

# Algorithmen und Datenstrukturen 1

## 6. Vorlesung

*Peter F. Stadler*

Universität Leipzig  
Institut für Informatik  
*studla@bioinf.uni-leipzig.de*

# Merge-Sort

Anwendbar für internes und externes Sortieren

Ansatz (Divide-and-Conquer-Strategie)

- rekursives Zerlegen in jeweils gleich große Teilfolgen
- Verschmelzen (Mischen) der sortierten Teilfolgen

Algorithmus (2-Wege-Merge-Sort)

`sort(A, l, r)` sortiert das Feld A im Bereich l bis r

Falls  $r-l=1$ : fertig

Falls  $r-l>1$

Mitte der Liste bestimmen: `int m = (l+r+1)/2`

linken Teil sortieren: `sort(A, l, m-1)`

rechten Teil sortieren: `sort(A, m, r)`

zusammenfügen: `merge(A, l, m, r)`

# Mischen für Merge-Sort

Pro Teilliste wird Zeiger (Index) auf nächstes Element geführt

- jeweils kleinstes Element wird in Ergebnisliste übernommen und Indexzeiger fortgeschaltet
- sobald eine Teilliste "erschöpft" ist, werden alle verbleibenden Elemente der anderen Teilliste in die Ergebnisliste übernommen
- lineare Kosten

merge(A, l, m, r) mischt die Bereiche l ... m-1 und m ... r in Feld B der Länge r-l+1

1. Schritt:                    j = 1; k = m  
Schleife zum Füllen von B:  
                                for i = 0; i < r - l + 1; i ++  
Auswahl:                        if k > r or (j < m and A[j] < A[k])  
  B[i] = A[j ++]  
  else  
  B[i] = A[k ++]
2. Schritt: Übertragen: for (i = 0, i < r - l + 1); i ++) A[l + i] = B[i]

# Beispiel Merge-Sort

<b>7</b>	<b>4</b>	<b>15</b>	<b>3</b>	<b>11</b>	<b>17</b>	<b>12</b>	<b>8</b>	<b>18</b>	<b>14</b>
<b>4</b>	<b>7</b>	<b>15</b>							
			<b>3</b>	<b>11</b>					
<b>4</b>	<b>7</b>	<b>3</b>	<b>11</b>	<b>15</b>					
<b>3</b>	<b>4</b>	<b>7</b>	<b>11</b>	<b>15</b>					

# Direktes (reines, nicht-rekursives) 2-Wege-Merge-Sort

## **Algorithmus**

- zunächst werden benachbarte Elemente zu sortierten Teillisten der Länge 2 gemischt
- fortgesetztes Mischen benachbarter (sortierter) Teillisten
- Länge sortierter Teillisten verdoppelt sich pro Durchgang
- Sortierung ist beendet, sobald in einem Durchgang nur noch zwei Teillisten verschmolzen werden

# Beispiel Direktes Merge-Sort

<b>7</b>	<b>4</b>	<b>15</b>	<b>3</b>	<b>11</b>	<b>17</b>	<b>12</b>	<b>8</b>	<b>18</b>	<b>14</b>
<b>4</b>	<b>7</b>	<b>3</b>	<b>15</b>	<b>11</b>	<b>17</b>	<b>8</b>	<b>12</b>	<b>14</b>	<b>18</b>
<b>3</b>	<b>4</b>	<b>7</b>	<b>15</b>	<b>8</b>	<b>11</b>	<b>12</b>	<b>17</b>	<b>14</b>	<b>18</b>
<b>3</b>	<b>4</b>	<b>7</b>	<b>8</b>	<b>11</b>	<b>12</b>	<b>15</b>	<b>17</b>	<b>14</b>	<b>18</b>

# Natürliches 2-Wege-Merge-Sort

## **Algorithmus**

- statt mit 1-elementigen Teillisten zu beginnen, werden bereits anfangs möglichst lange sortierte Teilfolgen ("runs") verwendet
- Nutzung einer bereits vorliegenden (natürlichen) Sortierung in der Eingabefolge

# Beispiel Natürliches Merge-Sort

<b>7</b>	<b>4</b>	<b>15</b>	<b>3</b>	<b>11</b>	<b>17</b>	<b>12</b>	<b>8</b>	<b>18</b>	<b>14</b>
<b>4</b>	<b>7</b>	<b>15</b>	<b>3</b>	<b>11</b>	<b>12</b>	<b>17</b>	<b>8</b>	<b>14</b>	<b>18</b>
<b>3</b>	<b>4</b>	<b>7</b>	<b>11</b>	<b>12</b>	<b>15</b>	<b>17</b>	<b>8</b>	<b>14</b>	<b>18</b>



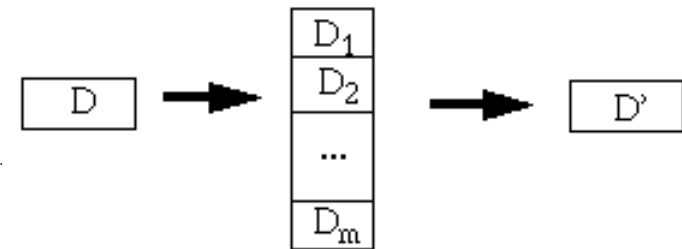
# Externes Sortieren

Anwendung, falls zu sortierende Daten nicht vollständig im Hauptspeicher gehalten werden können

Hauptziel: Minimierung von Externspeicherzugriffen

Ansatz

- Zerlegen der Eingabedatei in  $m$  Teildateien, die jeweils im Hauptspeicher sortiert werden können
- Interne Sortierung und Zwischenspeicherung der  $R$
- Mischen der  $m$  Runs ( $m$ -Wege-Mischen)



Bewertung

- optimale E/A-Kosten (1 Durchgang)
- setzt Dateispeicherung auf Direktzugriffsmedium (z.B. Magnetplatten) voraus, da ggf. sehr viele temporäre Dateien  $D_i$
- verfügbarer Hauptspeicher muß wenigstens  $m+1$  Seiten (Blöcke) umfassen

# Sortieren mit Bändern

Restriktiver, da hier kostengünstig nur sequenzielle Zugriffe möglich sind.

- Ausgeglichenes 2-Wege-Merge-Sort (4 Bänder)
- Ausgeglichenes k-Wege-Merge-Sort ( $2k$  Bänder)

## **Ausgeglichenes 2-Wege-Merge-Sort**

- vier Bänder B1, B2, B3, B4; Eingabe sei auf B1; Hauptspeicher fasse  $r$  Datensätze
- Wiederholtes Lesen von  $r$  Sätzen von B1, interne Sortierung und abwechselndes Ausschreiben der Runs auf B3 und B4 bis B1 erschöpft ist
- Mischen der Runs von B3 und B4 (ergibt Runs der Länge  $2r$ ) und abwechselndes Schreiben auf B1 und B2
- Fortgesetztes Mischen und Verteilen bis nur noch 1 Run übrig bleibt

# Beispiel: Sortieren mit 4 Bändern, $r=4$

Band

1	<b>14</b>	<b>4</b>	<b>3</b>	<b>17</b>	<b>22</b>	<b>5</b>	<b>25</b>	<b>13</b>	<b>9</b>	<b>10</b>	<b>1</b>	<b>11</b>	<b>12</b>	<b>6</b>	<b>2</b>	<b>15</b>
2																
3	3	4	14	17	1	9	10	11								
4	5	13	22	25	2	6	12	15								
1	3	4	5	13	14	17	22	25								
2	1	2	6	9	10	11	12	15								

# Verbesserungen beim Merge-Sort

## Ausgeglichenes k-Wege-Merge-Sort

- k-Wege-Aufteilen und k-Wege-Mischen mit  $2k$  Bändern

**Mischen bei Mehrwege-Merge-Sort durch Auswahlbaum** (Turnier-Sortierung oder Heap-Sort) beschleunigen: *Replacement Selection Sort*

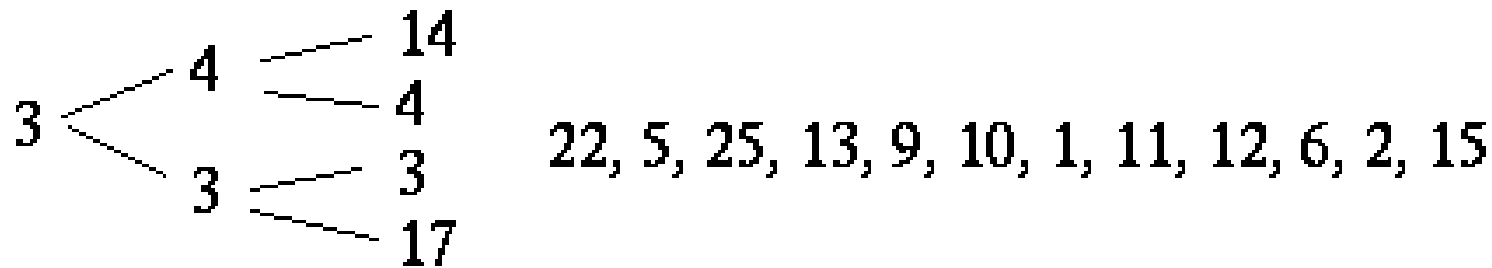
Bei  $k$  Schlüsseln kann Entfernen des Minimums und Hinzufügen eines neuen Schlüssels in  $O(\log k)$  Schritten erfolgen

Auswahlbaum kann auch bei unsortierter Eingabe (initiale Zerlegung) genutzt werden, um Länge der erzeugten Runs zu erhöhen  $\Rightarrow$  Sortierung erfordert weniger Durchgänge

- ausgegebenes (Wurzel-) Element wird im Auswahlbaum durch nächstes Element  $x$  aus der Eingabe ersetzt
- $x$  kann noch im gleichen Run untergebracht werden, sofern  $x$  nicht kleiner als das größte schon ausgegebene Element ist
- Ersetzung erfolgt solange bis alle Schlüssel im Auswahlbaum kleiner sind als der zuletzt ausgegebene ( $\Rightarrow$  neuer Run)
- im Mittel verdoppelt sich die Run-Länge (auf  $2r$ )

# Beispiel: Replacement Selection Sort

Beispiel 14, 4, 3, 17, 22, 5, 25, 13, 9, 10, 1, 11, 12, 6, 2, 15 (r=4)



14	4	3	17	22	5	25	13	9	10	1	11	12	6	2	15
3	14	4	22	17											
3	4	14	5	22	17										
3	4	5	14	25	22	17									
3	4	5	14	13	25	22	17								

# Zusammenfassung

Kosten allgemeiner Sortierverfahren wenigstens  $O(n \log n)$

Elementare Sortierverfahren: Kosten  $O(n^2)$

- einfache Implementierung; ausreichend bei kleinem  $n$
- gute Lösung: Insertion Sort

$O(n \log n)$ -Sortierverfahren: Heap-Sort, Quick-Sort, Merge-Sort

- Heap-Sort und Merge-Sort sind Worst-Case-optimal ( $O(n \log n)$ )
- in Messungen erzielte Quick-Sort in den meisten Fällen die besten Ergebnisse

Begrenzung der Kosten für Umkopieren durch indirekte Sortierverfahren

Vorteilhafte Eigenschaften

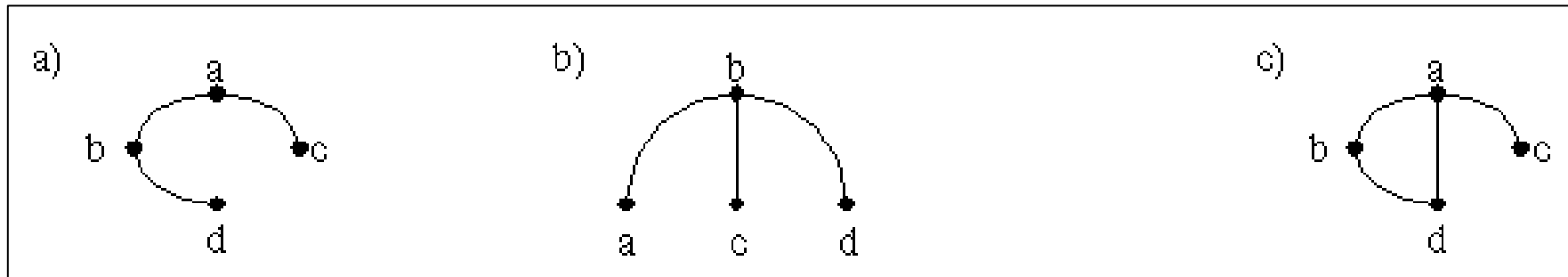
- Ausnutzen einer weitgehenden Vorsortierung
- Stabilität

Externes Sortieren

- fortgesetztes Zerlegen und Mischen
- Mehrwege-Merge-Sort reduziert Externspeicherzugriffe

# Wiederholung: Bäume

Bäume lassen sich als sehr wichtige Klasse von Graphen auffassen



Danach ist ein Baum

- ein azyklischer, einfach zusammenhängender Graph
- d.h., er enthält keine Schleifen und Zyklen; zwischen jedem Paar von Knoten besteht höchstens eine Kante

# Orientierte Bäume

Eine Menge  $B$  von Objekten ist ein orientierter (Wurzel-) Baum, falls

1. in  $B$  ein ausgezeichnetes Element  $w$  - Wurzel von  $B$  - existiert
2. die Elemente in  $B - \{w\}$  disjunkt zerlegt werden können in  $B_1, B_2, \dots, B_m$ , wobei jedes  $B_i$  ebenfalls ein orientierter Baum ist.

Anschaulich:

- Die Kanten des Graphen sind gerichtet.
- Von der Wurzel gehen nur Kanten aus, keine treffen ein.
- Jeder Knoten ist von der Wurzel aus auf genau einem Weg entlang der gerichteten Kanten erreichbar.
- Für jeden Nicht-Wurzelknoten gibt es Knoten, die nicht entlang der gerichteten Kanten erreichbar sind.

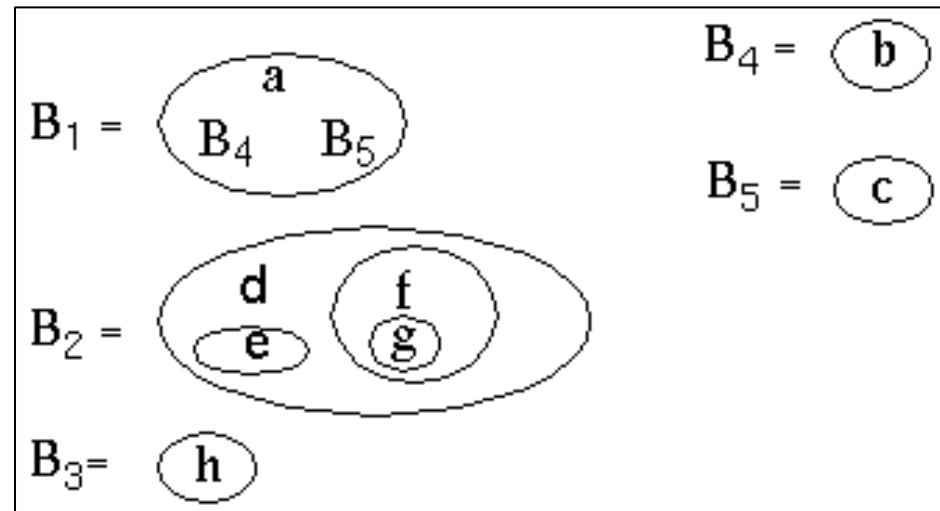
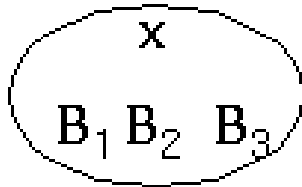


# Darstellungsarten für orientierte Bäume I

## 1. Mengendarstellung:

Objekte: a, b, c, ...

Bäume:  $B_1, B_2, B_3, \dots$



## 2. Klammerdarstellung: Wurzel = erstes Element innerhalb eines Klammerpaares

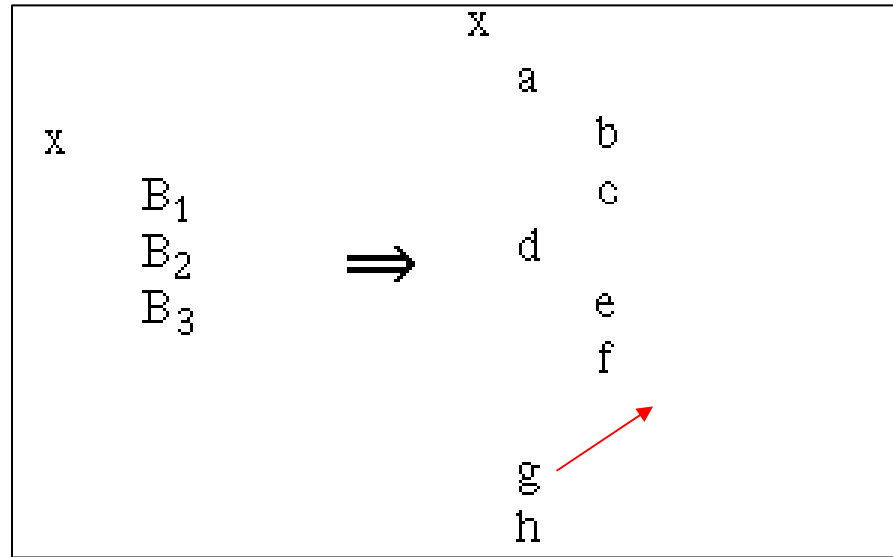
$\{x, B_1, B_2, B_3\}$

$\{x, \underbrace{\{a, \{b\}, \{c\}\}}_{B_1}, \underbrace{\{d, \{e\}, \{f, \{g\}\}\}}_{B_2}, \underbrace{\{h\}}_{B_3}\}$

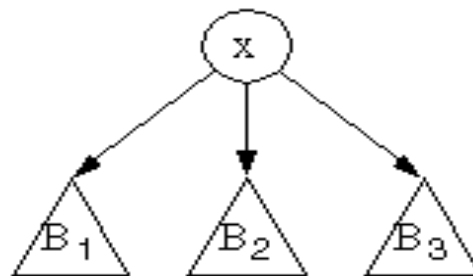
$\{a, \{b\}, \{c\}\} \equiv \{a, \{c\}, \{b\}\}$

# Darstellungsarten für orientierte Bäume II

## 3. Rekursives Einrücken



## 4. Graphendarstellung



# Graphendarstellung orientierter Bäume

Wurzel: x

Blätter: 5

innere Knoten: x,a,d,f

Grad K = # Nachfolger von K

Grad (x) = 3

Grad (g) = 0

Grad (Baum) = Max (Grad(K<sub>i</sub>)) = 3

Stufe (K<sub>i</sub>) = Pfadlänge l von Wurzel nach K<sub>i</sub>

Stufe 0: x

Stufe 1: a, d, h

Stufe 2:

Stufe 3:

Höhe h = 4

Gewicht w = # der Blätter = 5

