Kap. 3.3: Externe Sortierverfahren

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

9. VO *TEIL1* DAP2 SS 2009 14. Mai 2009

1. Übungstest

- **Termin:** Di 19. Mai 2009, Beginn: 12:15 Uhr (bitte um 12:00 Uhr anwesend sein)
- **Ort:** im Audi-Max (statt Vorlesung)
- **Dauer:** 30 Minuten
- **Stoff:** aus VO-Folien, Skript und Übungen bis Heap-Sort inklusive 3.1.6 Realisierung von Priority Queues durch Heaps

- Ab ca. 12:50 Uhr: Vorlesung bis 13:45 Uhr

Motivation

„Warum soll ich hier bleiben?“

Externspeicher: Neue Denkweise!!!

„Warum soll mich das interessieren?“

Externe Algorithmen werden immer wichtiger!!!

Überblick

- Motivation Externspeicher-Algorithmen
- Das Externspeichermodell
- I/O Analyse von MergeSort und HeapSort
- k-Multiway MergeSort

Externspeicheralgorithmen

- Motivation
- Das Externspeichermodell

Durchwandern eines Arrays

for i=1,…,N: D[i]:=i

C:=Permute(D)

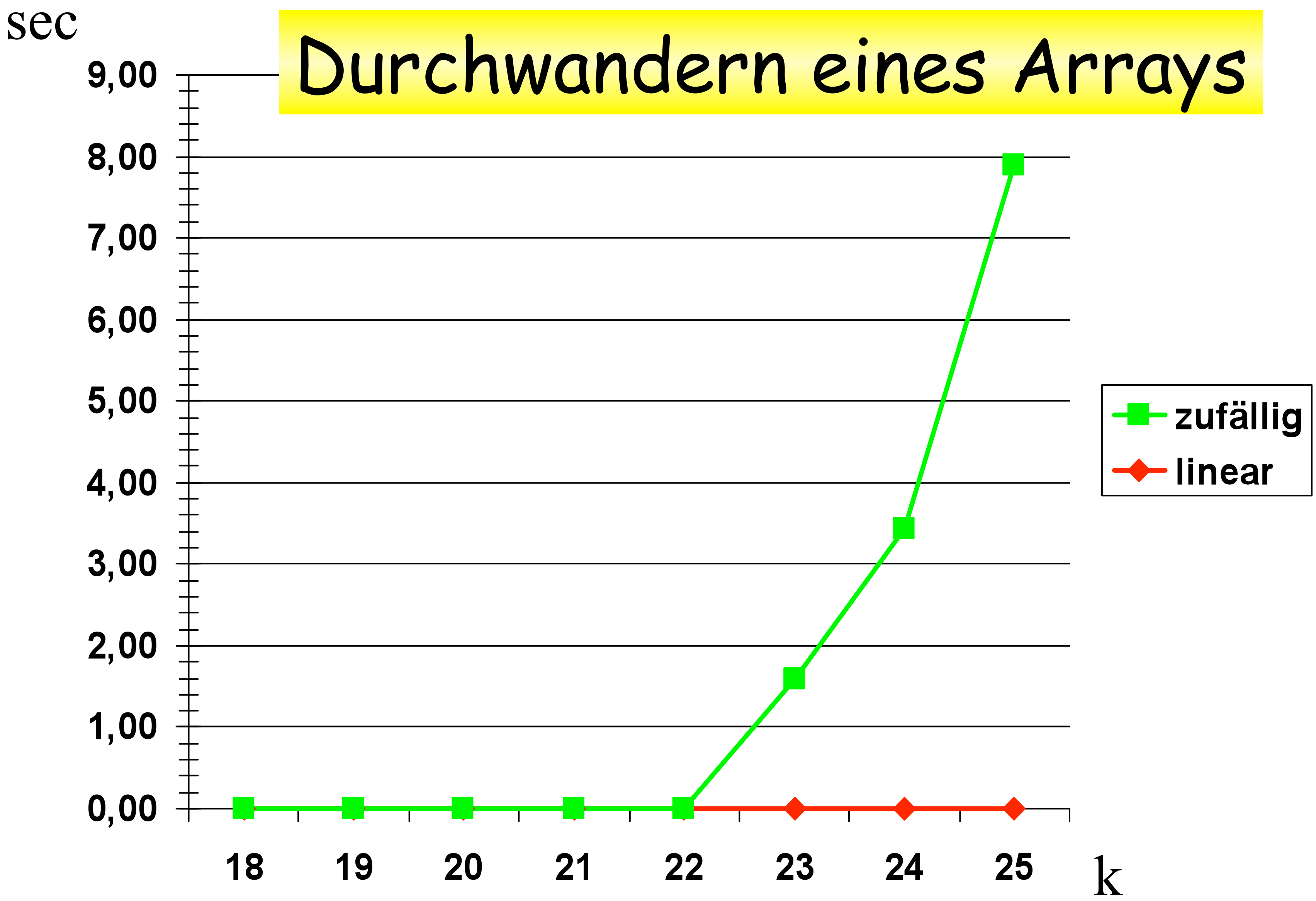
Lineares Durchlaufen:

for i=1,…,N: A[D[i]]:=A[D[i]]+1

Zufälliges Durchlaufen:

for i=1,…,N: A[C[i]]:=A[C[i]]+1

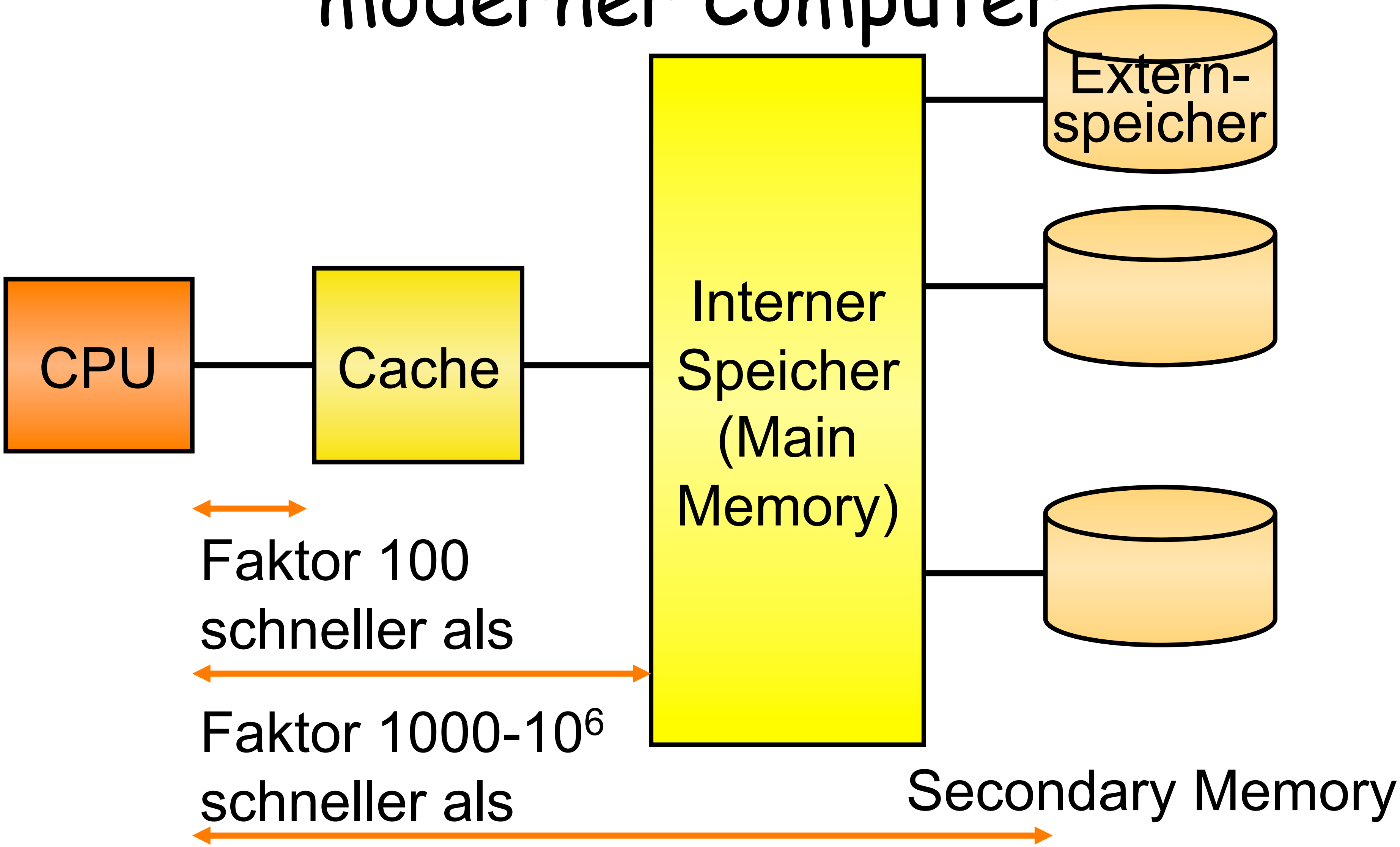
Durchwandern eines Arrays



Größe 2^k $2^{23}=8.388.608$

Rechner: CPU 2.4 GHz mit Cache 512 KB: für $N=2^{25}$: 0,39 Sek. vs. 7,89 Sek.

Hierarchisches Speichermodell moderner Computer



Probleme klassischer Algorithmen

- Ein Zugriff im Hauptspeicher spricht jeweils eine Speicherzelle an und liefert jeweils eine Einheit zurück

- Ein Zugriff im Externspeicher (ein I/O) liefert jeweils einen ganzen Block von Daten zurück

- Meist keine Lokalität bei Speicherzugriffen, und deswegen mehr Speicherzugriffe als nötig

Problem ist aktueller denn je, denn

- Geschwindigkeit der Prozessoren verbessert sich zwischen 30%-50% im Jahr;
- Geschwindigkeit des Speichers nur um 7%-10% pro Jahr

- „One of the few resources increasing faster than the speed of computer hardware is the amount of data to be processed.“

Donald E. Knuth: The Art of Computer Programming 1967 (Neuauflage 1998):

When this book was first written, magnetic tapes were abundant and disk drives were expensive. But disks became enormously better during the 1980s,... . Therefore the once-crucial topic of patterns for tape merging has become of limited relevance to current needs. Yet many of the patterns are quite beautiful, and the associated algorithms reflect some of the best research done in computer science during ist early years;

The techniques are just too nice to be discarded abruptly onto the rubbish heap of history. ...

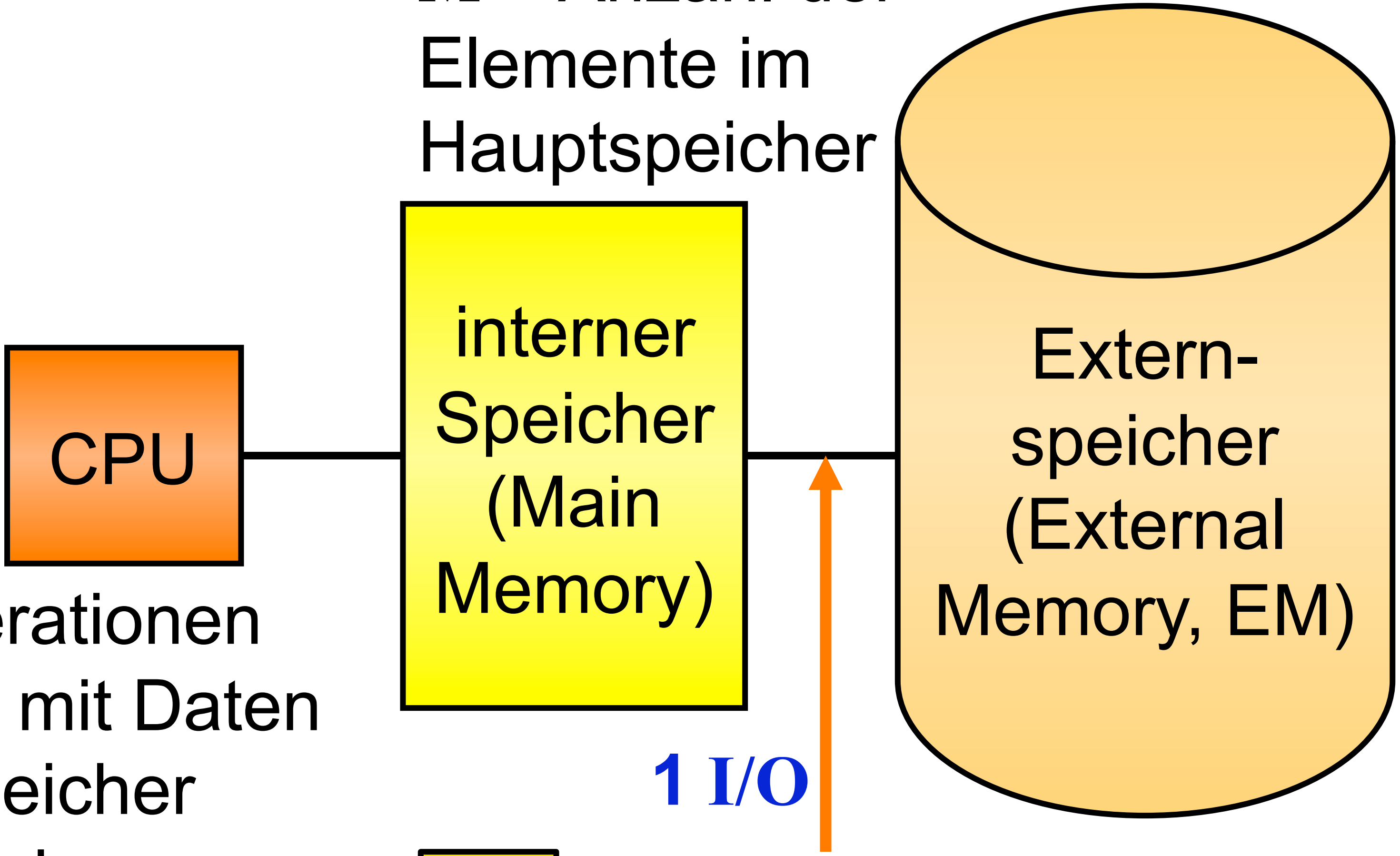
Therefore merging patterns are discussed carefully and completely below, in what may be their last grand appearance before they accept a final curtain call.

Pavel Curtis in Knuth:``The Art of Computer Programming´´ 1967 (Neuauflage 1998):

For all we know now, these techniques may well become crucial once again.

Das Externspeichermodell

M = Anzahl der Elemente im Hauptspeicher



Rechenoperationen können nur mit Daten im Hauptspeicher getätigt werden

Annahme: $B<M/4$

B = Anzahl der Elemente, die in einen Block passen

Analyse von Externen Algorithmen

- Anzahl der ausgeführten I/O-Operationen

- Anzahl der ausgeführten CPU-Operationen im RAM-Modell

- Anzahl der belegten Blöcke auf dem Sekundärspeicher

Ziele beim Entwurf Externer Algorithmen

- **Interne Effizienz:**
  - Anzahl der RAM-Operationen vergleichbar zu den besten internen Algorithmen

- **Örtliche Lokalität:**
  - Ein r/w Block sollte möglichst viele nützliche Daten enthalten

- **Zeitliche Lokalität:**
  - Daten, die im internen Speicher sind, sollten möglichst verarbeitet werden, bevor sie wieder herausgeschrieben werden.

I/O-Komplexität im EM-Modell

- Einlesen einer Menge von N Elementen benötigt mindestens $\Theta(N/B)$ I/O's

- I/O-Komplexität von HeapSort: $\Theta(???)$

I/O-Komplexität des HeapSort

Create-Heap():

- O(N) I/Os

HeapSort():

- Anzahl der I/Os pro Schleifendurchgang: O(1+log N) I/Os
- über alle N: O(N log N) I/Os

I/O-Komplexität im EM-Modell

- Einlesen einer Menge von N Elementen benötigt mindestens $\Theta(N/B)$ I/O‘s

- I/O-Komplexität von HeapSort: $\Theta(N \log N)$ I/O‘s

- I/O-Komplexität von MergeSort:

Zur Erinnerung: Merge-Sort

MergeSort(ref A,l,r)

```
procedure MergeSort(ref A,l,r)
  var Index m
  if l < r then {
      m:=⌊(l+r)/2⌋
      MergeSort(A,l,m)
      MergeSort(A,m+1,r)
      Merge(a,l,m,r)
  }
```

Aufruf: MergeSort(A,1,n)

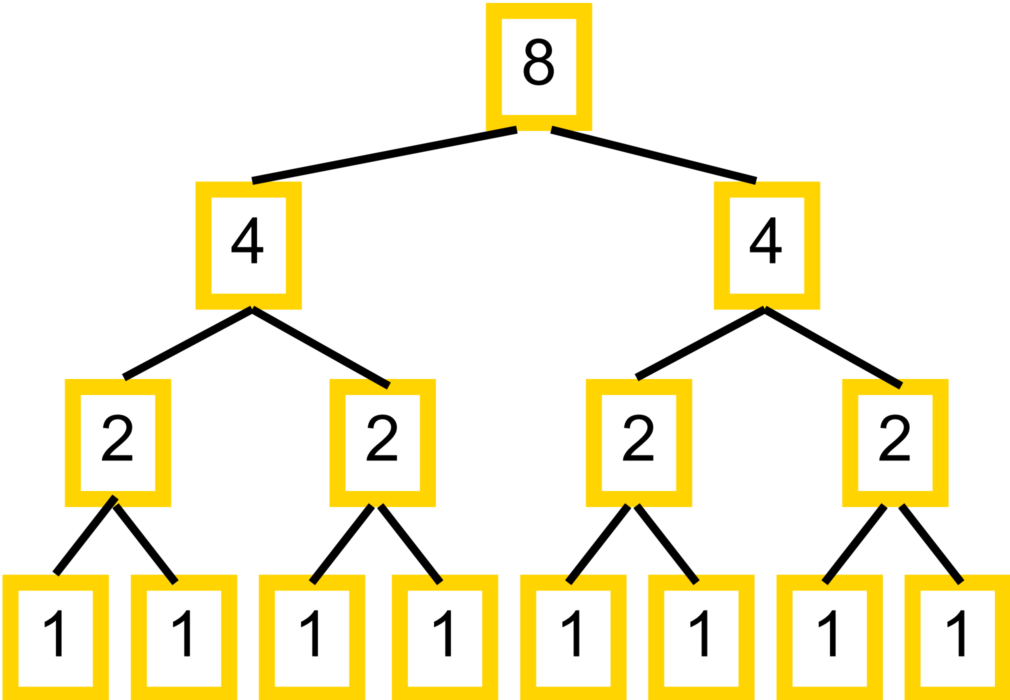
Merge(ref A,l,m,r)

```
procedure Merge(ref A,l,m,r)
  i:=l; j:=m+1
  for k:=l,…,r {
      if (i>m) or ((j≤r) and (A[i].key>A[j].key))
          then
        B[k]:=A[j]; j:=j+1
    else
        B[k]:=A[i]; i:=i+1
  }
Schreibe sort. Folge zurück von B nach A
```

Herleitung der Laufzeitfunktion

Sei $n=2^k$ für ein beliebiges k

hier: $n=8$, $k=3$:



Anzahl der Instanzen	Zeit pro Instanz	Gesamtzeit
$1 = 2^0$	$8 = 2^3$	$8 = 2^3$
$2 = 2^1$	$4 = 2^2$	$8 = 2^3$
$4 = 2^2$	$2 = 2^1$	$8 = 2^3$
$8 = 2^3$	$1 = 2^0$	$8 = 2^3$

Aufwand in jeder Stufe gleich $n=2^k$.

Es gibt $k+1=\log n + 1$ solcher Stufen

I/O-Komplexität des MergeSort

Merge-Phase:

- Verschmelzen der Teilfolgen S_1 und S_2: $O(1+(|S_1|+|S_2|)/B)$ I/Os
- Anzahl der I/Os auf einer Stufe: $O(N+N/B)$ I/Os
- über alle Schichten: $O((N+N/B) \log N)$ I/Os

Dies kann man leicht verbessern!

Externer Merge-Sort

1. Verbesserung: Verhindere in der Run-Formationsphase 1-elementige Mengen!

- Beende die Aufteilung bei Teilmengen der Länge M
- Lade die N/M Stücke nacheinander in das Main Memory
- Sortiere diese im Main Memory
- Schreibe die sortierte Teilfolge zurück nach EM

Externer Merge-Sort

- Beende die Aufteilung bei Teilmengen der Länge M
- Lade die N/M Stücke nacheinander in das Main Memory
- Sortiere diese im Main Memory
- Schreibe die sortierte Teilfolge zurück nach EM

- Diese Aufteilung kostet zusätzlich: $O((N/M)(M/B+1)) = O(N/B)$ I/Os für das Hin- und Herkopieren (das Sortieren an sich kostet kein I/O)
- Anzahl der I/Os auf einer Stufe: $O(N/M+N/B)$ I/Os
- Über alle Stufen: $O((N/M+N/B) \log (N/M))$ I/Os
- Dies ist gleich $O(N/B(1+\log(N/B)))$

denn: $N/M \leq N/B$

Externer Merge-Sort

2. Verbesserung: Verschmelze statt 2 jeweils $p=M/(2B)$ Runs (Run = aufsteigend sortierte Teilfolgen)

- Kopiere die jeweils kleinsten Elemente $x_{1,}\ldots,x_p$ der Runs $S_1,\ldots,S_p$ in das Main Memory (Blockzugriff)
- Kopiere das Minimum x_i der Elemente in den Output Run
- Lese das nächste Element von $S_{i\,,}$ u.s.w.

Externer Merge-Sort

- Kopiere die jeweils kleinsten Elemente $x_{1,}\ldots,x_p$ der Runs $S_1,\ldots,S_p$ in das MM (mittels Blockzugriff)
- Kopiere das Minimum x_i der Elemente in den Output Run
- Lese das nächste Elemente von S_i usw

- Verschmelzen der Teilfolgen $S_1,\ldots,S_p$: $O(p+(|S_1|+\ldots+|S_p|)/B)$ I/Os
- Anzahl der I/Os auf einer Stufe: $O(N/M+N/B)$ I/Os
- Über alle Stufen: $O((N/M+N/B)\log_{M/B}(N/B))$ I/Os
- Dies ist gleich $O(N/B(1+\log_{M/B}(N/B)))$

I/O-Komplexität im EM-Modell

- Einlesen einer Menge von N Elementen benötigt mindestens $\Theta(N/B)$ I/O's

- I/O-Komplexität von HeapSort: $\Theta(N \log N)$ I/O's

- I/O-Komplexität von MergeSort:

$$O(N/B(1+\log_{M/B}(N/B)))\ \text{I/O's}$$

Interne Laufzeit von k-Multiway Merging

- **Phase 1:** Sortiere N/M Stücke der Länge M: $O((N/M)\ (M \log M)) = O(N \log M)$
- pro Schicht: Verschmelze N Elemente inkl. Minimumsuche: $O(???)$

Effiziente Verschmelzung von p Runs?

Intern: Minimumsuche der Schlüssel
$x_{1,} \ldots, x_p$?

- **Naiv:** $O(p)$: zu teuer, denn dann würde die Laufzeit der Merge-Phase $O(p\ N \log_p(N/B))$ sein.
- **Lösung:** Halte die jeweils kleinsten Elemente $x_{1,}$ $\ldots, x_p$ (mit ihren Blockzugriffspartnern) in einer Priority Queue im Main Memory.
- **Laufzeit:** DeleteMin: $\log p$, Insert: $\log p$
- **Speicherplatz:** $Bp=BM/(2B))=M/2$.

Interne Laufzeit von k-Multiway Merging

- **Phase 1:** Sortiere N/M Stücke der Länge M: $O((N/M)\ (M \log M)) = O(N \log M)$
- pro Schicht: Verschmelze N Elemente inkl. Minimumsuche: $O(N \log p)$
- Tiefe des Baumes: $O(\log_p(N/B))$
- **Insgesamt**: $O((N \log M)+(N \log p) \log_p(N/B)) =$ $O((N \log M)+(N \log (M/B) \log_{M/B}(N/B))=O(N \log N)$
- Verwende Logarithmus-Gesetz: $\log_p a=(\log_2 a)/(\log_2 p)$

Externes Sortieren

- **Theorem:** Eine Menge von N Elementen kann mit Hilfe von k-Multiway Merging in interner Zeit $O(N \log N)$ mit $O(N/B(1+\log_{M/B}(N/B)))$ I/Os sortiert werden.

Man kann zeigen: Dies ist bestmöglich!

Bemerkung: Bei I/O schreibt man oft „+1“ innerhalb der Oh-Notation, weil sonst aus $\log (N/B)=0$ folgen würde: $O(N/B\ 0)=O(0)$ I/Os, obwohl man jedes Mal ein I/O braucht, also die echte I/O-Anzahl $O(N/B)$ wäre.

ENDE Kapitel 3: Sortieren und jetzt: Kapitel 4: Suchen