Algorithmen und Datenstrukturen

Praktische Einführung und Programmierung

Stefan Bosse

Universität Koblenz - FB Informatik

Sortierverfahren

- Bisher waren die drei wichtigsten Operationen auf Abstrakten Datentypen:

  1. Einfügen von Elementen
  2. Löschen von Elementen
  3. Suchen nach Elementen

- Ein weiteres grundlegendes Problem in der Informatik neben dem Suchen ist das Sortieren.
- Bereits bei den Listen und dann bei den Bäumen war Restrukturierung wichtig um optimale Rechenzeiten zu erzielen
- Sortieren von Elementen ist Restrukturierung

Taxonomie der Sortierverfahren

Unter Sortieren versteht man das Anordnen einer Menge von Objekten in einer bestimmten Ord-nung. Diese Ordnung kann z. B. bei numerischen Daten durch die größer/kleiner-Relation oder bei Texten durch die lexikografische Reihenfolge gegeben sein.

- Die zu sortierenden n Elemente seien in einem Feld a mit den Komponenten a[0], a[1], ... , a[n − 1] gespeichert.
- Sortieren bedeutet nun, dass die Feldkomponenten durch eine Permutation i_1 , i_2 , ... , i_n der Indizes 0, 1, ... , n − 1 in diejenige Reihenfolge gebracht werden, die der gewünschten Ordnung, ausgedrückt durch eine Ordnungsrelation $\leq_f$, entspricht:

$$a[i_1], a[i_2], a[i_3], \ldots, a[i_n]$$

mit

$$a[i_1] \leq_f a[i_2] \leq_f a[i_3] \leq_f \ldots \leq_f a[i_n]$$

Taxonomie der Sortierverfahren

Der Sinn des Sortierens bzw. Ordnens liegt darin, dass der Zugriff auf Datensätze in einer geordneten Datei oder Datenstruktur wesentlich effizienter und schneller vonstatten geht, als in einer nicht geordneten Anordnung.

Die Problematik des Sortierens

- Sortieren ist damit eine in der Datenverarbeitung elementare, weit verbreitete und wichtige Operation.
- In fast allen Problemstellungen spielt das Sortieren von Daten eine mehr oder weniger bedeutende Rolle.
- Dies gilt insbesondere für die kommerzielle Datenverarbeitung, aber auch den technisch/wissenschaftlichen Bereich.

Untersuchungen haben ergeben, dass auf kommerziellen Anlagen ca. **25%** der CPU-Zeit auf Sortierläufe entfällt.

Taxonomie der Sortierverfahren

Es existiert eine große Anzahl von Sortieralgorithmen unter denen man im Einzelfall den geeig-netsten auswählen muss.

- Die Sortiermethoden hängen stärker als die meisten anderen Algorithmen von der Struktur der zu sortierenden Daten ab.
- Es ist ein ganz wesentlicher Unterschied, ob man die zu sortierenden Daten im relativ beschränkten Hauptspeicher mit wahlfreiem Zugriff oder auf langsameren, jedoch größeren sequentiell oder zyklisch arbeitenden externen Speichermedien – also auf Band- oder Plattenlaufwerken – halten muss.
- Gerade bei Sortieralgorithmen spielen auch kleine Leistungssteigerungen eine große Rolle, da sich dies wegen der Häufigkeit von Sortierläufen sehr stark auswirken kann.
- Komplexitätsberechnungen sind oft nichttrivial.
- Bei Algorithmen mit im Normalfall hervorragendem Zeitverhalten kann das Verhalten im ungünstigsten Fall (worst case) katastrophal sein.

Taxonomie der Sortierverfahren

Welche dramatischen Effekte die Verbesserung der **Komplexitätsordnung eines Sortierverfahrens** haben kann, zeigt das Beispiel einer 1987 in der damaligen Bundesrepublik Deutschland durchgeführten Volkszählung.

- Mit ca. n = 60.000.000 Datensätzen
- Bei einem angenommenen Zeitbedarf von nur einer Mikrosekunde pro Schlüsselvergleich würde ein Sortierlauf unter Verwendung des **Bubblesort** mit einer Komplexität von der Ordnung $O(n^2)$ ca. **114 Jahre** dauern
- Bei Verwendung von **Quicksort** mit einer Komplexität von $O(n \log_{10} n)$ ergibt sich dagegen eine Sortierzeit von nur **26 Minuten**!

Taxonomie der Sortierverfahren

- Ein wichtiges Merkmal von Sortierverfahren ist die **Stabilität**.
  - Ein Verfahren heißt stabil, wenn es die relative Reihenfolge gleicher Schlüssel in der Datei beibehält.

Name	Alter
Endig, Martin	30
Geist, Ingolf	28
Höpfner, Hagen	24
Schallehn, Eike	28

Name	Alter
Höpfner, Hagen	24
Geist, Ingolf	28
Schallehn, Eike	28
Endig, Martin	30

Abb. 1. Stabilität von Sortierverfahren: Nehmen wir eine Liste mit Daten zu Personen an, die alphabetisch nach den Namen sortiert ist. Wird diese Liste nun nach dem Alter der Personen sortiert, so werden bei Anwendung eines stabilen Verfahrens Personeneinträge mit dem gleichen Alter auch weiterhin alphabetisch geordnet sein.

Taxonomie der Sortierverfahren

Sortierbare Aggregationen (Datenstrukturen/Objekte):

- Arrays (Tabellen)
- Listen
- Bäume
- (Graphen)

Zwei Speicherklassen werden unterschieden:

In-place Verfahren
: Die Sortierung findet innerhalb und mit dem zu sortierenden Objekt statt. Es wird keine neue Datenstruktur erzeugt, die bestehende wird nur modifiziert. Auch alle Zwischenschritte werden mit dem Objekt direkt verarbeitet.

Out-of-place Verfahren
: Die Sortierung findet außerhalb des zu sortierenden Objekts statt, auch alle Zwischenschritte werden in eignen Datenstrukturen ausgeführt.

Taxonomie der Sortierverfahren

Gemäß den Speicherklassen kann man direkte und indirekte Sortierverfahren unterscheiden.

Drei einfache, direkte Sortierverfahren, die ein gegebenes Feld a[i] von Objekten am Platz ordnen:

- Sortieren durch Einfügen (Insertion)
- Sortieren durch Auswählen (Selection)
- Sortieren durch Austauschen (Swap).

Die Umstellung der Elemente geschieht dabei auf dem Eingabe-Array, das dann bei der Ausgabe die geordneten Daten enthält.

- Die wesentlichen elementaren Operation beim Sortieren sind:

1. Vergleichen (Compare) von Daten;
2. Umstellen (Move) von Daten.

Sortieren durch Einfügen

Man geht von einem Array a mit n Komponenten und Indizierung von 0 bis n − 1 aus. Nun wird a in zwei Intervalle aufgeteilt, die Zielsequenz a[0] bis a[i − 1] und die Quellensequenz a[i] bis a[n − 1]. Beginnend mit i = 1 wird bei jedem Schritt das Element a[i] aus der Quellensequenz entfernt und an der durch die Ordnungsrelation gegebenen Stelle in die Zielsequenz eingefügt.

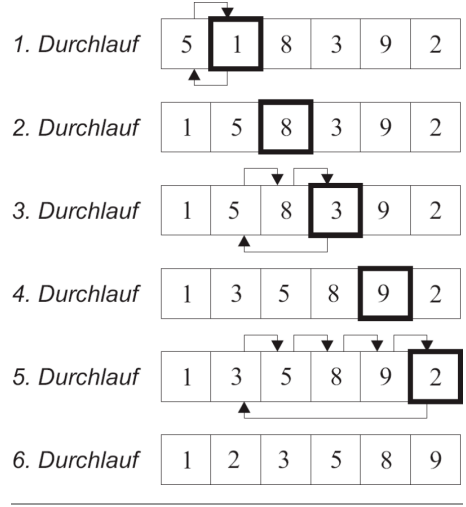


Abb. 2. InsertionSort am Beispiel

Sortieren durch Einfügen

```
function insertionSort(array) {
  for (i = 1; i < length(array); i++) {
    j = i;
    m = array[i];                                   m = array[i].key;
    // für alle Elemente links vom Marker-Feld
    while (j > 0 && array[j - 1] > m) {             .. array[j-i].key...
      // verschiebe alle größeren Elemente nach hinten
      array[j] = array[j - 1];
      j-- ;
    }
    // setze m auf das freie Feld
    array[j] = m;
  }
}
```

Alg. 1. Sortieren durch Einfügen bzw. Verschieben mit einem Array. Die Arrayelemente können auch Datenstrukturen sein und ein Attribute *key* bestimmt die Sortierung.

Komplexität des direkten Einfügens

Zur Berechnung der Komplexität berücksichtigt man,

- dass die Anzahl der Schlüsselvergleiche *c* beim Durchlauf i höchstens $c_{max}(i) = i$ und mindestens $c_{min}(i) = 1$ ist.
- Nimmt man an, dass alle Permutationen gleich wahrscheinlich sind, erhält man im Mittel pro Durchlauf $c_{mit}(i) = (c_{min}(i) + c_{max}(i))/2 = (i + 1)/2$ Schlüsselvergleiche.
- Die Zahl der Zuweisungen von Elementen *m* ist für die einzelnen Durchläufe immer C(i) + 2.

$$C_{\min}(n) = n - 1, C_{\max}(n) = \frac{n^2 - n}{2}, C_{avg}(n) = \frac{n^2 + n - 2}{4}$$

$$M_{\min}(n) = 3(n - 1), M_{\max}(n) = \frac{n^2 + 3n - 4}{2}, M_{avg}(n) = \frac{n^2 + 9n - 10}{4}$$

mit *C* als Gesamtkomplexität für den Vergleich und *M* für das Verschieben.

Komplexität des direkten Einfügens

Die Komplexität ist also im Mittel sowohl für C(n) als auch für M(n) von der Ordnung $O(n^2)$.

- Das Zeitverhalten ist am günstigsten, wenn alle Elemente von Anfang an geordnet sind und am ungünstigsten, wenn alle Elemente in umgekehrter Reihenfolge sortiert waren.
- Dieses Verhalten wird als natürlich bezeichnet.
- Offenbar ist diese Sortierfunktion auch stabil, da die Reihenfolge von Elementen mit übereinstimmenden Schlüsseln nicht geändert wird.

Sortieren durch Selektion

- Auch das zweite hier zu betrachtende Verfahren kann dem Sortieren beim Kartenspielen entlehnt werden.
- Die Idee ist hierbei, jeweils das größte Element auszuwählen und an das Ende der Folge zu setzen.
- Dies wird in jedem Schritt mit einem jeweils um eins verkleinerten Bereich der Folge ausgeführt, so dass sich am Ende der Folge die bereits sortierten Elemente sammeln.
- Das Prinzip des Auswählens oder Selektierens des größten Elementes hat diesem Verfahren auch den Namen **SelectionSort** gegeben.

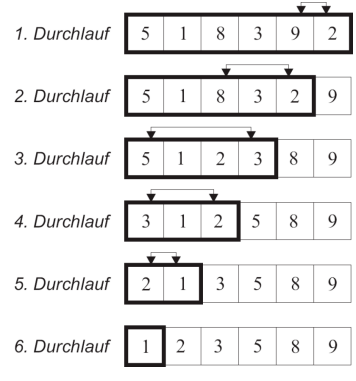


Abb. 3. SelectionSort am Beispiel

Sortieren durch Selektion

```
function swap (array,idx1,idx2) {
  t=array[idx1]
  array[idx1]=array[idx2]
  array[idx2]=t
}
function maxIndex(array,idx) {
  maxi=idx
  maxv=array[idx]
  for (var i=0;i<idx;i++) {
    if (array[i]>maxv) {
      maxi=i
      maxv=array[i]
    }
  }
  return maxi
}
function selectionSort(array) {
  p=length(array)-1
  while(p) {
    g = maxIndex(array,p)
    swap (array,p,g)
    p--
  }
}
```

Sortieren durch Vertauschen

- Eines der bekanntesten, wenn auch kein besonders effizientes Sortierverfahren ist **BubbleSort**.
- Der Name ist aus der Vorstellung abgeleitet, dass sich bei einer vertikalen Anordnung der Elemente der Folge verschieden große, aufsteigende Blasen (»Bubbles«) wie in einer Flüssigkeit von allein sortieren, da die größeren Blasen die kleineren »überholen«.
- Das Grundprinzip besteht demzufolge darin, die Folge immer wieder zu durchlaufen und dabei benachbarte Elemente, die nicht der gewünschten Sortierreihenfolge entsprechen, zu vertauschen.
- Elemente, die größer als ihre Nachfolger sind, überholen diese daher und steigen zum Ende der Folge hin auf.

Sortieren durch Vertauschen

```
function BubbleSort (array)
  // Eingabe: zu sortierende Folge F der Länge n
  do {
    swaps=0
    for (i=0;i<n;i++) {
      if (array[i]> array[i + 1]) {
        swap(array,i,i+1)
        swaps++
      }
    }
  } while (swaps==0)
}
```

Alg. 3. Nachbarschaftsvergleich und Vertauschung

Sortieren durch Vertauschen

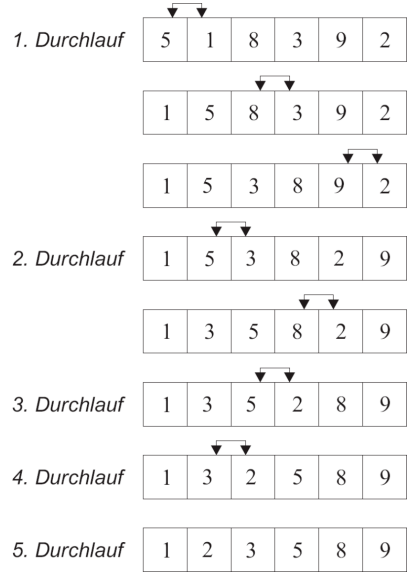


Abb. 4. Bubblesort im Beipiel, Gezeigt sind nur Schritte wo eine Vertauschung stattfindet.

Sortieren durch Vertauschen

- Der Algorithmus weist schon eine Verbesserung gegenüber der ursprünglichen Idee auf, indem die Folge nicht n · n-mal durchlaufen wird, sondern der Abbruch bereits dann erfolgt, wenn keine Vertauschung mehr stattgefunden hat.
- Eine weitere Optimierung ist dass in jedem Durchlauf das jeweils größte Element an das Ende bewegt wird. Daher genügt es, im zweiten Durchlauf nur den Bereich 1 … n − 2, im dritten Durchlauf den Bereich 1 … n − 3 usw. zu betrachten.
- Schließlich kann auch noch »Lokalität« von Feldzugriffen speziell bei großen Folgen ausgenutzt werden, indem der Durchlauf abwechselnd auf- bzw. abwärts erfolgt.
    - So kann die Effizienz der Sortierung dadurch verbessert werden, dass Teile der Folge noch im CPU-Cache – einem schnellen Zwischenspeicher des Prozessors – vorhanden sind, wenn ein neuer Durchlauf gestartet wird.

Nicht nur die Rechenkomplexitätsklasse ist relevant, auch die Implementierung im Detail, die signifikanten Einfluss auf die Rechenzeit haben kann.

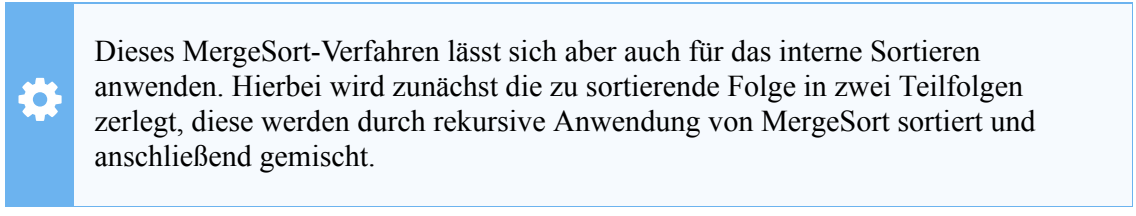
Vergleich Sortierenverfahren

Sortieren durch Mischen

MergeSort: Die bisher vorgestellten Verfahren erfordern einen direkten Zugriff auf einzelne Elemente der Folge und sind daher nur für internes Sortieren geeignet. Sollen dagegen Dateien sortiert werden, die nicht in den Hauptspeicher passen, müssen andere Verfahren zum Einsatz kommen. Eine Möglichkeit ist hier, das Sortieren in zwei Schritten oder Phasen auszuführen:

- Die Folge wird in Teile zerlegt, die jeweils in den Hauptspeicher passen und daher getrennt voneinander mit internen Verfahren sortiert werden können. Diese sortierten Teilfolgen werden wieder in Dateien ausgelagert.
- Anschließend werden die Teilfolgen parallel eingelesen und gemischt, indem jeweils das kleinste Element aller Teilfolgen gelesen und in die neue Folge (d.h. wieder in eine Datei) geschrieben wird.

Sortieren durch Mischen



Dieses MergeSort-Verfahren lässt sich aber auch für das interne Sortieren anwenden. Hierbei wird zunächst die zu sortierende Folge in zwei Teilfolgen zerlegt, diese werden durch rekursive Anwendung von MergeSort sortiert und anschließend gemischt.

- Es stellt sich jedoch die Frage, wie man dabei zu sortierten Teilfolgen kommt.
  - Dies wird dadurch erreicht, dass MergeSort so lange rekursiv angewendet wird, bis die zu sortierenden Teilfolgen nur noch aus einem Element bestehen.
  - In diesem Fall liefert das Mischen ja eine neue, sortierte Folge.
  - Das eigentliche Sortieren kann **rekursiv** realisiert werden.

Sortieren durch Mischen

Der Algorithmus ist aufgeteilt in:

1. Mischen von zwei Folgen mit Auswahl *Merge* (Conquer)
2. Sortieren durch Mischen *MergeSort* (Divide), obwohl hier nur das Teilen stattfindet.

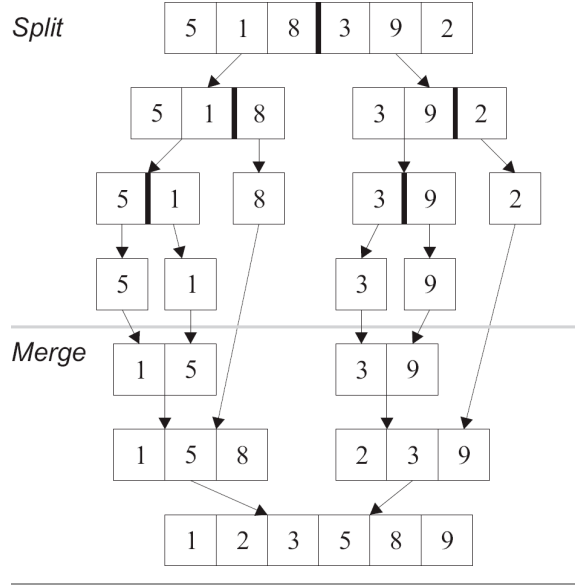


Abb. 5. Aufteilen, Verarbeiten, und Zusammenführung

Sortieren durch Mischen

```
function MergeSort (array) {
  // Eingabe: eine zu sortierende Folge F
  // Ausgabe: eine sortierte Folge FS
  n = length(array)
  if (n==1) // einelementig then
    return array
  else {
   // Teile F in F1 und F2;
   splitIndex = n/2
   F1 = slice(array,0,splitIndex-1)
   F2 = slice(array,splitIndex,n-1)
   F1 = MergeSort (F1)
   F2 = MergeSort (F2)
   return Merge (F1, F2)
  }
}
```

Alg. 4. Grundgerüst des Sortieren durch Mischen

Sortieren durch Mischen

Der Vorgang des Mischens erfordert in der Regel doppelten Speicherplatz, da ja eine neue Folge aus den beiden sortierten Folgen erzeugt wird. Eine Alternative ist das Mischen in einem Feld, allerdings ist hier ein aufwendigeres Verschieben notwendig.

- Die Aufteilung in die zwei Abschnitte »Split« und »Merge« entsteht nur durch die Rekursion im Algorithmus, indem zu Beginn jeweils das Zerlegen erfolgt und erst als letzter Schritt das eigentliche Mischen.