teilt Quick-Sort die Zahlenfolge in einen leeren Teil und in einen Teil auf, der ein Element weniger als vorher hat. Es kann also nicht von kleineren Teilen profitieren.
Der einfachste Fall ist, mit Quick-Sort eine schon sortierten Zahlenfolge zu bearbeiten. Das sieht so aus:

1 2 3 4 5 6 7

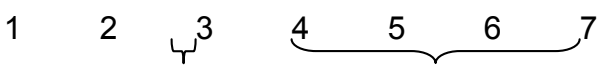
Zuerst wählt Quick-Sort die 1 als Pivot-Element. Nach dem ersten Durchgang sieht die Folge so aus:



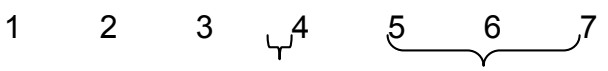
Man braucht 6 + 2 Vergleiche. Die 1 ist nun am richtigen Platz. Die neuen Teilfolgen sind: Die leere Folge und die sortierte Folge von 2 bis 7.
Die leere Folge ist bereits sortiert. Quick-Sort bearbeitet die rechte Teilfolge. Der nächsten Durchgang erfolgt analog:



Man braucht 5 + 2 Vergleiche. Die 2 ist am richtigen Platz. Die neuen Teilfolgen sind: Die leere Folge und die sortierte Folge von 3 bis 7. Die leere Teilfolge ist bereits sortiert. Quick-Sort kümmert sich nun um die rechte Teilfolge. Der nächsten Durchgang erfolgt wieder analog:



Die 3 ist am richtigen Ort. Man braucht 4 + 2 Vergleiche.



Die 4 ist am richtigen Ort. Man braucht 3 + 2 Vergleiche.

Die 5 ist am richtigen Ort. Man braucht 2 + 2 Vergleiche.

Die 6 ist am richtigen Ort. Man braucht 1 + 2 Vergleiche.
Im letzten Schritt registriert das Verfahren, dass auch die 7 sortiert ist, weil sie ein einzelnes Element ist.

Insgesamt musste Quick-Sort 8 + 7 + 6 + 5 + 4 + 3 = **33 Vergleiche** machen.

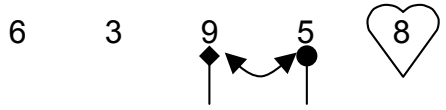
Diese Zahlenfolgen erreichen die gleiche Anzahl von Vergleichen:

7	1	2	3	4	5	6
7	2	3	4	5	6	1
1	7	2	3	4	5	6
7	6	5	3	4	2	1

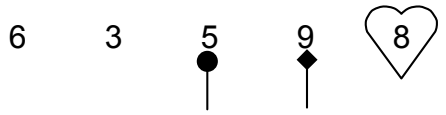
Es gibt noch mehr Lösungen. Falls deine Antwort hier nicht aufgelistet ist, kannst du prüfen, ob deine Lösung ebenfalls stimmt. Zähle dazu einfach die Vergleiche zusammen. Sind es auch 33, stimmt dein Resultat. Anderenfalls solltest du überprüfen, ob wirklich in jedem Schritt das Pivot-Element das grösste oder das kleinste Element ist.

Lösung 4.4 a)

Im ersten Schritt wird das letzte Element zum Pivot-Element, hier die 8.
Die Bedeutung der Pfeile kennt du schon.

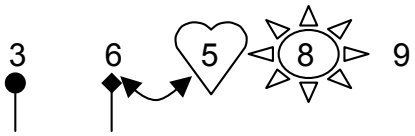


Die 5 und die 9 werden miteinander vertauscht.

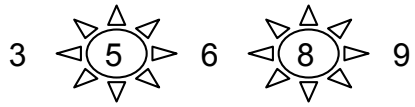


Nun haben sich die Pfeile überkreuzt; der Platz für das Pivot-Element ist gefunden. Will man aber die 8 an ihren richtigen Platz tauschen, ändert sich das Vorgehen. Man tauscht sie mit dem Grösseren der beiden Elemente 5 und 9. Das macht man, weil die 8 ganz hinten steht. Würde man sie mit der 5 tauschen, wären nicht mehr alle Elemente rechts von der 8 grösser als sie.

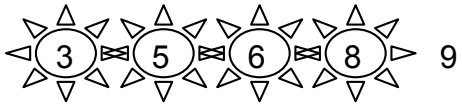
Die 8 ist nun an ihrem endgültigen Platz. Sie ist daher mit der Sonne gekennzeichnet. Man fährt auf der linken Seite fort. Die 5 wird zum neuen Pivot-Element. Da die Pfeile schon warten müssen, werden die dazugehörenden Elemente vertauscht.



Die Pfeile haben sich wieder überkreuzt, und man kann die 5 mit der 6 tauschen.



Man betrachtet die Seite links von der 5 und merkt, dass es sich um ein einzelnes Element handelt. Ein einzelnes Element ist bekanntlich sortiert. Man kann auf der rechten Seite weitermachen. Dort steht das Element 6 alleine, womit auch die rechte Seite sortiert ist.



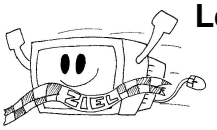
Nun betrachtet man die rechte Seite des ersten Pivot-Elements 8:

Weil die 9 alleine steht, ist die Sortierung abgeschlossen.

Lösung 4.4 b)

Der schlechteste Fall für diese Quick-Sort Variante tritt ein, wenn man das zweitgrösste oder das zweitkleinste Element als Pivot wählt. Es würde bedeuten, dass man gerade das grösste und das zweitgrösste oder das kleinste und das zweitkleinste Element in der Auswahl hatte. Dann besteht eine Teilfolge aus einem Element und die andere aus „Anzahl Elemente minus 2“ Elementen. Das ist kein grosser Gewinn für Quick-Sort.

5. Kapitel: Additum (Verschiedene Analysen)



Lernziele:

- Du kannst den minimalen Aufwand von verschiedenen Verfahren abschätzen.
- Du weisst, wie sich verschiedene Verfahren im aufwendigsten Fall verhalten.
- Wenn du etwas sortieren musst, kannst du zwischen geeigneten und ungeeigneten Verfahren für deine Aufgabe unterscheiden.

5.1 Einführung

In der Informatik werden Verfahren, wie sie in den letzten Kapiteln vorgestellt wurden, genauer auf ihre Vor- und Nachteile untersucht. Dort hat man es nicht mit zehn oder 100 Elementen zu tun, sondern mit Tausenden oder sogar Millionen von Elementen. Daher ist es wichtig zu wissen, wie lange ein Sortierverfahren braucht und wie viel Speicherplatz es benötigt.

5.2 Messverfahren

Wie kann man die verschiedenen Verfahren miteinander vergleichen? Es wäre nicht sinnvoll, lediglich die benötigte Sortierzeit zu messen, weil Computer unterschiedlich schnell sind. Neuere Computermodelle sind meistens schneller. Gesucht wird eine allgemeine Messgrösse für Sortierverfahren, die unabhängig von der Anwendung auf einem Computer ist.

Folgende Beobachtung ist hilfreich: Alle Verfahren machen Gebrauch von den elementaren Operationen, die du im Kapitel 1 kennengelernt hast. Sie vergleichen und vertauschen Zahlen. Man kann also die Vergleiche und Vertauschungen zählen. Diese Grösse bestimmt, wie lange ein konkretes Sortierverfahren benötigt und ist nicht vom verwendeten Computer abhängig.

Das Problem der Bewertung ist damit noch nicht vollständig gelöst. Die Anzahl der notwendigen Vergleiche oder Vertauschungen eines Verfahrens hängt stark von der gegebenen Zahlenfolge ab. Es gibt kein Verfahren, das für jede Zahlenfolge optimal ist. Zahlenreihen können derart unterschiedlich sein, dass man sie nicht alle testen kann, um das beste Sortierverfahren zu ermitteln.
Die Gemeinsamkeit der Verfahren besteht darin, dass sie aufwendiger werden, je länger die Zahlenfolge ist. Weil man sich nicht auf eine bestimmte Länge festlegen möchte, benennt man die Länge allgemein mit n. Das n kann für jede beliebige Zahl stehen. Wenn man jetzt ein Verfahren genauer beschreibt, kann man Aussagen machen, wie z. B.: „Dieses Verfahren löst jeden Problemfall der Länge n mit maximal (n × Anzahl n der elementaren Operationen)“.

Wie man in den vorigen Kapiteln sehen konnte, kommt es nicht nur auf die Länge der Folge, sondern auch auf die Anordnung der Zahlen in der Reihe an. Je nach Aufbau der Zahlenreihe ist ein Verfahren mal sehr schnell, mal sehr langsam. Man betrachtet daher für jedes Verfahren seinen individuell schlechtesten Fall (Worst Case). Man nimmt dabei an, dass die aufwendigste Folge sortiert werden soll.

Der Worst Case ist aber nicht immer sehr aussagekräftig, wie man z. B. bei Quick-Sort gesehen hat. Quick-Sort braucht bei den meisten Zahlenfolgen sehr wenige Vergleiche. Im Worst Case benötigt es aber ähnlich viele Vergleiche wie Selection-Sort. Deshalb interessiert man sich auch für den durchschnittlichen Fall (Average Case) und für den besten Fall (Best Case).

Für die meisten Anwendungen ist der Average Case der interessanteste Fall. Im Fokus steht damit, wie schnell das Verfahren im Durchschnitt ist und nicht wie langsam es im schlechtesten Fall ist. Manchmal tritt der schlechteste Fall so selten ein, dass man seine Betrachtung vernachlässigen kann. Es gibt aber viele Anwendungen, bei denen eine Sortierung eine vorgegeben Zeit nicht überschreiten darf. In diesen Fällen muss man wissen, wie ein Sortierverfahren im Worst Case funktioniert.

Weil es bei einzelnen Verfahren sehr schwierig ist, den Best Case und den Average Case zu berechnen, wird darauf hier nicht genauer eingegangen.

5.3 Analyse von Bubble-Sort

Anzahl Vergleiche:
Geh zurück zum ersten Kapitel, ruf dir das Verfahren nochmals schnell in Erinnerung. Schau dir Aufgabe 1.4 b) mit der Lösung an, dort findest du die aufwendigste Folge für das Bubble-Sort Verfahren.

Eine absteigend sortierte Folge, die man in eine aufsteigend sortierte Folge umwandeln möchte, ist für Bubble-Sort am aufwendigsten.

Beispiel: Man betrachtet eine Folge mit n Elementen. Wie lange braucht Bubble-Sort im schlechtesten Fall, um sie zu sortieren?

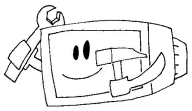
Man zählt die maximale Anzahl der Vergleiche, die in einem Durchgang gemacht werden:
Für n Elemente muss man in einem Durchgang (n - 1) Vergleiche machen.

Um eine sortierte Endfolge zu erreichen, braucht man mehrere Durchgänge.
Wie viele sind es?
Nach jedem Durchgang steht ein Element mehr am endgültigen Platz.
Wenn (n - 1) Elemente an ihrem Platz sind, dann steht auch das letzte Element richtig. Der Computer braucht zusätzlich einen Durchgang, um die richtige Sortierung zu bestätigen. Ingesamt braucht man n Durchgänge.

Zusammenstellung: Anzahl Vergleiche

Anzahl Vergleiche pro Runde:	(n - 1)
Anzahl Runden	n
Laufzeit:	(n - 1) × n

Die beiden Werte werden multipliziert, da man n Durchgänge hat mit (n - 1) Vergleichen. Man muss also n mal (n - 1) Vergleiche machen.



Lernkontrolle:

Aufgabe 5.3 a)
Überlege, wie viele Vergleiche man im besten, also im am wenigst aufwendigen Fall machen muss. Wie sieht die Ausgangsfolge dafür aus?

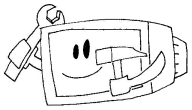
Anzahl Positionsvertauschungen

Neben der Anzahl der Vergleiche kann man auch die Anzahl der Positionsvertauschungen zählen.
Man geht für den Worst Case wieder von der umgekehrt sortierten Reihe aus.

Beispiel:

4	3	2	1	Ausgangszahlenfolge
3	2	1	4	In der ersten Runde werden drei Vertauschungen gemacht: 4 ⇔ 3, 4 ⇔ 2 und 4 ⇔ 1.
2	1	3	4	In der zweiten Runde werden zwei Vertauschungen gemacht: 3 ⇔ 2 und 3 ⇔ 1
1	2	3	4	In der letzten Runde werden noch die ersten beiden Elemente vertauscht: 2 ⇔ 1.

In jeder Runde tauscht man einmal weniger als in der vorherigen Runde. Man startet mit (n - 1) und benötigt daher (n - 1) + (n - 2) + (n - 3) + … + 3 + 2 + 1 Vertauschungen.



Lernkontrolle:

Aufgabe 5.3 b)
Wie viele Vertauschungen benötigt der Best Case?

Aufgabe 5.3 c)
Vervollständige folgende Tabelle mit der Zusammenfassung von Bubble-Sort.

Zusammenfassung von Bubble-Sort:

	Vergleiche	Vertauschungen
Schlechtester Fall:	(n - 1) × n	(n - 1) + (n - 2) +…+ 3 + 2 + 1
Bester Fall:		

5.4 Analyse von Selection-Sort

Zur Erinnerung an das Verfahren lese bei Bedarf Kapitel 2.3. Wenn man eine Zahlenfolge der Länge n hat, benötigt man im ersten Durchgang (n - 1) Vergleiche, um das Minimum zu finden. Jedes Element wird mit dem bisherigen Minimum verglichen. Die Ausnahme bildet die erste Zahl. Sie muss mit keiner anderen verglichen werden, deshalb subtrahiert man 1 und rechnet (n -1). Um das gefundene Minimum an seinen richtigen Platz zu stellen, benötigt man zusätzlich eine Vertauschung.

In der zweiten Runde benötigt man für die Minimumsuche einen Vergleich weniger, da ein Element bereits am richtigen Platz ist. Daher benötigt man in der zweiten Runde (n - 2) Vergleiche. Trotzdem benötigt man auch in der zweiten Runde genau eine Vertauschung. Im Allgemeinen braucht man in jeder Runde einen Vergleich weniger als in der vorhergehenden Runde plus eine Vertauschung.

Wie viele Durchgänge sind notwendig? Nach jeder Runde ist ein Element mehr am richtigen Platz. Hat man n Elemente, braucht man n Durchgänge, damit jedes Element am richtigen Platz ist.

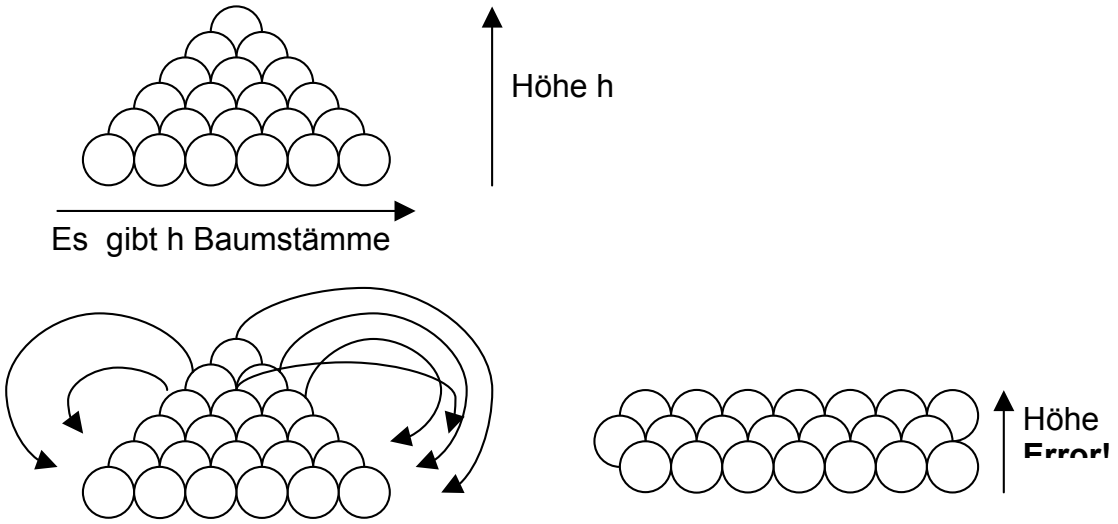
Zusammenfassung: Aufstellung der Runden für den Worst Case

Runde	Anzahl Vergleiche	Anzahl Vertauschungen
1. Runde	n - 1	1
2. Runde	n - 2	1
3. Runde	n - 3	1
.	.	.
.	.	.
.	.	.
Drittletzte Runde (Runde n - 2)	2	1
Zweitletzte Runde (Runde n - 1)	1	1
Letzte Runde (Runde n)	0	1

Die Anzahl Vertauschungen ist einfach zu berechnen: Man hat n Runden und in jeder Runde gibt es eine Vertauschung. Man benötigt also n × 1 = n Vertauschungen.

Wie viele Vergleiche sind notwendig?
Man addiert die Durchgänge:
1 + 2 + 3 + 4 + 5 + 6 + 7 +….+ (n - 3) + (n - 2) + (n - 1)

Umwandlung von 1 + 2 + 3 + … + (n - 3) + (n - 2) + (n - 1)
Zur Veranschaulichung: Die Rechnung ist vergleichbar mit der Addition der Baumstämme im unteren Bild. Wenn auf der Basis h Baumstämme liegen, hat der Stapel genau die Höhe h.



Die Baumstämme könnte man auch umsortieren: Der Oberste kommt neben den Untersten. Die beiden zweitobersten Baumstämme werden zu den zweituntersten Stämmen gelegt, usw. Wenn man so sortiert, ist der Stapel nur noch halb so hoch und in jeder Reihe liegen die gleiche Anzahl Baumstämme. In einer Reihe liegen jetzt genau so viele Baumstämme wie vorher an der Basis lagen plus 1.
Nun ist es leicht, die Anzahl der Baumstämme insgesamt zu berechnen:
Höhe des neuen Stapels (= **Error!**) multipliziert mit der Anzahl Baumstämme pro Reihe (h + 1).
Die Rechnung lautet: **Error!** × (h + 1).
Diese Formel kann man auch anders herleiten.
Wenn man die Zahlen n, n - 1, n - 2 …, 3 , 2 , 1 in einer Zeile notiert und in der Zeile darunter die selben Zahlen in umgekehrter Reihenfolge schreibt, sieht die Tabelle so aus:

	n	n - 1	n - 2	…	…	3	2	1
	1	2	3	…	…	n - 2	n - 1	n
Summe:	n + 1	n + 1	n + 1	n + 1	n + 1	n + 1	n +1	n + 1

Addiert man nun die Spalten lautet die Summe immer (n + 1).
Und da es n Spalten gibt, lautet die dazugehörende Rechnung n × (n + 1).
Weil jede Zahl zweimal aufgeschrieben wurde – einmal in der oberen und einmal in der unteren Zeile – muss man die Summe halbieren.
Man erhält wieder die Formel n × **Error!**.
Zurück zur eigentlichen Aufgabe: Wie viele Vergleiche benötigt man nun? Man hat n Runden. Weil man in der letzten Runde keinen Vergleich durchführt, kann

man diese Runde für die Berechnung vernachlässigen. Es liegen also (n - 1) Runden vor und man beginnt mit (n - 1) Vergleichen.

Das erinnert sehr an die Rechnung mit den Baumstämmen. Für die Höhe h setzt man (n - 1) ein. Folglich ist die Anzahl der Vergleiche zu berechnen mit:
Error! × (n - 1 + 1) = **Error!** × n = n × **Error!**

Lernkontrolle:

Aufgabe 5.4 a)
Du hast nun die Berechnungen für den Worst Case bei Selection-Sort gesehen. Überlege nun, wie die Abschätzung für den Best Case aussieht.

Aufgabe 5.4 b)
Versuche den Average Case abzuschätzen.

Fasse selbst zusammen, wie viele Vergleiche und Vertauschungen Selection-Sort machen muss.

Zusammenfassung: Selection-Sort

	Vergleiche	Vertauschungen
Schlechtester Fall:		
Bester Fall:		
Durchschnittlicher Fall:		

Bei Selection-Sort ist es unwichtig, wie die Zahlenreihe vor dem Verfahren lautet. Für die Bemessung gibt es keine Unterschiede zwischen dem Worst, Best und Average Case.

5.5 Analyse von Insertion-Sort

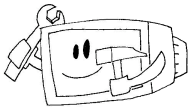
Lese bei Bedarf über dieses Verfahren im Kapitel 2 nach. Welches ist der Worst Case für Insertion-Sort? Bei welcher Zahlenfolge muss es am meisten arbeiten? Am aufwendigsten ist die Sortierung einer absteigenden Zahlenfolge. Überlege kurz, warum das so ist oder schau die Lösung zu Aufgabe 2.3 d) an.
Man geht wieder von einer Zahlenfolge der Länge n aus.

Zusammenfassung: Aufstellung der Runden für den Worst Case

Runde	**Anzahl Vergleiche**	**Anzahl Vertauschungen**
1. Runde	0	0
2. Runde	1	1

3. Runde	2	2
.	.	.
.	.	.
.	.	.
Drittletzte Runde (n - 2 Runde)	n - 3	n - 3
Zweitletzte Runde (n - 1 Runde)	n - 2	n - 2
Letzte Runde (n Runde)	n - 1	n - 1

In jeder Runde sortiert man ein Element mehr in die bereits sortierte Zahlenfolge ein. Weil es n Elemente gibt, benötigt man n Runden, wie auch mit Selection-Sort. In der ersten Runde betrachtet man das vordere Element als sortiert, deswegen benötigt man weder einen Vergleich noch eine Vertauschung. In der zweiten Runde muss man einmal vergleichen und einmal tauschen, da im Worst Case das betrachtete Element immer bis an die erste Stelle getauscht werden muss.



Lernkontrolle:

Aufgabe 5.5 a)
Versuche die Abschätzungen, also die absolute Anzahl von Vergleichen und Vertauschungen, für den Worst Case von Insertion-Sort selbstständig zu ermitteln. Orientiere dich dabei an der Analyse von Selection-Sort.

Aufgabe 5.5 b)
Fülle diese Aufstellung aus.

Zusammenfassung: Aufstellung der Runden für den Best Case

Runde	**Anzahl Vergleiche**	**Anzahl Vertauschungen**
1. Runde:	…….	…….
2. Runde:	…….	…….
3. Runde:	…….	…….
.	.	.
.	.	.
.	.	.
Drittletzte Runde (n - 2 Runde)	…….	…….
Zweitletzte Runde (n - 1 Runde)	…….	…….
Letzte Runde (n Runde)	…….	…….

Aufgabe 5.5 c)
Die Aufstellung für den Best Case hast du jetzt ausgefüllt. Kannst du auch die anderen Abschätzung selbstständig berechnen?

Fasse zusammen, wie viele Vergleiche und Vertauschungen Insertion-Sort machen muss.

Zusammenfassung: Insertion-Sort

	Vergleiche	Vertauschungen
Schlechtester Fall:		
Bester Fall:		
Durchschnittlicher Fall:		

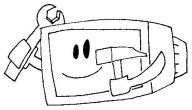
5.6 Analyse von Merge-Sort

Die Analyse von Merge-Sort mit dem Ziel, eine allgemeingültigen Formel zu ermitteln, ist kompliziert. In diesem Abschnitt wird deshalb der Best und Worst Case anhand einer Reihe mit acht Elementen, den Zahlen von 1 bis 8, bestimmt.

Wie man in Kapitel 3 gesehen hat, kann man beim Merge-Sort Verfahren einige Vergleiche sparen, wenn eine Teilreihe vor der anderen in die grosse Reihe geschrieben wird. Sprich: Wenn beim Verbinden (merge) eine der beiden Teilreihen schon leer ist, während die andere noch mehr als ein Element beinhaltet.

Wenn Unklarheiten mit diesen Aussagen bestehen, lese Kapitel 3.

Best Case:
Am meisten Vergleiche können gespart werden, wenn man beim Aufteilen in Schritt 1 die hohen von den niedrigen Zahlen trennt. Das bedeutet, dass die schon sortierte Reihe (1, 2, 3, 4, 5, 6, 7, 8) am wenigsten Vergleiche benötigt.



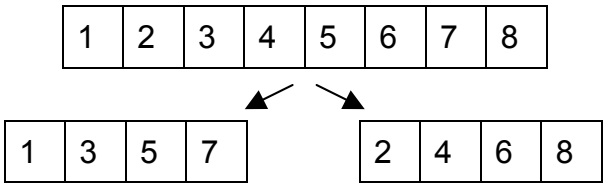
Lernkontrolle:

Aufgabe 5.6 a)
Zähle, wie viele Vergleiche man für den Best Case mit acht Einträgen machen muss. Gehe von der Reihe(1, 2, 3, 4, 5, 6, 7, 8) aus.

Worst Case:
Bei der Analyse des Worst Case untersucht man den Fall, der immer alle Vergleiche zwingend benötigt. Er ist das Gegenteil des Best Case. Welches ist die aufwendigste Reihe für die Sortierung im Worst Case?

Man teilt die Reihe nicht wie im Best Case in grosse und kleine Zahlen auf, sondern verteilt hohe und niedrige Zahlen auf beide Teilreihen. Wenn also jede zweite Zahl in der einen Hälfte und die restlichen Zahlen in der anderen Hälfte stehen, dann müssen am meisten Vergleiche gemacht werden.

Jede zweite Zahl in der gleichen Reihe:



Die Aufteilung der beiden Teile erfolgt nach dem gleichen Prinzip. So werden beim Zusammenfügen möglichst viele Vergleiche gemacht.

Eine Reihe, die sicher alle Vergleiche benötigt, ist dementsprechend:

1 5 3 7 2 6 4 8

Es gibt noch viele andere Beispiele.

Anzahl der Vergleiche im Worst Case für die Reihe (1, 5, 3, 7, 2, 6, 4, 8):

Zeichenerklärung:

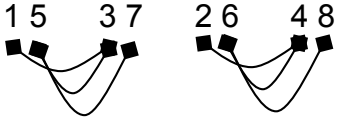
Dieses Zeichen signalisiert einen Vergleich zwischen den darüber stehenden Zahlen.

Man beginnt beim ersten Zusammenführen, also auf der untersten Stufe, dort wo die einzelnen Zahlen in Teilreihen stehen.

In der untersten Stufe:

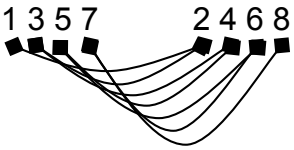
→ 4 Vergleiche

Zweitunterste Stufe:



→ 6 Vergleiche

Oberste Stufe:



→ 7 Vergleiche

Insgesamt braucht man für den Worst Case 4 + 6 + 7 = 17 Vergleiche.

Zusammenfassung: Merge-Sort
Best Case: 12 Vergleiche für n = 8
Worst Case: 17 Vergleiche für n = 8

Vergleich mit anderen Verfahren
Gegenüberstellung der Anzahl notwendiger Vergleiche der bisher vorgestellten Sortierverfahren für acht Elemente:

Sortierverfahren	**Vergleiche im Best Case**		**Vergleiche im Worst Case**	
Bubble-Sort	n - 1 =	7	(n - 1) × n =	56
Insertion-Sort	n - 1 =	7	n × **Error!** =	28
Selection-Sort	n × **Error!** =	28	n × **Error!** =	28
Merge-Sort		12		17

Anhand der Tabelle wird deutlich, dass Bubble-Sort und Insertion-Sort bei schon sortierten Folgen, dem Best Case für die beiden Verfahren, am effektivsten ist. Merge-Sort ist dafür im schlechtesten Fall besser als die anderen drei Sortierverfahren.

Die Tabelle oben ist nicht repräsentativ. Sie stellt nur die Anzahl der benötigten Vergleiche gegenüber. Die Anzahl der Vertauschungen sowie der Speicherbedarf der einzelnen Verfahren bleiben wegen der Komplexität in diesem Kapitel unberücksichtigt.

Vorteile der Teile-und-Herrsche-Verfahren:
Weil hier nur Reihen mit acht Elementen untersucht wurden, werden die Vorteile eines Teile-und-Herrsche-Verfahrens nicht so deutlich, als würde man grössere Reihen analysieren. Folgende Überlegung verdeutlicht jedoch die Schnelligkeit dieses Verfahrens.

Merge-Sort mit Bubble-Sort:
Betrachtet wird eine Reihe mit n Elementen. Man führt ein abgewandeltes Merge-Sort für diese Reihe durch, bei dem man den zweiten Schritt von Merge-Sort durch Bubble-Sort ersetzt.

Abgewandeltes Merge-Sort:

1. Teile
2. Sortiere beide Teile einzeln mit Bubble-Sort
3. Vereinige

Die Reihe wird in zwei gleich grosse Reihen aufgeteilt. Führe nun ein Bubble-Sort mit beiden Teilreihen aus. Sie bestehen aus **Error!** Elementen.

Bubble-Sort im schlechtesten Fall für eine Reihe mit **Error!**Elementen:
Error! × (**Error!**- 1)

Da es zwei solcher Reihen gibt, braucht man 2 × **Error!** × (**Error!** - 1) Vergleiche, um beide Teilreihen separat zu sortieren. Fügt man diese Teile im dritten Schritt für den Worst Case zusammen, braucht man nochmals (n - 1) Vergleiche. Das ergibt insgesamt
2 × **Error!** × (**Error!** - 1) + (n - 1) Vergleiche.

Umwandlung:
2 × **Error!** × (**Error!**- 1) + (n - 1) = n × (**Error!** - 1) + (n - 1)
= **Error!** - n + n - 1
= **Error!** - 1
= (n × **Error!**) - 1.

Dieses Ergebnis entspricht ungefähr dem Aufwand mit Insertion-Sort oder Selection-Sort. Man hat im vorliegenden Fall nur einmal geteilt. Teilt man häufiger, dann verringert sich die Anzahl der Vergleiche weiter. Diese Eigenschaft gilt für alle Teile-und-Herrsche Verfahren und somit auch für Quick-Sort.

Allgemein kann man sagen, dass Merge-Sort einerseits sehr wenige Vergleiche braucht, andererseits aber auch sehr viel Speicherplatz, da es sich die Teilreihen merken muss.

5.7 Analyse von Quick-Sort

Zusammenfassung: Aufstellung der Runden für den Worst Case

Runde	Anzahl Vergleiche	Anzahl Vertauschungen
1. Runde	n -1 + 2 = n + 1	1
2. Runde	n - 2 + 2 = n	1
3. Runde	n - 3 + 2 = n - 1	1
.	.	.
.	.	.
.	.	.
Drittletzte Runde (n - 2 Runde)	2 + 2 = 4	1
Zweitletzte Runde (n - 1 Runde)	1 + 2 = 3	1
Letzte Runde (n Runde)	0	1

In der ersten Runde benötigt man (n - 1) Vergleiche, um alle Elemente mit dem Pivot-Element zu vergleichen. Zwei Vergleiche kommen hinzu, weil sich die Pfeile überkreuzen müssen.
Hätte man zusätzlich noch eine Runde mit zwei und eine Runde mit einem Vergleich, so erhielte man eine Abschätzung von
(n + 1) × **Error!**
Hiervon muss man noch die drei Vergleiche subtrahieren, die man nicht hat.
Das ergibt also (n + 1) × **Error!** - 3.

Der Best Case und der Average Case sind sehr kompliziert, deshalb werden hier nur Überlegungen dazu angestellt. Wenn man eine grosse Zahlenreihe mit z. B. 2000 Einträgen hat, in der Informatik ist das eher wenig, dann gibt es genau zwei schlechteste Fälle für die Pivotwahl: Wenn man das grösste oder das kleinste Element auswählt.
Die Wahrscheinlichkeit dafür ist gering, denn es gibt 1998 bessere Elemente. Es ist von Vorteil, wenn man die Zahlenreihe in zwei Teilreihen mit der Grösse 1399 und 600 Elemente aufteilt. Dann muss man statt 1999 Vergleiche im nächsten Schritt nur 1398 Vergleiche durchführen. Es gibt also mehr Fälle, bei denen man von Quick-Sort profitiert als solche, bei denen Quick-Sort nicht geeignet ist. Wählt man einmal einen schlechten Fall, ist das nicht so gravierend, denn im nächsten Durchgang wählt man wahrscheinlich nicht noch einmal ein schlechtes Pivot.

Im Average Case ist Quick-Sort sehr gut geeignet. Daher ist es eines der meistgenutzten Sortierverfahren.

Lernkontrolle:

Aufgabe 5.7 a)
Wieso kommt der Vorteil von Quick-Sort nur bei langen Folgen zum Tragen?

Zusammenfassung Quick-Sort

Bei Quick-Sort gilt in den vielen Fällen: Nomen est Omen. Es ist ein sehr schnelles Verfahren, welches keinen zusätzlichen Speicher benötigt. Für bereits geordnete Folgen und bei kurzen Folgen ist Quick-Sort ungeeignet, weil es bei diesen Bedingungen nicht von kleineren Teilreihen profitieren kann. Trotzdem ist es im schlechtesten Fall nicht viel schlechter als die bisher vorgestellten Sortierverfahren. Daher wird Quick-Sort oft angewendet.

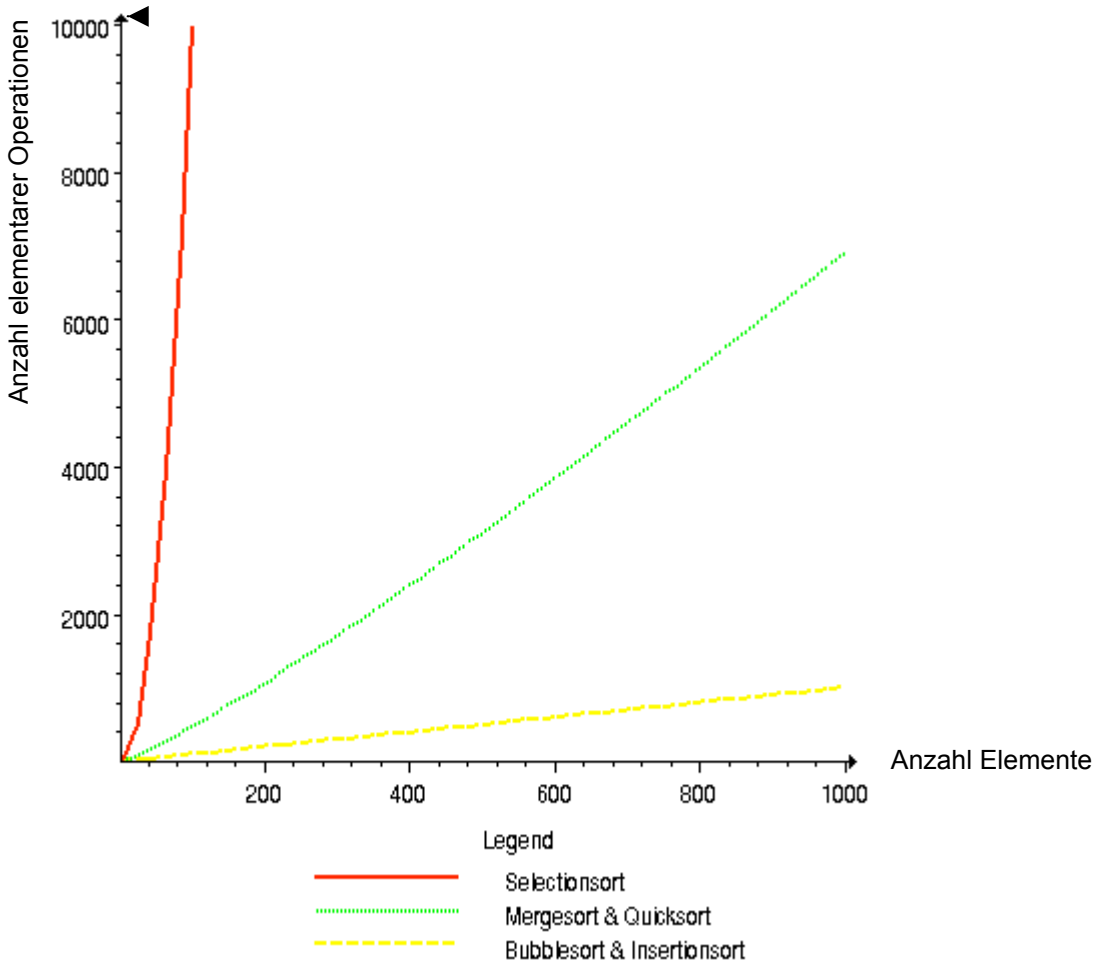
5.8 Grafischer Vergleich der verschiedenen Sortierverfahren

In diesem Leitprogramm wurden einige Betrachtungen ausgelassen, weil sie zu aufwendig gewesen wären. Trotzdem sollst du einen Eindruck bekommen, wie sich die vorgestellten Verfahren im Best, Average und Worst Case verhalten, dazu ist jeweils eine Grafik erstellt worden.

Diese Grafiken stellen ungefähr den Zusammenhang zwischen der Anzahl der Elemente und der Anzahl der benötigten elementaren Operationen dar. Kleine Unterschiede werden nicht beachtet. Es wird deutlich, wie schnell die Anzahl der benötigten elementaren Operationen wächst.

Beachte, dass für die x-Achse und die y-Achse nicht die gleiche Einheit verwendet wurde.

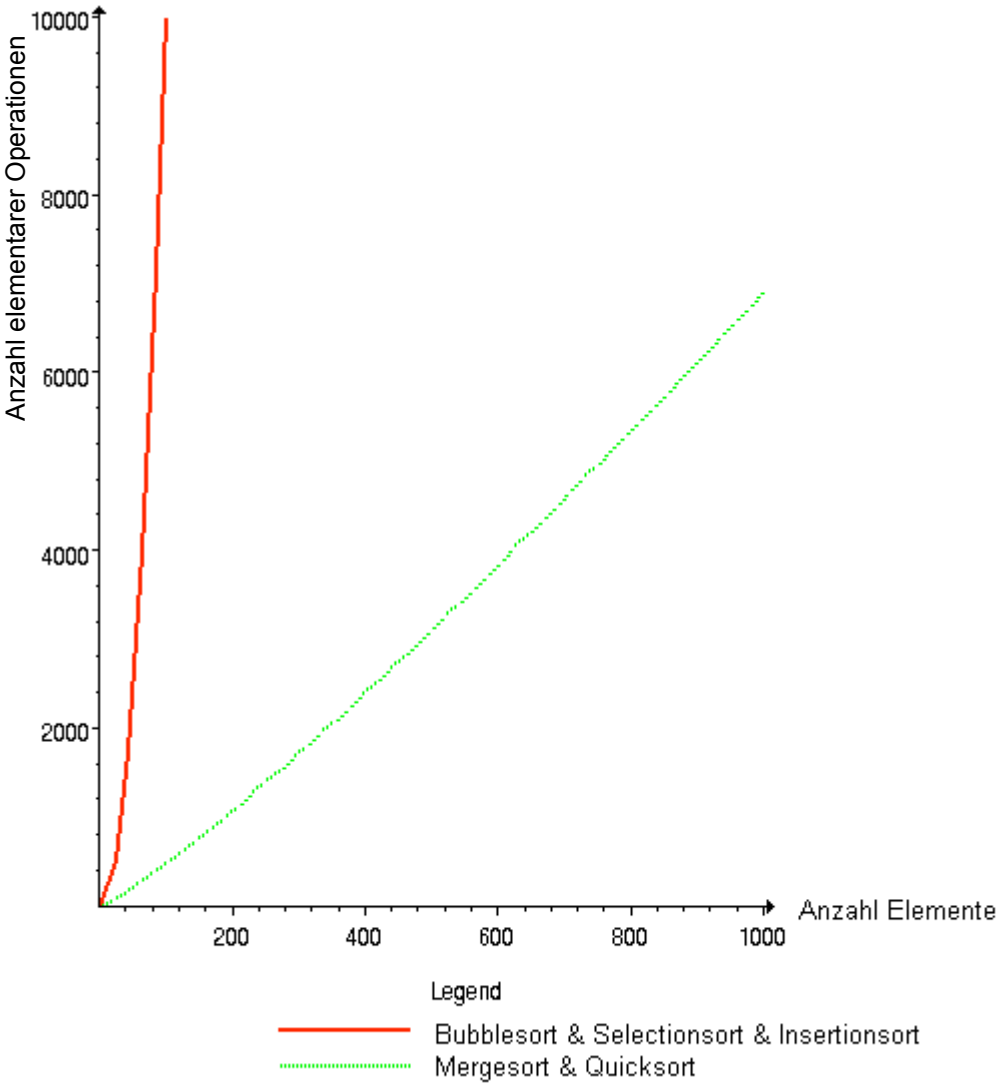
Grafik für den Best Case



Beim Best Case sind die Unterschiede am grössten. Bubble-Sort und Insertion-Sort schneiden mit Abstand am besten ab. Sie brauchen nur etwa so viele elementare Operationen wie Elemente, die sie sortieren müssen.

Selection-Sort hingegen benötigt sehr schnell sehr viele elementare Operationen. Merge-Sort und Quick-Sort benötigen zwar viel mehr elementare Operationen als Bubble-Sort und Insertion-Sort, aber auch deutlich weniger als Selection-Sort.

Grafik für den Average Case:

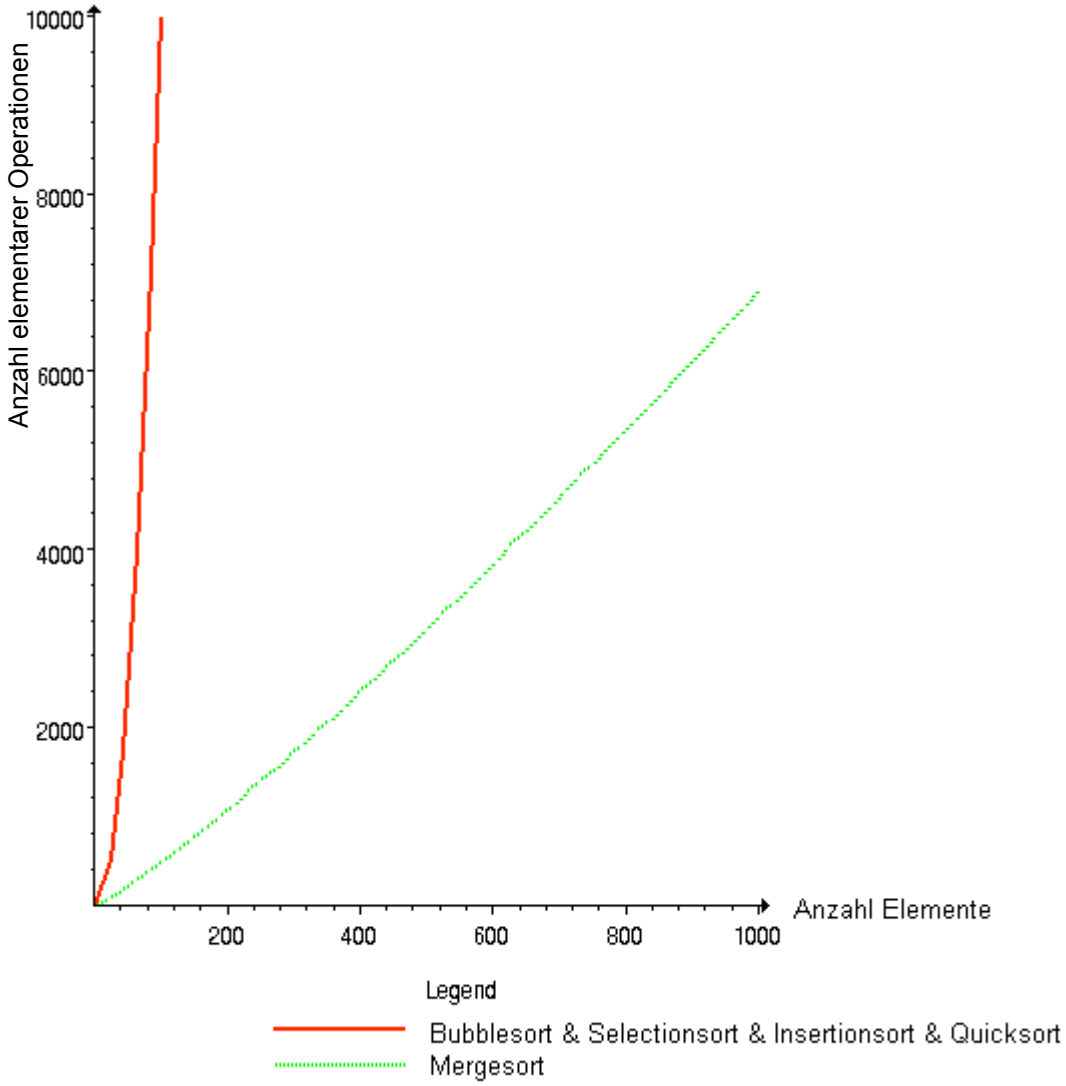


Im Average Case erkennt man, dass sowohl Quick-Sort als auch Merge-Sort deutlich weniger Vergleiche benötigen als die übrigen vorgestellten Sortierverfahren. Der Average Case ist der wichtigste Fall für die Vergleichsmessung, da man nicht auf den Best Case hoffen sollte.

Hat man zufällige Zahlenfolgen zu sortieren, mögen Bubble-Sort und Insertion-Sort im Best Case besser geeignet sein als Merge-Sort und Quick-Sort. Merge-Sort und Quick-Sort sind im Durchschnitt trotzdem viel besser.

In dieser Grafik wird nicht deutlich, dass Merge-Sort trotz seiner Schnelligkeit mehr Speicherplatz benötigt.

Grafik für den Worst Case



Im Worst Case sind alle Verfahren gleich schlecht – mit Ausnahme von Merge-Sort. Sogar Quick-Sort scheint nicht mehr so quick zu sein. Aber wie schon erwähnt, kommt bei Quick-Sort der Worst Case nicht so häufig vor.

Ist es jedoch Bedingung, dass ein Verfahren eine bestimmte Anzahl Vergleiche nicht überschreitet, kommt für die Sortierung nur Merge-Sort in Frage.

5.9 Zusammenfassung

Jedes Sortierverfahren beruht auf einer unterschiedlichen Grundidee und hat somit seine individuellen Stärken und Schwächen. Sind diese bekannt, kann man das bestgeeignete Verfahren verwenden, um ein Problem zu lösen.

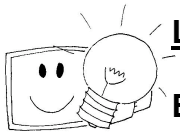
Wenn man schon eine sortierte Reihe hat und ein Element einfügen will, dann verwendet man idealerweise Bubble-Sort oder Insertion-Sort. Merge-Sort und Quick-Sort sind in diesem Fall ungeeignet.

Merge-Sort und Quick-Sort sind sehr schnelle Verfahren. Merge-Sort hat gegenüber Quick-Sort den Vorteil, die Zahlenreihe immer in zwei gleich grosse Teilstücke zu teilen. Quick-Sort macht eine ungünstige Folge wenig aus und benötigt nur einen zusätzlichen Speicherplatz. Quick-Sort braucht also nicht mehr Speicherplatz als Selection-Sort oder Bubble-Sort und ist trotzdem in den meisten Fällen sehr schnell.

Bei der Wahl der Sortierverfahren sollte man auch beachten, wie gross die zu sortierenden Datenreihen sind. Wenn diese gross sind und viele Vertauschungen notwendig werden, dauert es sehr lange. In diesem Fall wählt man besser ein Verfahren wie Selection-Sort. Es benötigt zwar zahlreiche Vergleiche, aber nur einen Tausch pro Element, denn das jeweilige Minimum wird direkt an seinen richtigen Platz getauscht.

5.10 Schlusswort

Das Leitprogramm ist nun beendet. Die Autorinnen hoffen, dass es dir gefallen hat und du eine kurzweilige und interessante Zeit hattest.



Lösungen zum Additum

Bubble-Sort

Lösung 5.3 a)
Für den besten Fall von Bubble-Sort betrachtet man die schon sortierte Folge. Man muss alle benachbarten Elemente einmal miteinander vergleichen und bemerkt dann, dass man schon fertig ist. Folglich benötigt man (n - 1) Vergleiche.

Lösung 5.3 b)
Im besten Fall ist keine Vertauschung nötig, da die Elemente schon an ihren endgültigen Positionen stehen.

Lösung 5.3 c)

	Vergleiche	Vertauschungen
Schlechtester Fall:	(n - 1) × n	(n - 1) + (n - 2) +…+ 3 + 2 + 1
Bester Fall:	(n - 1)	keine

Selection-Sort

Lösung 5.4 a)
Man geht im Best Case genau so vor wie im Worst Case: Man sucht jeweils das Minimum und tauscht es an die richtige Stelle. Die Abschätzungen sind deshalb für beide Fälle gleich: Man braucht n × **Error!** Vergleiche und n Vertauschungen.

Lösung 5.4 b)
Weil mit Selection-Sort jede Zahlenfolge genau gleich bearbeitet wird, hat man die gleiche Abschätzung wie im besten und im schlechtesten Fall:
n × **Error!** Vergleiche und n Vertauschungen.

Insertion-Sort

Lösung 5.5 a)
Es gibt n Runden. Da aber in der ersten Runde nichts passiert, braucht man diese Runde für die Abschätzung nicht beachten und geht von (n - 1) Runden aus. In jeder Runde gibt es einen Vergleich mehr als in der vorherigen. Es ist unwesentlich, ob man
1 + 2 + 3 +…+ (n - 3) + (n - 2) + (n - 1) oder
(n - 1) + (n - 2) + (n - 3) +…+ 3 + 2 + 1 rechnet.
Die Anzahl der Vergleiche entspricht denen von Selection-Sort: n × **Error!**
Man vertauscht ebenso häufig wie man vergleicht, deshalb ist auch die Anzahl Vertauschungen n × **Error!**

Lösung 5.5 b)

Runde	Anzahl Vergleiche	Anzahl Vertauschungen
1. Runde:	0	0
2. Runde:	1	0
3. Runde:	1	0
.	.	.
.	.	.
.	.	.
Drittletzte Runde (n - 2 Runde)	1	0
Zweitletzte Runde (n - 1 Runde)	1	0
Letzte Runde (n Runde)	1	0

Bei Insertion-Sort ist der Best Case die bereits sortierte Folge. In der ersten Runde betrachtet man ein einzelnes Element. Deswegen gibt es in der ersten Runde keinen Vergleich. Weil man in den restlichen Runden noch nicht weiss, ob das Element schon am richtigen Platz ist, benötigen man in jeder Runde genau einen Vergleich. Tauschen muss man selbstverständlich nicht.

Lösung 5.5 c)
Man benötigt n Runden bis jedes Element seinen Platz gefunden hat. In der ersten Runde braucht man keinen Vergleich, da man nur ein einzelnes Element betrachtet. Es sind daher (n - 1) × 1 = n - 1 Vergleiche. Bei den Vertauschungen ist es noch einfacher: Man braucht keine.

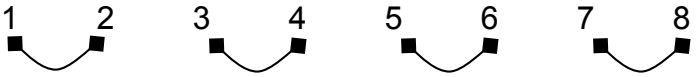
Merge-Sort

Lösung 5.6 a)
Zeichenerklärung:

Dieses Zeichen signalisiert einen Vergleich zwischen zwei Zahlen.

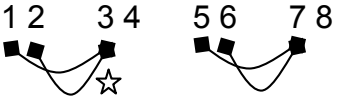
Man beginnt beim ersten Zusammenführen, der untersten Stufe, dort wo einzelne Zahlen die Teilreihen bilden:

In der untersten Stufe:



→ 4 Vergleiche

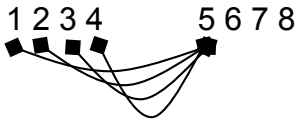
Zweitunterste Stufe:



→ 4 Vergleiche

☆ Hier vergleichet man zuerst die 1 mit 3. 1 ist kleiner und wird in die grössere Reihe geschrieben. Dann vergleichet man 2 mit 3 und schreibt die 2 in die grössere Reihe. Die erste Teilreihe ist nun leer und die 3 und die 4 können ohne weitere Vergleiche in die grössere Reihe geschrieben werden.

Oberste Stufe:



→ 4 Vergleiche

Insgesamt braucht man im Best Case 4 + 4 + 4 = **12 Vergleiche**.

Es gibt verschiedene Reihen, die auch 12 Vergleiche benötigen, z. B. (5, 6, 7, 8, 1, 2, 3, 4) oder (3, 4, 1, 2, 5, 6, 7, 8) und noch einige mehr.

Quick-Sort

Lösung 5.7 a)

Die Wahrscheinlichkeit, dass man ein ungünstiges Pivot-Element wählt, ist bei kürzeren Zahlenfolgen grösser als bei langen. In jeder Zahlenfolge gibt es immer ein grösstes und ein kleinstes Element.

Betrachtet man eine Zahlenfolge mit acht Elementen, trifft man mit zwei von acht Elementen, also bei einem Viertel der Elemente, den Worst Case. Bei einer Zahlenfolge mit 2000 Elementen tritt der Worst Case bei zwei von 2000 Elementen ein, also bei einem Tausendstel der Elemente.

6. Grundbegriffe

Abbruchbedingung
Bei einem rekursiven Programm definiert die Abbruchbedingung das Ende eines Selbstaufrufs. Wenn man die Abbruchbedingung nicht definiert, ruft sich das Programm endlos selbst auf und gelangt niemals zum nächsten Schritt.

Durchgang
Um ein Verfahren besser darzustellen, teilt man es in verschiedene Durchgänge auf. Ein Durchgang ist abgeschlossen, wenn alle Elemente einmal angeschaut und ggf. verschoben wurden. Die Reihe ist meist nach dem ersten Durchgang noch nicht sortiert, deshalb muss man mehrere Durchgänge nacheinander ausführen. Wenn ein Verfahren folglich mehrfach die ganze Reihe bearbeitet und dabei alle Elemente analysiert, vergleicht und ggf. verschiebt, spricht man von verschiedenen Durchgängen.

Elementare Operationen
Das sind Tätigkeiten, die man nicht mehr weiter aufteilen kann. Im beschriebenen Zusammenhang sind es Vergleichsoperationen und Vertauschungen.

Elementares Problem
Ein Problem, das man nicht weiter aufteilen kann oder will. In Teile-und-Herrsche Verfahren ist das elementare Problem die Abbruchbedingung für das Teilen der Zahlenfolge. Falls ein elementares Problem vorliegt, kann es ohne erneuten rekursiven Aufruf gelöst werden. Danach kann das Verfahren dort seine Arbeit fortsetzen, wo es zuletzt geteilt hat. Man geht damit wieder eine Rekursionsstufe nach oben.

Problem
Die Gesamtheit der Problemfälle, die man einem grossen, übergeordneten Problem zuordnen kann.

Wenn man alle Fälle lösen möchte, in denen man drei Elemente zu sortieren hat, kann man nicht sagen, dass man die Zahl x nach vorne stellt. Bei der Lösung eines Problems müssen alle denkbaren Reihen mit drei Elementen mit einem Verfahren lösbar sein. Es ist also die Verallgemeinerung einer konkreten Aufgabe auf alle Fälle, die auftreten können.

Eine Möglichkeit zur Lösung des Problems ist z. B.: Suche das Minimum, schreibe es an die erste Stelle. Suche das Maximum und verschiebe es an die letzte Stelle. Mit diesem Verfahren werden alle Reihen mit drei Elementen sortiert – unabhängig von der Ausgangsreihenfolge.

Problemfall
Eine genau bestimmte Aufgabe, die man lösen muss.
Wenn die Reihe (3, 5, 2) gegeben ist, setzt man die 2 an die erste Position und hat diesen konkreten Problemfall gelöst.

Schritt
Durchgänge von Verfahren können in verschiedene Schritte geteilt werden. Die einzelnen Schritte werden nacheinander ausgeführt. Mehrere Schritte ergeben einen Durchgang oder ein Verfahren.

Rekursion
Rekursion bedeutet, dass etwas auf sich selbst verweist oder sich selbst aufruft. Im vorliegenden Fall drückt es die Anwendung des immer gleichen Verfahrens aus. Nach jedem Aufruf wird der zu lösende Problemfall durch die Aufteilung kleiner und somit besser und schneller lösbar.

Verfahren
Ein Verfahren ist eine Strategie, um ein bestimmtes Problem zu lösen. Es wird häufig durch die Aneinanderreihung von elementaren Operationen dargestellt. Man führt wiederholt Vergleiche zwischen zwei Elementen durch, die nach Regeln verschoben werden.

Man betrachtet dabei mehrfach die ganze Zahlenfolge und wendet immer wieder die gleichen elementaren Operationen an. Im vorliegenden Fall ist das Problem gelöst, wenn die Folge sortiert ist.

Anhang A: Kapiteltests

Kapiteltest 1

Aufgabe 1 (schriftlich nicht möglich)
Die Lehrperson hält vier Spielkarten bereit. Sie legt die Karten ungekehrt auf den Tisch und lässt den Schüler den Algorithmus Bubble-Sort anhand dieser Karten durchspielen.

Aufgabe 2
Wie könnte man das Verfahren Bubble-Sort verbessern oder beschleunigen?

Aufgabe 3
Wenn man eine sortierte Zahlenreihe hat und ein neues Element hinzufügen möchte, würdest du es vorne oder hinten anhängen? Warum?

Aufgabe 4
Was passiert, wenn man mit dem Verfahren von hinten beginnt, also zuerst die letzten beiden Elemente miteinander vergleicht, dann die Zweitletzten, usw.? Welche Unterschiede kannst du feststellen?

Lösungen zu Kapiteltest 1

Lösung 1
Der Schüler/die Schülerin muss immer zwei Karten aufdecken, miteinander vergleichen, ggf. vertauschen und die Karten wieder umdrehen. Den ganzen Algorithmus durchspielen lassen.

Lösung 2
Es gibt mehrere richtige Antworten:

- Man macht maximal (n – 1) Schritte, wobei n die Anzahl der Elemente ist. Damit kann man sich den letzten Durchgang eventuell sparen.
- Da im ersten Durchgang das grösste Element auf seinen definitiven Platz kommt, beim zweiten Durchgang das Zweitgrösste usw., kann man in jedem Durchgang einen Vergleich sparen:
  D.h. im ersten Durchgang alle Elemente anschauen/vergleichen, im zweiten Durchgang nur noch die ersten (n – 1) Elemente vergleichen, im dritten Durchgang (n - 2) Elemente vergleichen, usw.

Lösung 3
Vorne!
Wenn man das Element **vorne** anhängt, dann braucht man maximal einen Durchgang, bis es an seiner endgültigen Stelle steht und die Zahlenreihe somit wieder sortiert ist.
Wenn man es **hinten** anhängt, braucht man im schlechtesten Fall n Durchgänge, bis die Reihe wieder sortiert ist. Wobei n die Anzahl der Elemente ist. Das ist der Fall, wenn das neue Element das Kleinste der Folge ist.

Lösung 4
Lässt man den Algorithmus von hinten nach vorne laufen, dann schiebt man das jeweils kleinere Element um eine Position nach vorne. Damit steht das kleinste Element nach dem ersten Durchgang vorn. Im anderen Fall ist nach dem ersten Durchgang das grösste Element hinten. Sonst gibt es keine Unterschiede.

Kapiteltest 2

Aufgabe 1
Wie viele Vergleiche benötigt man, um in folgender Zahlenreihe das Minimum zu finden?

8 41 25 36 1512 12 46

Aufgabe 2
Erkläre das Verfahren „Selection-Sort“ in eigenen Worten.

Aufgabe 3 (schriftlich nicht möglich)
Die Lehrperson hält für den Schüler/die Schülerin sieben Karten in der Hand und bittet darum, Insertion-Sort zu demonstrieren.

Aufgabe 4
Du hast vier Karten und sollst sie mit Insertion-Sort sortieren. Wie viele Vergleiche brauchst du maximal, bis alle sortiert sind?
Wie oft musst du Karten maximal verschieben?

Lösungen zu Kapiteltest 2

Lösung 1
Um das Minimum in einer Reihe mit sieben Zahlen zu finden, benötigt man
7 - 1 = 6 Vergleiche

Lösung 2
Eine Variante ist: Man wählt das kleinste Element aus und setzt es an den richtigen Platz. Dann sucht man das zweitkleinste Element und setzt auch dieses an seinen Platz, usw. So ist der Anfang immer sortiert. Macht man es so oft, wie es Elemente gibt, ist die Zahlenfolge sortiert.

Lösung 3
Der Schüler/ die Schülerin sollte die Karten in der Hand behalten und immer die erste Karte, die noch nicht sortiert ist, in den sortierten Anfang einfügen.

Lösung 4

1. Durchgang: je maximal	0 mal vergleichen	/	0 mal verschieben
2. Durchgang: je maximal	1 mal vergleichen	/	1 mal verschieben
3. Durchgang: je maximal	2 mal vergleichen	/	2 mal verschieben
4. Durchgang: je maximal	3 mal vergleichen	/	3 mal verschieben

Das ergibt zusammen maximal sechs Vergleiche und maximal sechs Verschiebungen.

Kapiteltest 3

Aufgabe 1
Was ist ein elementares Problem und wieso braucht man es?

Aufgabe 2
Erkläre das Teile-und-Herrsche-Konzept mit eigenen Worten.

Aufgabe 3 (schriftlich nicht möglich)
Die Lehrperson hält vier bis fünf Spielkarten bereit, die sie umgekehrt auf den Tisch legt. Der Algorithmus Merge-Sort soll anhand dieser Karten durchgespielt werden.

Aufgabe 4
In den Aufgaben hast du gesehen, dass die sortierte Folge am wenigsten Vergleiche benötigt. Warum braucht sie nur so wenige? Kennst du noch eine andere Folge, die auch so wenige Vergleiche benötigt?

Lösungen zu Kapiteltest 3

Lösung 1
Ein Problem, das man nicht weiter aufteilen kann oder will. In Teile-und-Herrsche-Verfahren ist das elementare Problem die Abbruchbedingung für das Teilen der Zahlenfolge. Falls ein elementares Problem vorliegt, kann es ohne erneuten rekursiven Aufruf gelöst werden. Danach kann das Verfahren dort seine Arbeit fortsetzen, wo es zuletzt geteilt hat. Man geht damit eine Rekursionsstufe nach oben.

Man braucht elementare Probleme als Abbruchbedingung. Wenn man sie nicht hätte, dann würde der Computer nicht wissen, wann er mit dem Teilen aufhören kann.

Lösung 2
Ein grosses Problem in kleinere Teilprobleme aufteilen, diese lösen und zu einer Gesamtlösung zusammenfügen.

Lösung 3
Der Schüler/die Schülerin muss die Karten zuerst in zwei Teile aufteilen, diese dann jeweils weiter aufteilen und anschliessend sortieren. Die sortierten Teile werden danach wieder zu einem Ganzen zusammengefügt.

Lösung 4
Die sortierte Folge benötigt so wenige Vergleiche, weil beim Zusammenführen immer zuerst die erste Teilreihe in die grosse Reihe geschrieben wird und danach erst die zweite Teilreihe. Wenn eine Teilreihe leer ist, kann man die andere Teilreihe ohne weitere Vergleiche in die grosse Reihe schreiben.

Eine andere Reihe, die ebenso viele Vergleiche braucht, wäre zum Beispiel:
5 6 7 8 1 2 3 4.

Hier wird der zweite Teil zuerst in die grosse Reihe geschrieben und dann erst der erste Teil. Die Anzahl der Vergleiche ist somit gleich hoch.

Andere Beispiele: (3, 4, 1, 2, 5, 6, 7, 8), (1, 2, 3, 4, 7, 8, 5, 6), ...

Kapiteltest 4

Aufgabe 1
Sortiere folgende Reihe mit Quick-Sort:

27 14 29 53 128 10

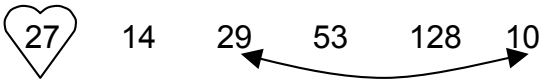
Aufgabe 2
Braucht Quick-Sort mehr oder weniger Vergleiche als Bubble-Sort?

Aufgabe 3
Wie viele Vergleiche braucht man im besten Fall bei elf Elementen?
Wie oft muss man tauschen?

Aufgabe 4
Welche Versionen von Quick-Sort kennst du?

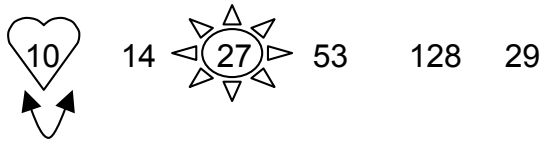
Lösungen zu Kapiteltest 4

Lösung 1



Die 27 wird zum Pivot-Element und die 29 tauscht mit der 10 den Platz.

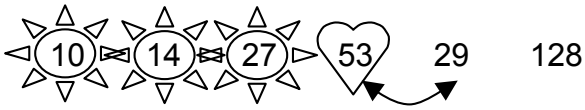
Die 27 wird mit der 10 vertauscht und kommt somit an ihren endgültigen Platz.



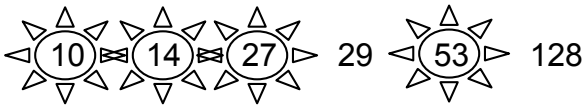
Die 10 wird das neue Pivot-Element. Da sie das kleinste Element der linken Seite ist, wird sie mit sich selbst vertauscht.

Die neu entstandene leere Folge und die Folge mit dem einzelnen Element 14 sind bereits sortiert. Man kann auf der rechte Seite fortfahren.

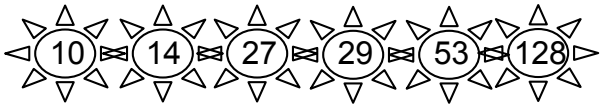
Die 53 wird zum neuen Pivot-Element. Die 128 und die 29 tauschen die Plätze.



Nun wird die 53 an ihren Platz getauscht.



Die neu entstandenen Teile enthalten je ein Element, sind also bereits sortiert.



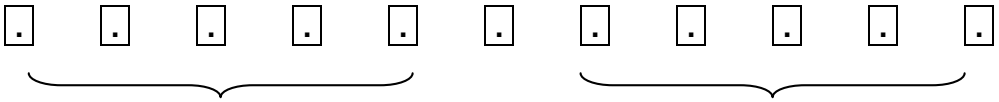
Und somit ist die Aufgabe fertig.

Lösung 2

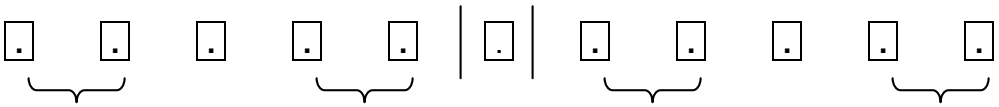
Diese Frage kann nicht allgemein beantwortet werden. Je nach Zahlenfolge, die sortiert wird, kann Quick-Sort sehr schlecht geeignet sein. Bei der sortierten Zahlenfolge reicht Bubble-Sort ein einziger Durchgang mit (n - 1) Vergleichen, während Quick-Sort (n + 2) × **Error!** - 3 Vergleiche benötigt.

In einem Idealfall für Quick-Sort wie der Aufgabe 4.3 a), ist Quick-Sort aber weit besser. In der Regel ist Quick-Sort besser als Bubble-Sort.

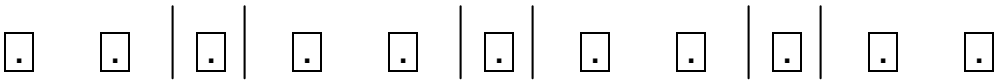
Lösung 3
Im besten Fall mit elf Elementen:



Im Idealfall teilt das Pivot-Element die Zahlenfolge in zwei gleich grosse Teile. Im ersten Schritt sind es 10 + 2 Vergleiche und eine einzige Tauschoperation, damit das Pivot an den richtigen Platz kommt. Die anderen Elemente sind im Idealfall schon an der richtigen Seite.



Im Idealfall wird auch im zweiten Schritt die Zahlenfolge in genau zwei Teile auf beiden Seiten geteilt. Das geschieht zwar nacheinander, spielt aber für diese Betrachtung keine Rolle. Pro Seite sind 4 + 2 Vergleiche notwendig und auf jeder Seite wird einmal das Pivot-Element getauscht.



Um zwei Elemente zu sortieren braucht man 1 + 2 Vergleiche. Falls sie in der richtigen Reihenfolge stehen, muss man sie nicht vertauschen. Trotzdem tauscht man das Pivot-Element mit sich selbst. Man benötigt also 4 × 3 Vergleiche und 4 Vertauschungen. Das ergibt in der Summe 36 Vergleiche und sieben Tauschoperationen.

Lösung 4
Es wurden im Leitprogramm drei verschiedene Versionen vorgestellt.
Sie unterscheiden sich alle lediglich in der Wahl des Pivot-Elements und den daraus resultierenden Umstellungen.

- Das erste Element wird als Pivot-Element gewählt.
- Das letzte Element wird als Pivot-Element gewählt.
- Man betrachtet das erste das mittlere und das letzte Element und wählt das Element mit dem mittleren Wert.

Anhang B:

Internethinweise für die Schüler/Schülerinnen

http://www.cs.pitt.edu/~kirk/cs1501/animations/Sort3.html

www.educeth.ch/lehrpersonen/informatik/unterrichtsmaterialien_inf/programmieren/array/uebung_computer.pdf

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/insertionSort/insertionSort.html

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html

Alle hier aufgeführten Zugänge sind kostenlos. Inhalte und Formen dürfen nur für eigenes Lernen verwendet und nicht weiterverbreitet werden.

Anhang C

Material für die Lernenden

- Kartenspiel für die Kapiteltests
- Jeder Lernende sollte einen Computer mit Windows Betriebssystem inkl. Microsoft PowerPoint und einen Internetzugang zur Verfügung haben.
- CD/ Dateien mit den PowerPoint Präsentationen zu diesem Leitprogramm

Anhang D

Quellen

Literaturverzeichnis:

Aho A., J. Hopcroft, J. Ullman: *Data Structures and Algorithms,* Addison-Wesley 1983

Cormen T., C. Leiserson, R. Rivest, C. Stein: *Introduction to Algorithms*, 2. Auflage, McGraw-Hill 2001

Ottmann T., P. Widmayer: *Algorithmen und Datenstrukturen,* 3. Auflage, Spektrum 1996