Algorithmen und Datenstrukturen 1

5. Vorlesung

Martin Middendorf / Peter F. Stadler

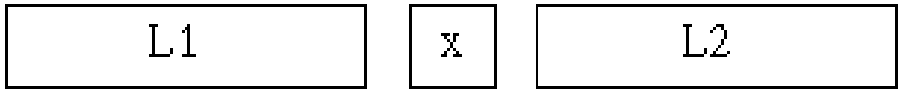
Universität Leipzig
Institut für Informatik
middendorf@informatik.uni-leipzig.de
studla@bioinf.uni-leipzig.de

Quick-Sort I

Hoare, 1962; andere Bezeichnung: Partition-Exchange-Sort

Anwendung der Divide-and-Conquer-Strategie auf Sortieren durch Austauschen:

- Bestimmung eines Pivot-Elementes x in L
- Zerlegung der Liste in zwei Teillisten L1 und L2 durch Austauschen, so daß linke (rechte) Teilliste L1 (L2) nur Schlüssel enthält, die kleiner (größer oder gleich) als x sind



- Rekursive Sortierung der Teillisten, bis nur noch Listen der Länge 1 verbleiben

Realisierung der Zerlegung

- Duchlauf des Listenbereiches von links über Indexvariable i, solange bis L [i] >= x vorliegt
- Duchlauf des Listenbereiches von rechts über Indexvariable j, solange bis L [j] <= x vorliegt
- Austausch von L[i] und L [j]
- Fortsetzung der Durchläufe bis i > j gilt

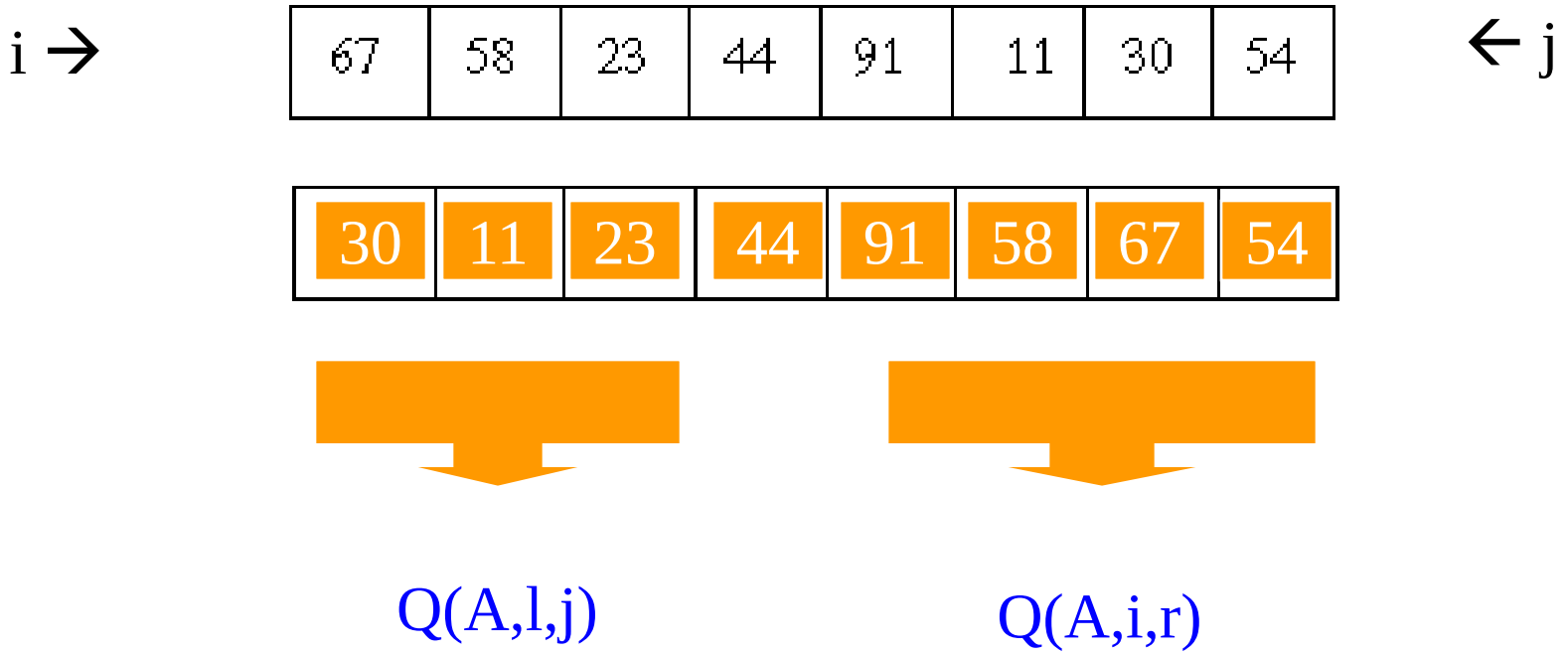
Quick-Sort III: Algorithmus

Algorithmus `qsort(A,l,r)` sortiert die Felder `A[l] bis A[r]`:

```
Initialisieren:     i=l; j=r
Fertig?             If (r<=l) return
Pivot-Element:      piv = A[(l+r)/2]
Schleife            do
   Links suchen:          while (A[i]<piv) i++
   Rechts suchen:         while (A[j]>piv) j--
   Tauschen:              if i<=j
                                tausche A[i] und A[j]
                                i++; j--
                    while (i<=j)
Rekursiver Aufruf:  qsort(A,l,j)
                    qsort(A,i,r)
```

Quick-Sort II

Beispiel: Berechne Q(A,l,r)



Quick-Sort: Eigenschaften I

In-situ-Verfahren; nicht "stabil"

Kosten am geringsten, wenn Teillisten stets gleichlang sind, d.h. wenn das Pivot-Element dem mittleren Schlüsselwert (Median) entspricht

- Halbierung der Teillisten bei jeder Zerlegung
- Kosten O (n log n)

Worst-Case

- Liste der Länge k wird in Teillisten der Längen 1 und k-1 zerlegt (z.B. bei bereits sortierter Eingabe und Wahl des ersten oder letzten Elementes als Pivot-Element)
- Kosten O (n^2)

Quick-Sort: Eigenschaften II

Wahl des Pivot-Elementes ist von entscheidender Bedeutung.

Sinnvolle Methoden sind:

- mittleres Element (wie im Algorithmus `qsort(A,l,r)`)
- zufälliges Element
- mittlerer Wert von k Elementen (z.B. k=3)

=> bei fast allen Eingabefolgen können Kosten in der Größenordnung O (n log n) erzielt werden

Zahlreiche Variationen von Quick-Sort

(z.B. Behandlung von Duplikaten, Umschalten auf elementares Sortierverfahren für kleine Teillisten)

Turnier-Sortierung

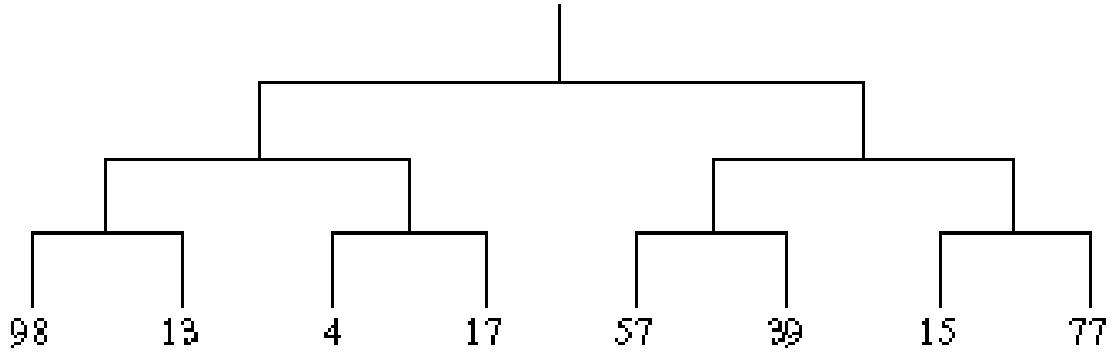
Maximum- bzw. Minimum-Bestimmung einer Sortierung analog zur Siegerermittlung

bei Sportturnieren mit KO-Prinzip

- paarweise Wettkämpfe zwischen Spielern/Mannschaften
- nur Sieger kommt weiter
- Sieger des Finales ist Gesamtsieger

Zugehörige Auswahlstruktur

ist ein binärer Baum

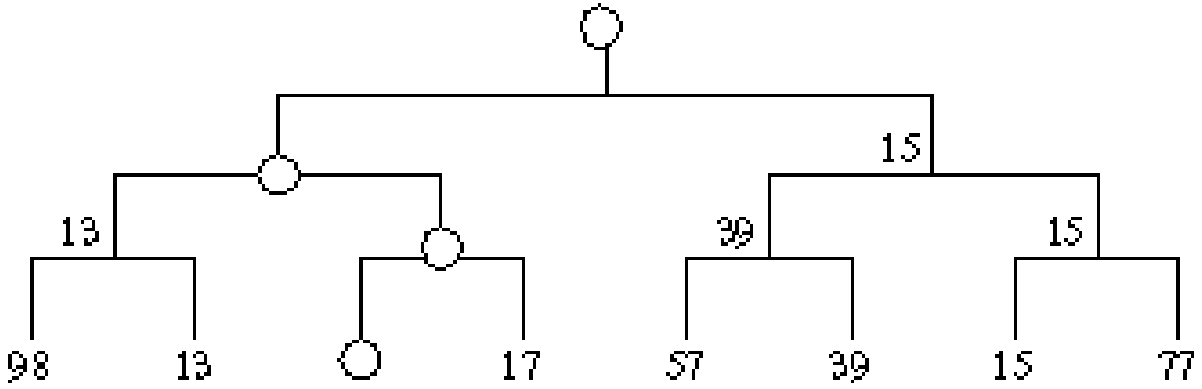


Turnier-Sortierung: Zweiter Sieger

Aber: der zweite Sieger (das zweite Element der Sortierung) ist nicht automatisch der Verlierer im Finale

Stattdessen: Neuaustragung des Wettkampfes auf dem Pfad des Siegers (ohne seine Beteiligung)

- Pfad für Wurzelelement hinabsteigen und Element jeweils entfernen
- Neubestimmung der Sieger

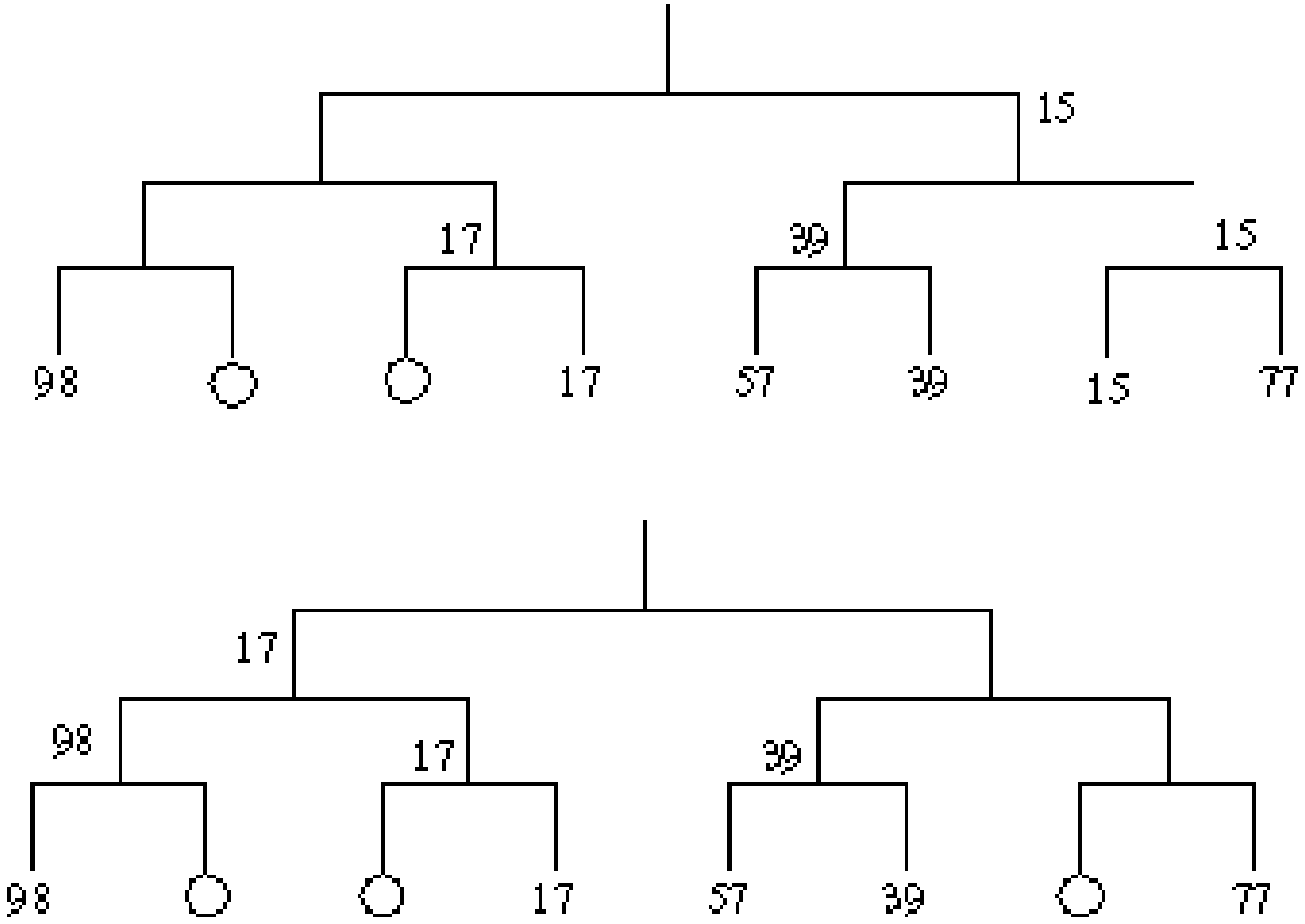


Turnier-Sortierung: Algorithmus

Algorithmus TOURNAMENT SORT:

```
"Spiele ein KO-Turnier und erstelle dabei einen binären
Auswahlbaum"
FOR I := 1 TO n DO
        "Gib Element an der Wurzel aus"
        "Steige Pfad des Elementes an der Wurzel hinab und
           lösche es"
        "Steige Pfad zurück an die Wurzel und spiele ihn
           dabei neu aus"
   END
```

Turnier-Sortierung: Beispiel weiter



Turnier-Sortierung: Aufwand

Anzahl Vergleiche (Annahme $n = 2^k$)

- #Vergleiche für Aufbau des initialen Auswahlbaumes

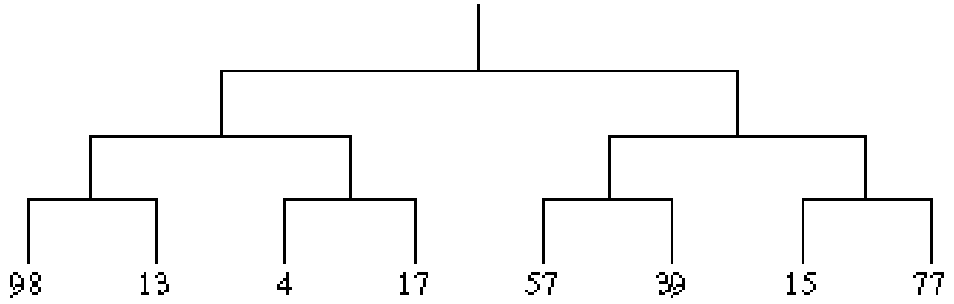
  $2^{k-1} + 2^{k-2} + ... + 1 = 2^k - 1 = n - 1$

- pro Schleifendurchlauf Absteigen und Aufsteigen im Baum über k Stufen:

  $2k = 2 \log_2 n$ Vergleiche

- Gesamtaufwand = $n - 1 + 2k(n - 2)$

Platzbedarf für Auswahlbaum



$2^k + 2^{k-1} + 2^{k-2} + ... + 1 = 2^{k+1} - 1 = 2n - 1$

Heap-Sort (Williams, 1964)

Reduzierung des Speicherplatzbedarfs gegenüber Turnier-Sort, indem "Löcher" im Auswahlbaum umgangen werden

Heap (Halde):

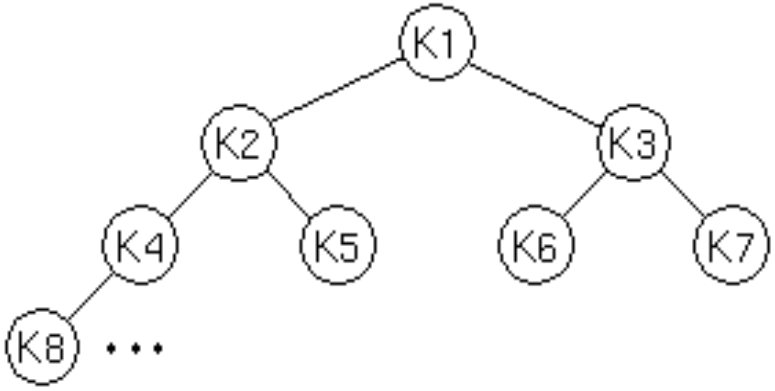
- hier: binärer Auswahlbaum mit folgender Eigenschaft (Heap-Eigenschaft): das größte Element jedes Teilbaumes befindet sich in dessen Wurzel

  => Baumwurzel enthält größtes Element

Heap-Sort: Absinken

Vorgehensweise

Repräsentation der Schlüsselfolge K1, K2, K3, ... in einem Binärbaum minimaler Höhe nach nebenstehendem Schema



Herstellen der Heap-Eigenschaft: ausgehend von Blättern durch "Absinken" von Knoteninhalten, welche kleiner sind als für einen der direkten

Nachfolgerknoten

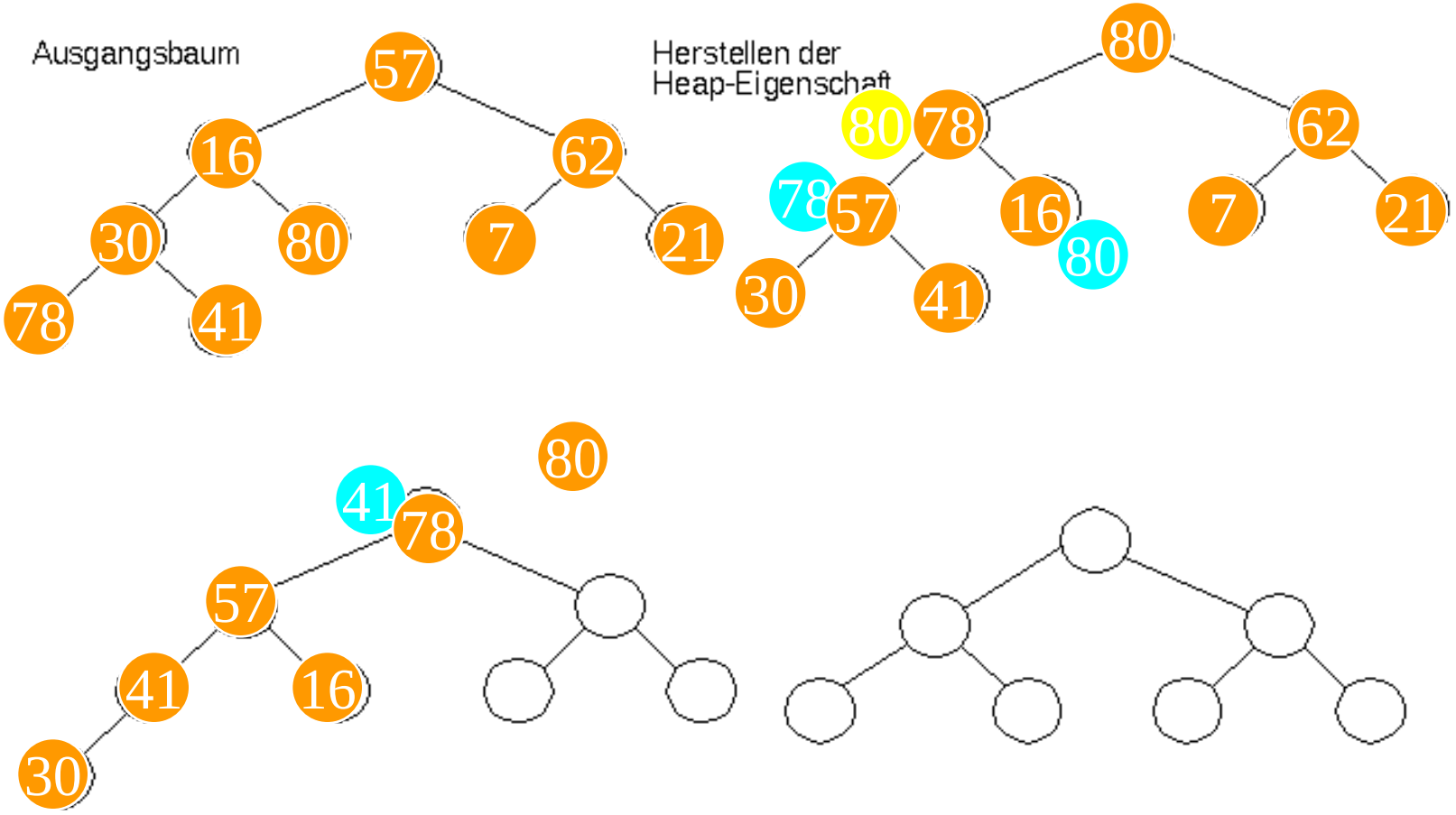
Ausgabe des Wurzelelementes

Ersetzen des Wurzelelementes mit dem am weitesten rechts stehenden Element der untersten Baumebene

Wiederherstellen der Heap-Eigenschaft durch "Absenken" des Wurzelelementes; Ausgabe der neuen Wurzel usw. (solange bis Baum nur noch 1 Element enthält)

Heap-Sort: Beispiel

Beispiel für Schlüsselfolge 57, 16, 62, 30, 80, 7, 21, 78, 41



Heap-Sort: Algorithmus

Effiziente Realisierbarkeit mit Arrays

- Wurzelelement = erstes Feldelement A [1]
- direkter Vaterknoten von A [i] ist A [i/2]
- Heap-Bedingung: A [i] <= A [i/2]

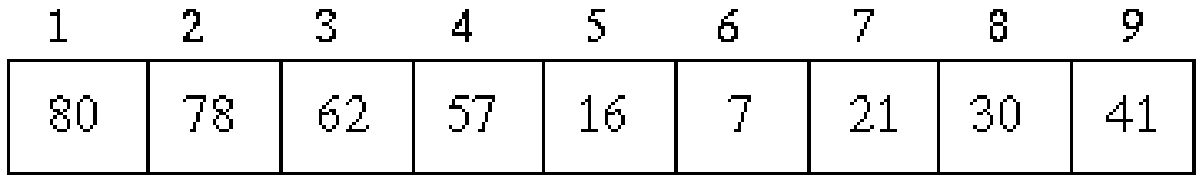
Algorithmus

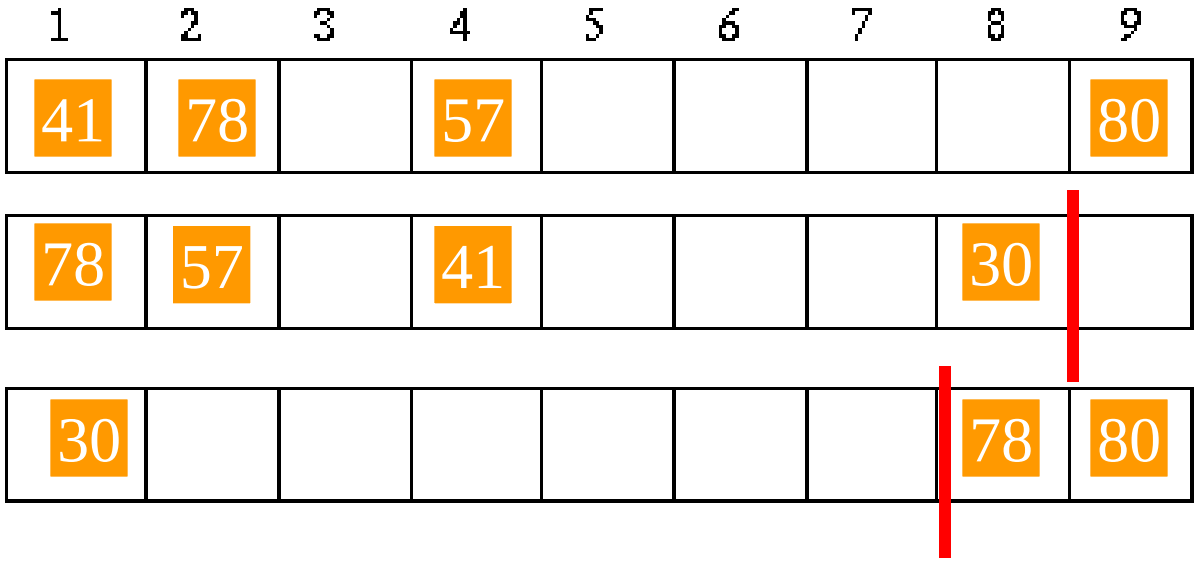
1. Aufbau des Heaps durch Absenken aller Vaterknoten A[n/2] bis A[1]
2. Vertauschen von A[1] mit A[n] (statt Entfernen der Wurzel)
3. Reduzierung der Feldlänge n um 1
4. Falls n > 1:
   - Absenken der neuen Wurzel A[1] (Prozedur Sink)
   - Gehe zu 2

Heap-Sort: Beispiel

Ausgangsliste 57, 16, 62,30, 80, 7, 21, 78, 41

Heap-Darstellung





Heap-Sort: Aufwand

Für $2^{k-1} <= n < 2^k$ gilt

- Baum hat die Höhe k mit k = 1,2,...
- Anzahl der Knoten auf Stufe i ($0 \leq i \leq k-1$) ist 2^i

Kosten der Prozedur Sink

- Schleife wird höchstens h - 1 mal ausgeführt (h ist Höhe des Baumes mit Wurzelelement i)
- Zeitbedarf O(h) mit $h \leq k$

Kosten zur Erstellung des anfänglichen Heaps

- Absenken nur für Knoten mit nicht-leeren Unterbäumen (Stufe k-2 bis Stufe 0)
- Kosten für einen Knoten auf der Stufe k-2 betragen höchstens $1 \cdot c$ Einheiten, die Kosten für die Wurzel $(k-1) \cdot c$ Einheiten.
- max. Kosten insgesamt $2^{k-2} \cdot 1 \cdot c + 2^{k-3} \cdot 2 \cdot c + ... + 2^0 \cdot (k-1) \cdot c =$

$$= \Sigma c \cdot i \cdot 2^{k-i-1} = c \cdot 2^{k-1} \ \Sigma\, i \cdot 2^{-i} \ < \ c \cdot n \cdot \ \Sigma\, i \cdot 2^{-i} < \ c \cdot 2n$$

- Gesamtkosten O (n)

Kosten der Sortierung: n-1 Aufrufe von Sink mit Kosten von höchstens $O(k) = O(\log_2 n)$

maximale Gesamtkosten von HEAPSORT: $O(n) + O(n \log_2 n) = O(n \log_2 n)$

Sortieren durch Streuen und Sammeln (Distribution-Sort, Bucket-Sort)

Lineare Sortierkosten O (n) !

Kein allgemeines Sortierverfahren, sondern beschränkt auf kleine und zusammenhängende Schlüsselbereiche 1..m

Idee: Die Häufigkeit der Schlüsselwerte wird für alle Werte bestimmt und daraus die relative Position jedes Eingabewertes in der Sortierfolge bestimmt

Vorgehensweise

- Hilfsfeld COUNT [1..m]
- Bestimmen der Häufigkeit des Vorkommens für jeden der m möglichen Schlüsselwerte ("Streuen")
- Bestimmung der akkumulierten Häufigkeiten in COUNT
- von i = 1 bis m: falls COUNT [i] > 0 wird i-ter Schlüsselwert COUNT [i]-mal ausgegeben ("Sammeln")

Beispiel

Beispiel: m=10; Eingabe 4 0 1 1 3 5 6 9 7 3 8 5 (n=12)

Streuen:

```
0  1  .  3  4  5  6  7  8  9
   1     3     5
```

Sammeln:

```
0 1 1 3 3 4 5 5 6 7 8 9
```

Kosten:

- Keine Duplikate	a) Streuen	$C_1 \cdot n$
	b) Sammeln	$C_2 \cdot m$
- mit d Duplikaten:	a) Streuen	$C_1 \cdot n$
	b) Sammeln	$C_2 \cdot (m+d)$

Distribution-Sort / Fachverteilen

Verallgemeinerung: Sortierung von k-Tupeln gemäß lexikographischer Ordnung ("Fachverteilen")

Lexikographische Ordnung:

$A = \{a_1, a_2, \ldots, a_n\}$ sei Alphabet mit gegebener Ordnung $a_1 < a_2 < \ldots < a_n$

die sich wie folgt auf A^* fortsetzt:

$v \leq w$ $(v, w \in A^*) \Leftrightarrow$ ($w = v\,u$ mit $u \in A^*$) oder

($v = u\,a_i\,u'$ und $w = u\,a_j\,u''$ mit $u, u', u'' \in A^*$ und $a_i, a_j \in A$ mit $i < j$)

Die antisymmetrische Relation "$\leq$" heißt lexikographische Ordnung.

Sortierung erfolgt in k Schritten:

- Sortierung nach der letzten Stelle
- Sortierung nach der vorletzten Stelle

  ...
- Sortierung nach der ersten Stelle

Beispiel: Sortierung von Postleitzahlen

67663, 04425, 80638, 35037, 55128, 04179, 79699, 71672

Sortierwettbewerbe

Wettbewerb berücksichtigt

- Sortieralgorithmus
- Systemsoftware
- Hardware

Und hat die Aufgabe, schnelle Sortiermethoden zu finden.

Erster Wettbewerb: ***Datamation: 1 April, 1985***

- Sort a million 100-byte records where the first 10-bytes are the key.
- tests file system, IO system, utility access
- Was 1 hour, now 1 second!

Datamation-Ergebnisse 1985-1999

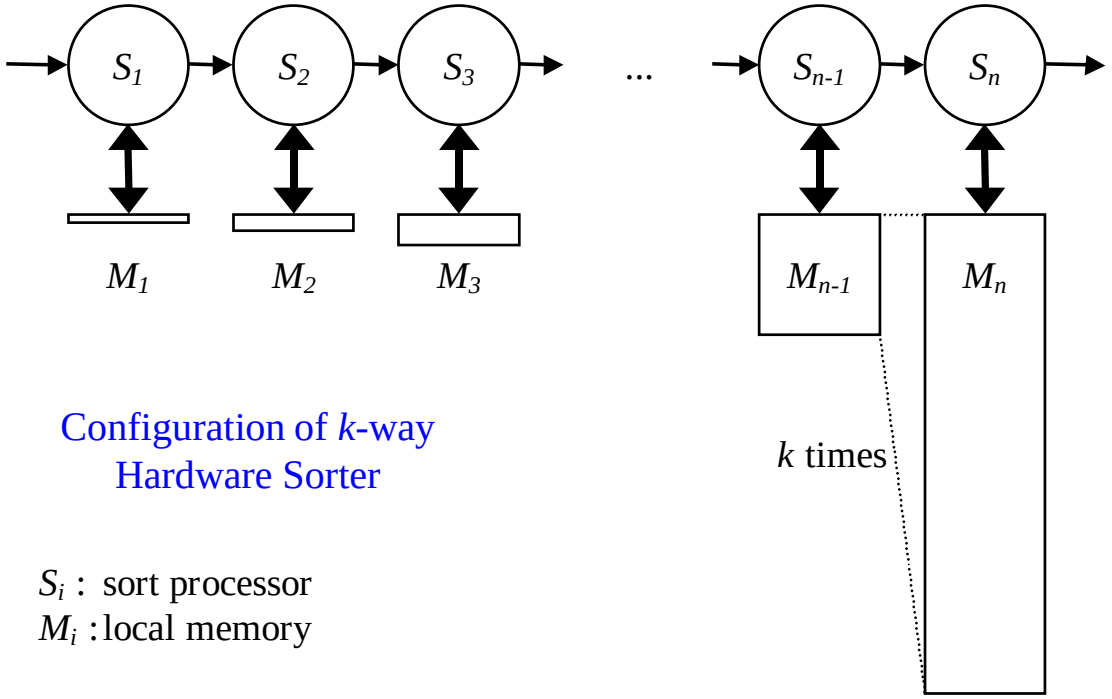
1987	980sec
1988	28 sec
1993	9 sec
1994	7 sec
1996	4.2 sec
1997	2.4 sec
1999	1.18 sec
2000	.998 sec
2001	.44 sec

Gewinner 2000

Mitsubishi DIAPRISM Hardware Sorter with

- HP 4 x 550MHz Xeon PC server + 32 SCSI disks,
- Windows NT4
- 32 PC Linux-Cluster

Mitsubishi DIAPRISM Hardware Sorter

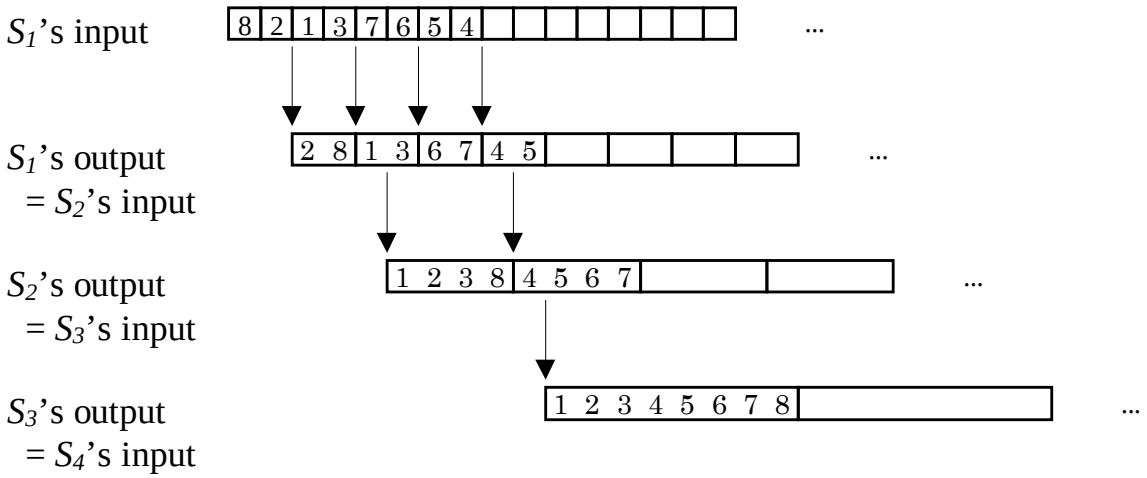


Configuration of k-way Hardware Sorter

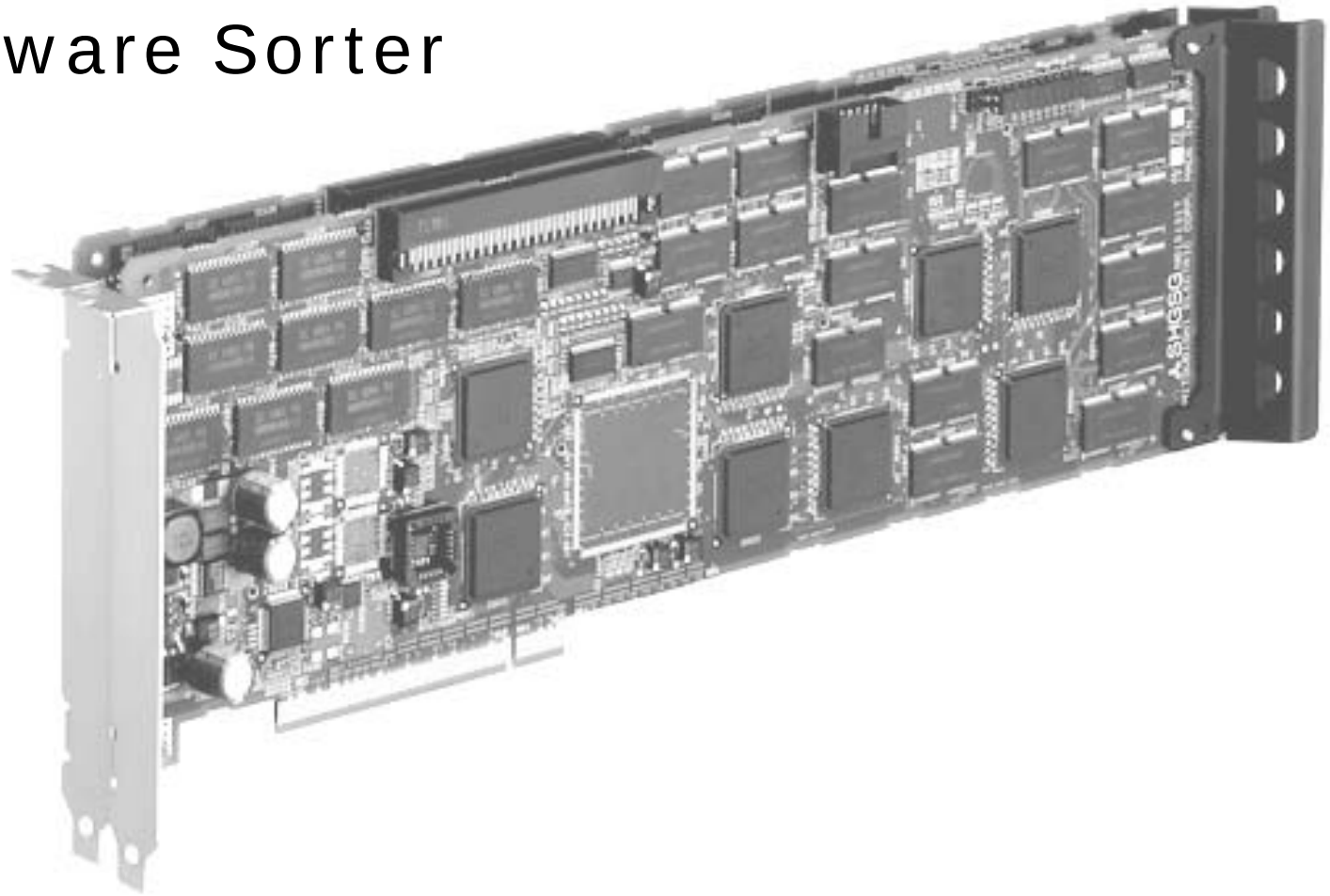
S_i : sort processor
M_i : local memory

Algorithmus:

2-way Pipeline Merge Sort



Mitsubishi DIAPRISM Hardware Sorter



The DIAPRISM hardware sorter board has eight sort processors and one 64-bit 33-MHz PCI interface. As 1GB memory is connected to the eighth sort processor, the sorter can handle up to 16 million ($=8^8$) records or 1.1GB data.

Neue Sortierwettbewerbe

- Minute Sort:
  - Sortiere soviel 100-byte records wie möglich in einer Minute
- Penny Sort:
  - Sortiere soviel wie möglich für einen Penny
- Terabyte Sort:
  - Sortiere1 Terabyte so schnell wie möglich
  - (geplant ab 2009 100 Terabyte Sort)
- Joule Sort:
  - Sortiere 1 Terabyte so energieeffizient wie möglich

2 Kategorien:

- Daytona (stock car): nur kommerziell erhältliche Standardsysteme sind erlaubt
- Indy (formula 1): speziell konstruierte Systeme sind erlaubt

Ergebnisse 2008

	Daytona	Indy
Penny	1,812 M records (181 GB) in 2,408 seconds on a 2.4 Ghz AMD Athlon 64, 2 GB RAM, 4x160GB SATA disks, Linux Paolo Bertasi, Marco Bressan and Enoch Peserico Univ. Padova, Italy	1,896 M records (190 GB) in 2,408 seconds on a 2.4 Ghz AMD Athlon 64, 2 GB RAM, 4x160GB SATA disks, Linux Paolo Bertasi, Marco Bressan and Enoch Peserico Univ. Padova, Italy
Minute	214 GB (2140 million records) On a tx2500 disk cluster, 400 nodes x (2 processors, 6-disk RAID, 8 GB memory) Bradley C. Kuszmaul, MIT	264 GB (2640 M records) On a tx2500 disk cluster, 400 nodes x (2 processors, 6-disk RAID, 8 GB memory) Bradley C. Kuszmaul, MIT
TeraByte	209 seconds (3.48 minutes) on a 910 nodes x (4 dual-core processors, 4 disks, 8 GB memory) Owen O'Malley, Yahoo	197 seconds (3.28 minutes) on a tx2500 disk cluster, 400 nodes x (2 processors, 6-disk RAID, 8 GB memory) Bradley C. Kuszmaul, MIT

Historischer Preis-Leistungs-Vergleich

Table 3: Historical Performance/Price results.						
year	MB/sec	GB/$	System	Sys price (M$)	CPUs	
1985	0.02	0.05	M6800 Bitton et al	0.03	1	Datamation
1986	0.03	0.01	Tandem Tsukerman	0.3	3	Datamation
1987	3.85	0.05	Cray YMP, Weinberger	7.0	1	Datamation
1991	14.29	0.54	IBM 3090, DFsort/Saber	2.5	1	Datamation
1990	0.31	0.15	Kitsuregawa	0.2	1	Datamation
1993	1.20	0.11	Sequent, Graefe	1.0	32	Datamation
1994	1.72	0.16	IPSC/Wisc DeWitt	1.0	32	Datamation
1994	11.11	5.25	Alpha, Nyberg	0.2	1	Datamation
1995	28.57	2.70	SGI/Ordinal, Nyberg	1.0	16	Minute/Daytona
1995	19.61	37.10	IBM, Agarwal	0.05	1	Minute/Indy
1996	100.00	15.76	NOW, Arpaci-Dusseau	0.6	32	Minute/Indy
1997	140.17	8.41	Now 95 , Arpaci-Dusseau	2.0	95	Minute/Indy
1997	86.21	6.27	SGI/Ordinal, Nyberg	1.3	14	Minute/Daytona
1998	1.74	125.00	PostmanSort	0.0013	1	Penny/Daytona
1998	1.74	144.00	NTSort	0.0012	1	Penny/Indy
1999	2.23	174.99	Postman Sort	0.0012	1	Penny/Daytona
1999	2.46	220.59	NTSort	0.0010	1	Penny/Indy
1999	3.51	314.51	HMSort	0.0010	1	Penny/Indy
1999	3.78	338.17	HMSort Post-April 1st	0.0010	1	Penny/Indy
2000	6.50	608.86	HMSort	0.0010	1	Penny/Indy
2001	6.50	608.86	HMSort	0.0010	1	Penny/Indy
2002	**8.64**	**1165.7**	**DMSort**	**0.000672**	**1**	**Penny/Indy**
2002	**10**	**1079.5**	**THSort**	**0.000857**	**1**	**Penny/Daytona**