Kap. 1: Priority-Queues
1.2 Externe Array-Heaps

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

4.-6. VO A&D WS 08/09 23./28./30. Oktober 2008

Literatur für diese VO

Andreas Crauser: LEDA-SM: External Memory Algorithms and Data Structures in Theory and Praxis. Dissertation, Max-Planck-Institut für Informatik, Saarbrücken, 2001. Kapitel 4: Priority Queues; http://www.mpi-sb.mpg.de/~crauser/degrees.html

- hierzu gibt es auch ein ausgearbeitetes Skriptum auf unserer VO Web-Seite

- Schönes Buch zum Thema (für Interessierte):
- U. Meyer, P. Sanders und J. Sibeyn (Eds.), Algorithms for Memory Hierarchies, Advances Lectures, Lecture Notes in Computer Science 2625, Springer 2003

Überblick

1.2.1 Einführung
- Motivation
- Das Externspeichermodell
- Exkurs: Grundlegende externe Datenstrukturen

1.2.2 Externe Array Heaps (Prioritätswarteschlange)
- Modell, Operationen und Realisierung
- Analyse

1.2.1 Einführung

Durchwandern eines Arrays:

```
for (i=0; i<N; i++) D[i]=i
C=Permute(D)
```

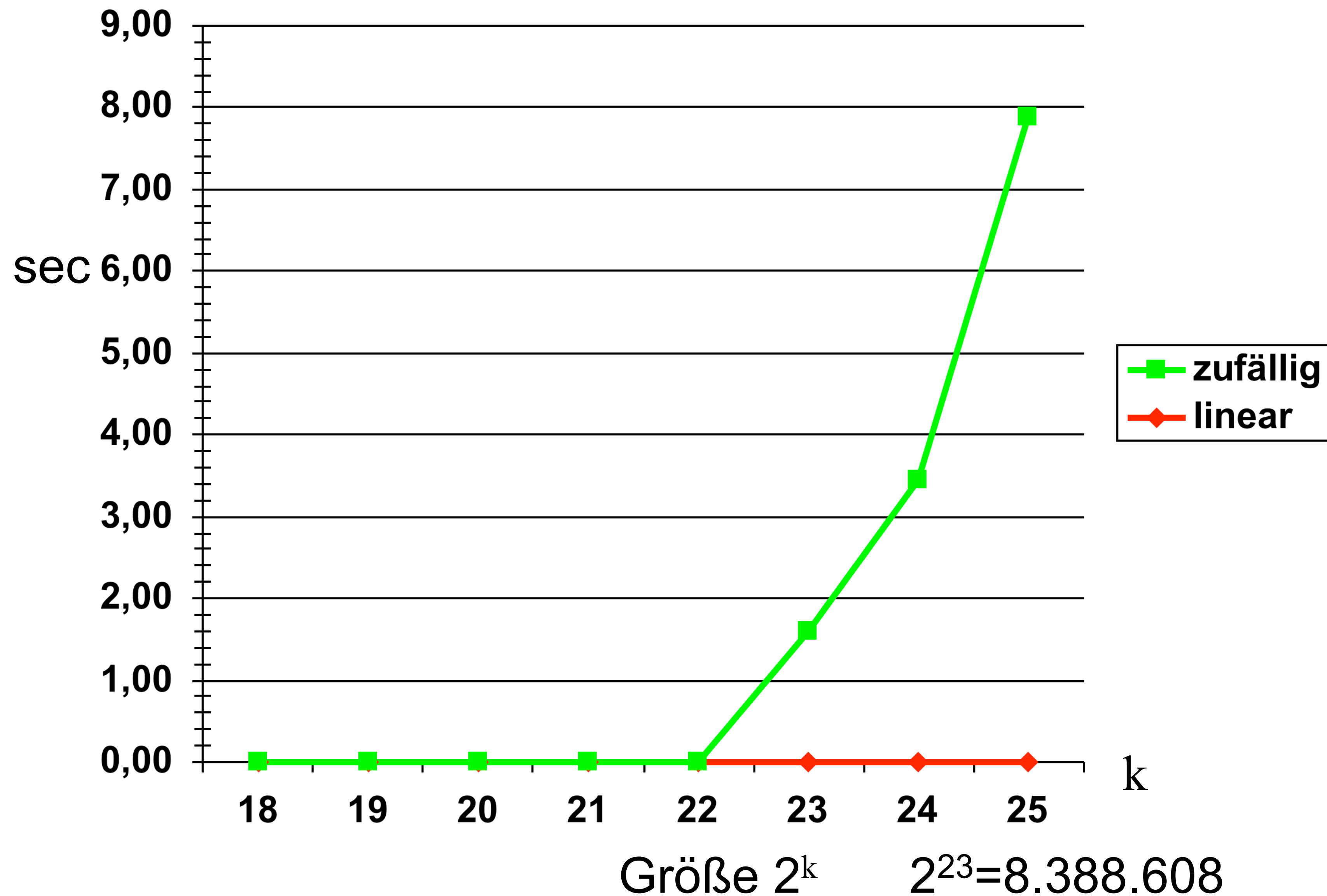
Lineares Durchlaufen:

```
for (i=0; i<N; i++) A[D[i]]=A[D[i]]+1
```

Zufälliges Durchlaufen:

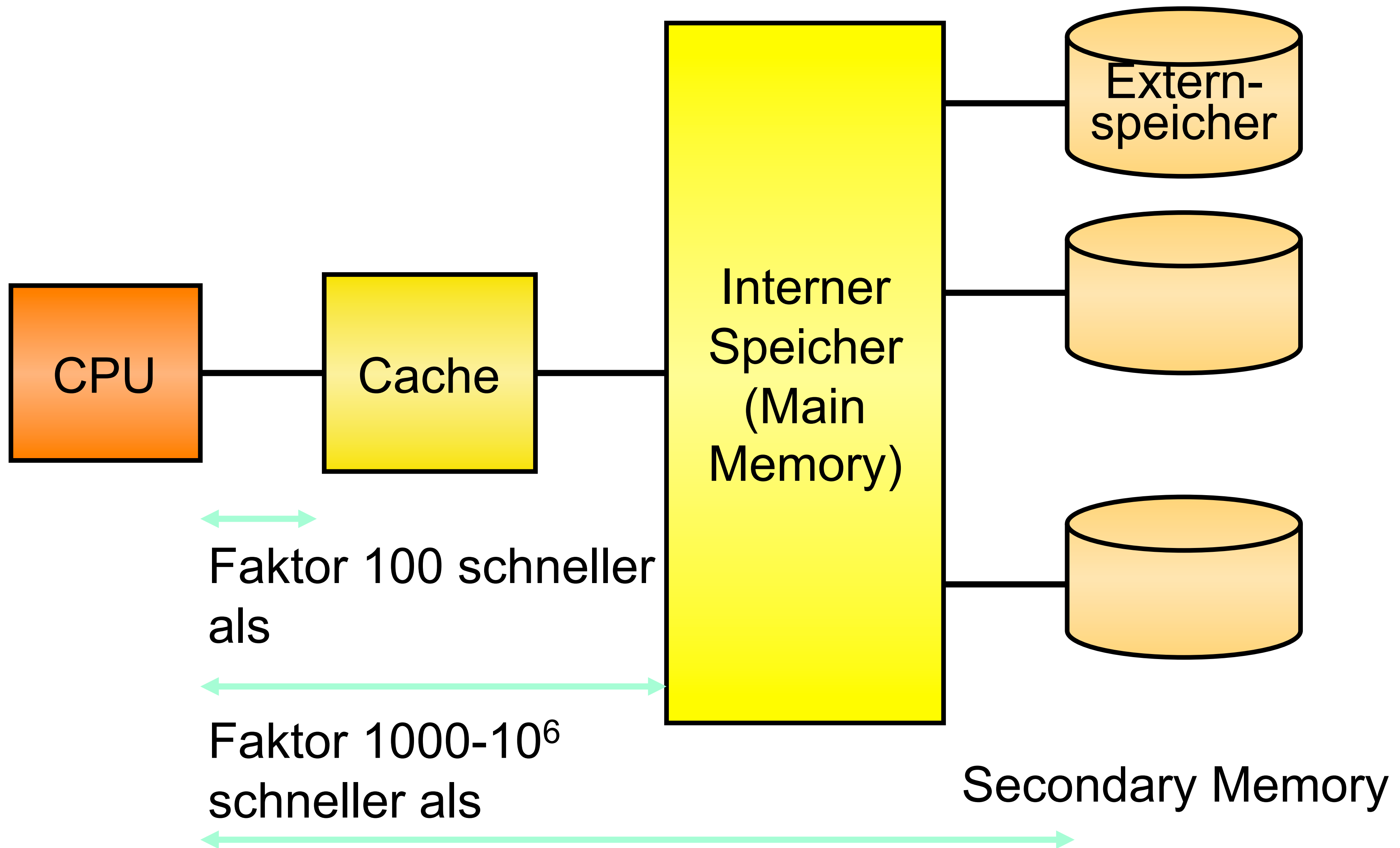
```
for (i=0; i<N; i++) A[C[i]]=A[C[i]]+1
```

Durchwandern eines Arrays



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 its 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.

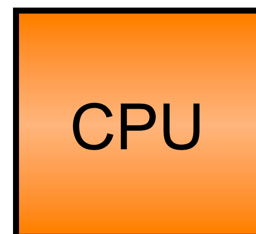
Pavel Curtis in Knuth: The Art of Computer Programming 1967 (Neuauflage 1998):

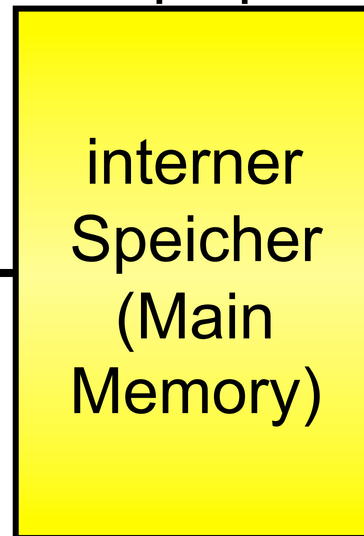
- For all we know now, these techniques may well become crucial once again.

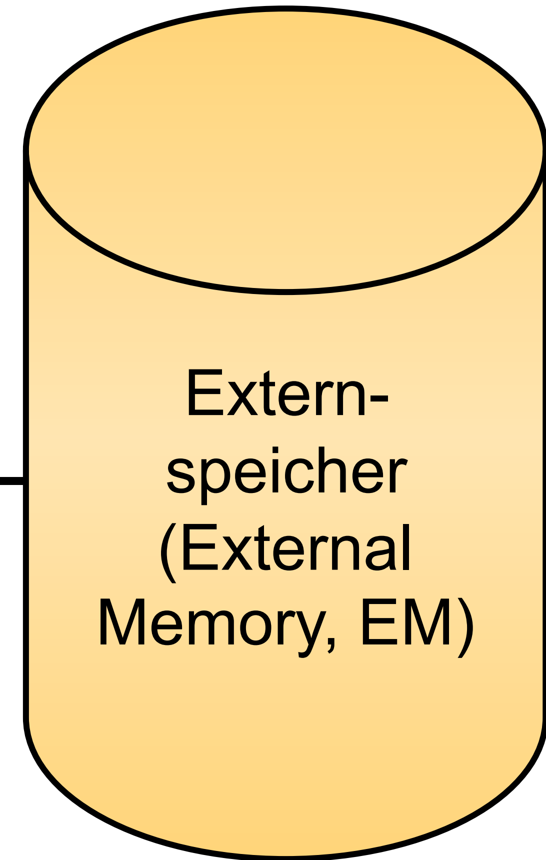
Das Externspeichermodell

Modell von Aggarwal,Vitter und Shriver 1994

M = Anzahl der Elemente im Hauptspeicher

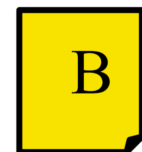






1 I/O

Rechenoperationen können nur mit Daten im Haupt-speicher getätigt werden



= Anzahl der Elemente, die in einen Block passen

Annahme: $B<M/2$

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.

Externe Datenstrukturen: Stacks

- Ein Stack S repräsentiert eine dynamische Menge von Elementen (maximal N)

- **Operationen:**
  - Insert(x): Einfügen eines neuen Elements in S
  - Delete: Ausgabe und Entfernung des letzten eingefügten Elements aus S

Interne Algorithmen für Stack der Größe N:
- Array der Länge N und zwei Zeiger

Im Worst Case: 1 I/O per Insert und Delete Operation

Externe Stacks

- **Buffer für externe Stacks:**
  - Array im internen Speicher der Länge $2B$
  - enthält zu jedem Zeitpunkt die letzten k eingefügten Elemente, wobei $k \leq 2B$
- **Insert(x):**
  - Meistens 0 I/Os benötigt, außer: wenn Buffer voll ist
  - Dann: die B ältesten Elemente werden in EM ausgelagert.
- **Delete:**
  - Meistens 0 I/Os benötigt, außer: wenn Buffer leer
  - Dann: 1 I/O holt die nächsten B Elemente aus dem EM (diejenigen, die als letztes ausgelagert wurden)

Externe Stacks

- **Analyse der Operationen:**
  - Insert(x): 1/B I/Os amortisiert
  - Delete: 1/B I/Os amortisiert

- **Dies ist bestmöglich!**
- Denn: mit einer I/O können nicht mehr als B Elemente gleichzeitig gespeichert oder gelesen werden

Externe Datenstrukturen: Queues

- **Operationen:**
  - Insert(x): Einfügen eines neuen Elements in S
  - Delete: Ausgabe und Entfernung des ältesten Elem. aus S

- **Zwei Buffer: R und W Buffer:**
  - Zwei Arrays im internen Speicher der Länge jeweils B

- **Insert(x):**
  - zu W Buffer, außer: wenn voll
  - Dann: schreibe alle B Elemente in EM

Analyse: 1/B I/Os

- **Delete:**
  - aus R Buffer, außer: wenn Buffer leer
  - Dann: 1 I/O holt die nächsten B Elemente aus dem EM (wenn keine dort, dann aus W Buffer (Buchführung!))

Analyse: 1/B I/Os

Externe Datenstrukturen: Lineare Listen

s. Übung

Untere Schranken im EM-Modell

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

- Sortieren einer Menge von N Elementen benötigt mindestens

  $\Theta(N/B \log_{1+M/B} (1+N/B))$

- Suche in dynamischen Daten von N Elementen benötigt mindestens Zeit

  $\Theta(\log N / \log B)$ I/O-Operationen

1.2.2 Externspeicherdatenstruktur für Prioritätswarteschlangen

- Dynamische Datenstruktur für Elemente:
  Schlüssel + Information

- **Operationen:**
  - Get_Min: Ausgabe der Elemente mit kleinstem Schlüssel
  - Del_Min: Ausgabe und Entfernung des kleinsten Elements
  - Insert: Einfügen eines neuen Elements

Welche Datenstrukturen kennen Sie dafür?

Externe Array-Heaps

- Im internen Arbeitsspeicher: Heap

- Im externen Speicher: Menge von sortierten Feldern unterschiedlicher Länge

Externe Array-Heaps

Lemma 1: $l_{i+1}=l_i\,(\mu+1)$

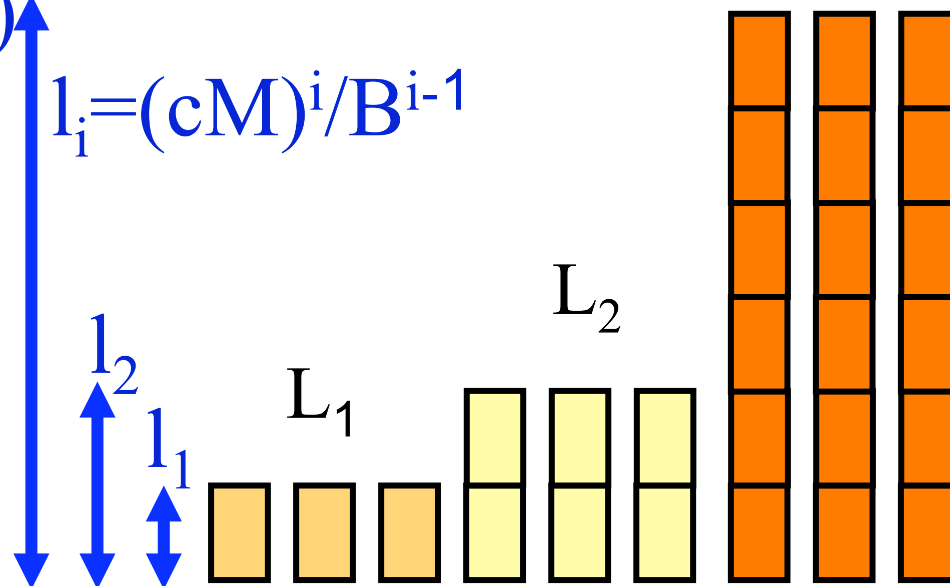
$l_1=cM$

$l_2=(cM)^2/B$

$=cM(\mu+1)$

L Schichten L_i L_3

jeder Slot enthält sortierte Folge oder ist leer



$\mu=(cM/B)-1$ μ μ

c=1/7

L<=4

Die Anzahl der Plätze in einem Slot von L_{i+1} entspricht der Anzahl aller Plätze in L_i plus l_i

Lemma 1: $l_{i+1}=l_i(\mu+1)$

$l_1=cM$

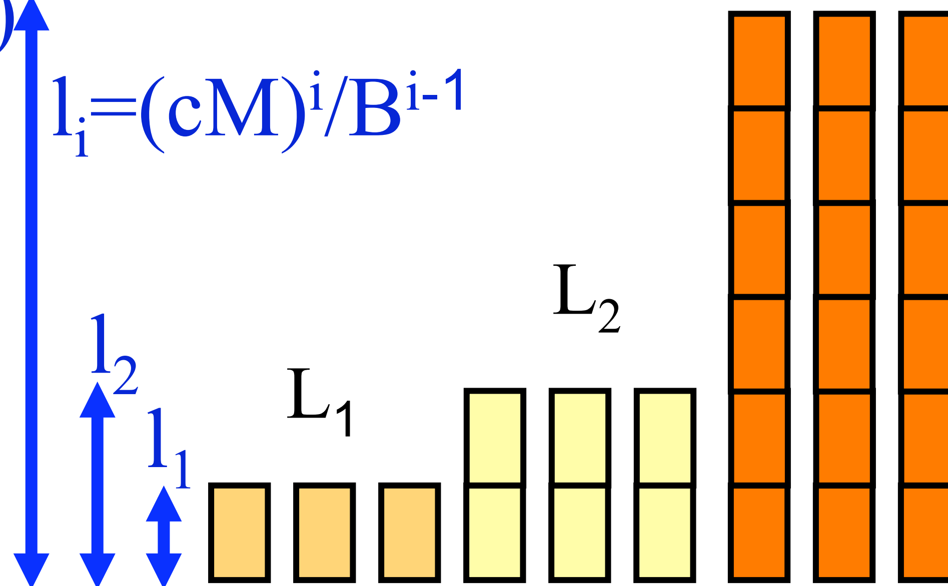
$l_2=(cM)^2/B$

$=cM(\mu+1)$

L Schichten L_i

L_3

jeder Slot enthält sortierte Folge oder ist leer



$c=1/7$

$L<=4$

$\mu=(cM/B)-1$ μ μ

Operation Insert

- Fügt neue Elemente immer in den internen Heap H ein

- Falls kein Platz mehr in H ist, dann werden vorher $l_1 = cM$ dieser Elemente in den Sekundärspeicher bewegt:
  - Falls freier Slot in L_1 existiert, dann werden diese Elemente in sortierter Folge dorthin bewegt
  - Sonst: Alle Elemente in L_1 werden mit den neuen Elementen aus H zu einer sortierten Liste gemischt, die dann in einen freien Slot von L_2 geschrieben werden.
  - Falls L_2 auch kein freier Slot, wiederhole L_3,...bis frei

Operation Del_Min

- **Invariante:** Das kleinste Element befindet sich immer in H

- Dazu: Heap wird in zwei Heaps geteilt: H_1 und H_2:
  - H_1 enthält immer die neu eingefügten Elemente, maximal $2cM$
  - H_2 speichert maximal die kleinsten B Elemente in jedem belegten Slot j in jeder Schicht L_i

- **Lemma 2:** Es befinden sich maximal $cM(2+L)$ Elemente im Hauptspeicher

$2cM+B\mu L < 2cM+B(cM/B)L$

- Zusätzlich wird $(\mu+1)B=cM$ gebraucht, um die μ Slots plus eine Overflow Folge zu mischen

Es muss gelten: $M \geq cM(3+L)$;
Daraus folgt: bei $c=1/7$ => $L \leq 4$

- Invariante: Das kleinste Element befindet sich immer in H

- Dazu: Heap wird in zwei Heaps geteilt: H_1 und H_2:
  - H_1 enthält immer die neu eingefügten Elemente, maximal $2cM$
  - H_2 speichert maximal die kleinsten B Elemente in jedem belegten Slot j in jeder Schicht L_i

- **Lemma 2:** Es befinden sich maximal $cM(2+L)$ Elemente im Hauptspeicher

- Zusätzlich wird $(\mu+1)B=cM$ gebraucht, um die μ Slots plus eine Overflow Folge zu mischen

Operationen (1)

- **Merge-Level (i,S,S´):**
  - produziert eine sortierte Folge S' durch das Mischen der sortierten Folge der μ Slots in L_i (inkl. der ersten Blocks in H_2) und der sortierten Sequenz S.
  - Analyse: $O(l_{i+1}/B)$ I/O´s

- **Store(i;S):**
  - Annahme: L_i enthält einen leeren Slot und die Folge S besitzt Länge im Bereich $[l_i/2 , l_i]$
  - S wird in einen leeren Slot von L_i gespeichert und seine kleinsten B Elemente werden nach H bewegt.
  - Analyse: $O(l_i/B)$ I/O´s

Operationen (2)

- **Load (i,j):**
  - Holt die nächsten B kleinsten Elemente vom j-ten Slot aus L_i in den internen Heap H_2.
  - Analyse: $O(1)$ I/O´s

- **Compact(i):**
  - Annahme: es existieren mind. 2 Slots in L_i, mit Gesamtzahl an Elementen (inkl. H_2), höchstens l_i.
  - Diese beiden Slots werden gemischt, und in einen freien Slot von L_i eingetragen. Damit wird ein Slot in L_i frei.
  - Analyse: $O(l_i/B)$ I/O´s

Operation Insert

- Fügt neue Elemente immer in den internen Heap H ein

- Falls kein Platz mehr in H_1 ist, dann werden die größten $l_1=cM$ Elemente nach L_1 bewegt (und die kleinsten B davon nach H_2):
  - Falls freier Slot in L_1 existiert, dann wird Store(1,S) aufgerufen
  - Sonst: Alle Slots in L_1 (bis auf einen) enthalten mindestens $l_1/2$ Elemente: Merge-Level(1,S,S´)
  - Falls freier Slot in L_2 existiert, dann: Store(2,S´)
  - Sonst: wiederhole L_3,...bis frei.

Operation Del_Min

- Das kleinste Element wird vom internen Heap entfernt (H_1 oder H_2).

- Falls in H_2: dann korrespondiert dieses zu Slot j einer Schicht L_i.
- Falls es das letzte Element in H_2 ist, das zu j gehört, dann werden die nächsten B Elemente von Slot j nach H_2 mittels $\mathrm{Load}(i,j)$ bewegt.
- Nach jedem $\mathrm{Load}(i,j)$ wird $\mathrm{Compact}(i)$ bei Bedarf aufgerufen

Korrektheit

Lemma 3: Das kleinste Element ist immer in H (H_1 oder H_2).

Lemma 4: Bei der Ausführung von Store(i,S) ist immer garantiert, dass S zwischen $l_i/2$ und l_i Elementen enthält.

Beweis: Betrachte Schicht i-1: Sei a_j die Anzahl der Elemente in Slot j. Wir wissen: $a_j+a_k>l_{i-1}$ für alle Paare j und k
→ Summe über alle Paare:

$$(\mu-1)\sum_{j=1}^{\mu} a_j = \sum_{\substack{j,k=1 \\ j\neq k}}^{\mu} (a_j + a_k) > \frac{\mu(\mu-1)}{2}\, l_{i-1}$$

Hinzu kommt das vorige S mit mindestens $l_{i-1}/2$ Elementen.
Dies sind also zusammen mind. $(\mu+1)\, l_{i-1}/2 = l_i/2$ Elemente.

I/O Schranken

Annahme: cM>3B

Lemma 5: Nach N Operationen existieren höchstens $L \leq \log_{cM/B}(N/B)$ Schichten.

Beweis: Im Worst Case sind alle N Operationen Inserts. Dann wird bei jedem Überlauf die maximal mögliche Anzahl von Elementen in die nächste Schicht bewegt. Die Anzahl der Elemente in j Slot-Schichten ist also:

Elemente in H_1

$$
\begin{aligned}
N - cM &= \sum_{i=1}^{j} l_i \mu = \sum_{i=1}^{j} \frac{(cM)^i}{B^{i-1}} \left(\frac{cM}{B} - 1\right) = \sum_{i=1}^{j} \left(\frac{(cM)^{i+1}}{B^i} - \frac{(cM)^i}{B^{i-1}}\right) = \\
&= \frac{(cM)^{j+1}}{B^j} - cM = \left(\frac{cM}{B}\right)^j cM - cM \iff \\
j &= \lceil \log_{cM/B} \frac{N}{cM} \rceil \text{ und damit gilt} \\
L &\leq \log_{cM/B} \frac{N}{cM}. \text{ Es gilt auch: } N/(cM) < N/(3B) < N/B
\end{aligned}
$$

Speicherplatz im Hauptspeicher

- **Beobachtung 1:** Im Hauptspeicher wird Platz für bis zu $cM(3+L) \leq cM(3+\log_{cM/B}(N/B))$ Elemente benötigt.
- Daraus läßt sich nun eine Maximalgrenze für die Anzahl N an Operationen berechnen, die garantiert, dass alles Platz hat ($M \geq cM(3+\log_{cM/B}(N/B))$ Auflösen nach N und Abschätzungen (o.Bw.)).
- **Beobachtung 2:** Bei einer Folge von bis zu $N \leq B(cM/B)^{(1/c)-3}$ Operationen haben alle Elemente ausreichend Platz.
- **Beispielrechnung:**
- Für $M=10^9$, $B=10^6$ und $c=1/7$, wäre $N \leq 0{,}416\ 10^{15}$ und $L \leq 4$

Annahme: $cM>3B$

I/O Schranken

Lemma 6: Store(i,S) benötigt höchstens $3l_i/B$ I/O´s. Merge-Level(i,S,S´) und Compact(i+1) benötigen höchstens $3l_{i+1}/B$ I/O´s.

Beweis:

Store(i,S): r+w: $2\lceil l_i/B\rceil \leq 2(l_i/B)+2 \leq 3l_i/B$ (wg. $l_i/B>3$ da $l_i \geq l_1 = cM > 3B$)

Merge-Level(i,S,S´): r+w: $2(\mu\lceil (l_i/B)\rceil + |S|/B) \leq$
$\leq 2(\lceil (|S|+\mu l_i)/B\rceil + (\mu+1)) \leq 2(l_{i+1}/B)+2cM/B \leq 3l_{i+1}/B$ (da $l_{i+1} \geq 2cM$)

Compact(i): $\lceil a_j/B\rceil + \lceil a_k/B\rceil + \lceil l_i/B\rceil \leq (a_j+a_k)/B + l_i/B + 3 \leq 2l_i/B+3 \leq 3l_i/B$, wg. $l_i/B>3$

I/O Schranken

Theorem:

Annahme: $cM>3B$ und $0<c<1/3$ und $N \leq B(cM/B)^{(1/c)-3}$

In einer Folge von N Operationen vom Typ Insert und Del_Min benötigt

- Insert amortisiert $18/B\ (\log_{cM/B}(N/B))$ I/O´s und
- Del_Min $7/B$ amortisierte I/O´s.

Die Schranke für N kommt aus Platzbeschränkungen her (s. Beobachtung 2)

Beweis: Amortisierte Analyse (1)

- Insert: $18/B(\log_{cM/B}(N/B)) \geq 18L/B$ amortisierte I/O´s
- Del_Min $7/B$ amortisierte I/O´s.

- Bankkonto-Methode:
  - Jedes Element erhält beim Einfügen ein Guthaben von $18L/B$
  - Wir zeigen: es werden höchstens $18/B$ benötigt um von einer zur anderen Schicht zu wandern
  - Beim Entfernen werden $7/B$ Einheiten im Heap belassen

Beweis: Amortisierte Analyse (2)

- Insert mit Overflow kostet $6l_{i+1}/B$, denn:
  - Merge_level(i,S;S´): kostet $3l_{i+1}/B$ und Store(i+1,S´): $3l_{i+1}/B$
- Wie können diese Kosten bezahlt werden?

- **Fall 1:** Overflow zur Schicht L_1:
- Jedes Element, das nun bewegt wird, gibt von seinem Bankkonto jeweils $12/B$ Einheiten dafür ab; da der Slot in Schicht L_1 mindestens zur Hälfte gefüllt ist, kommen so mind. $(12/B)\,(l_1/2)=6l_1\,/\,B$ Einheiten zusammen.
- (Interpretation: stellen Sie sich vor, jedes Element erhält beim Einfügen $18L/B$ Einheiten; nun möchten die Elemente in die Schicht L_1 wechseln, das kostet aber insgesamt $6l_1/B$. Diese können dadurch aufgebracht werden, indem alle bewegten Elemente jeweils $12/B$ Einheiten von ihrem momentanen Bankkonto abgeben.)

Beweis: Amortisierte Analyse (2)

- Insert mit Overflow kostet $6l_{i+1}/B$, denn:
  - Merge_level(i,S;S´): kostet $3l_{i+1}/B$ und Store(i+1,S´): $3l_{i+1}/B$
- Wie können diese Kosten bezahlt werden?

- **Fall 2:** Overflow von Schicht L_i nach L_{i+1}:
- Jedes Element hatte anfangs $18L/B$ Einheiten zur Verfügung; das sind für Schicht i genau $18/B$ Einheiten, die das Element verbrauchen kann. Da fast alle Slots von Schicht L_i mind. zur Hälfte gefüllt sind, können je $12/B$ Einheiten von den Bankkonten der bewegten Elemente genommen werden.

- **Beob.:** Damit hat jedes Element nach der Merge_level() und Store()-Operation noch $6/B$ Einheiten pro Schicht übrig.

Beweis: Amortisierte Analyse (2)

- **Invariante:** Zu jedem nicht-leeren Slot j der Schicht L_i gehört ein Deposit $D_{i,j}$ von $6x/B$, wobei x die Anzahl der freien Felder in j entspricht.

- Das heisst: Um die Invariante zu erfüllen, muss jedes durch den Store() Aufruf nach Schicht L_{i+1} bewegte Element $6/B$ Einheiten an $D_{i+1,j}$ abgeben

- Insgesamt: kostet also eine Merge_Level() und eine Store() Operation pro Overflow (Schicht) $18/B$ Einheiten per Element.

ENDE

Beweis: Amortisierte Analyse (3)

- Beim Entfernen werden $7/B=(1+6)/B$ Einh. im Heap belassen

- Eine Load()-Operation wird durch das Nehmen von $B(1/B)=1$ Einheiten aus dem Heap bezahlt.
- Diese Einheiten kamen jeweils durch die letzten (aus Slot j) B entfernten Elemente zustande.
- Die restlichen $B(6/B)=6$ Einheiten dieser entfernten Elemente werden dem $D_{i,j}$ zugeordnet, auf dem Load() operiert hat (denn danach sind es dort B Einheiten weniger, es werden also $6*B/B=6$ Einheiten mehr in $D_{i,j}$ benötigt).

- Insgesamt: sind das $7/B$ Einheiten für die Load() Operation

Beweis: Amortisierte Analyse (4)

- Es bleibt: die Bezahlung für Compact(i): $3l_i/B$

- Dies wird durch die Deposits $D_{i,j}$ an den slots $j1$ und $j2$, die kompaktiert werden, bezahlt:
- Die Gesamtanzahl der leeren Plätze in den slots $j1$ und $j2$ ist mindestens l_i.
- Dafür gibt es in den Deposits $D_{i,j1}$ und $D_{i,j2}$ zusammen mindestens $6l_i/B$ Einheiten.
- Nach dem Mischen gibt es in $D_{i,j1}$ höchstens $l_i/2$ freie Slots, d.h. für das neue $D_{i,j1}$ werden nur $3l_i/B$ Einheiten benötigt
- Die anderen $3l_i/B$ Einheiten werden für Compact(i) ausgegeben.

Speicherplatzbedarf

Lemma 7: Jede Schicht enthält höchstens einen Slot, der nicht-leer und aus weniger als $l_i/2$ Elementen besitzt.

Speicherplatzbedarf

Theorem 2: Die Gesamtanzahl der benützten externen Blöcke ist höchstens $2(X/B)+L$, wobei X die Anzahl der Elemente in unserer Datenstruktur ist.

Der Gesamtspeicherplatz im MM beträgt $cM(3+L)$.

Beweis:

- In jedem Slot jeder Schicht existiert höchstens ein nur teilweise gefüllter Speicherplatz, nämlich der oberste.
- Pro Schicht existiert höchstens ein nicht-leerer Slot mit weniger als $l_i/2$ Elementen. Von diesen gibt es zusammen höchstens L.
- Für die anderen Slots gilt: für jeden halb gefüllten gibt es mindestens auch einen ganz gefüllten Block: $\leq 2X/B$

Experimentelles Setup

8 verschiedene PQ Implementierungen:

- **extern:** Externe Array-Heaps, externe Radix Heaps (Achtung: nur für monotone DEL-Folgen und integers einsetzbar), Buffer Trees, B-trees
- **intern:** Fibonacci Heaps, k-ary Heaps, Pairing Heaps, interne Radix Heaps

SPARC ULTRA 1/143:

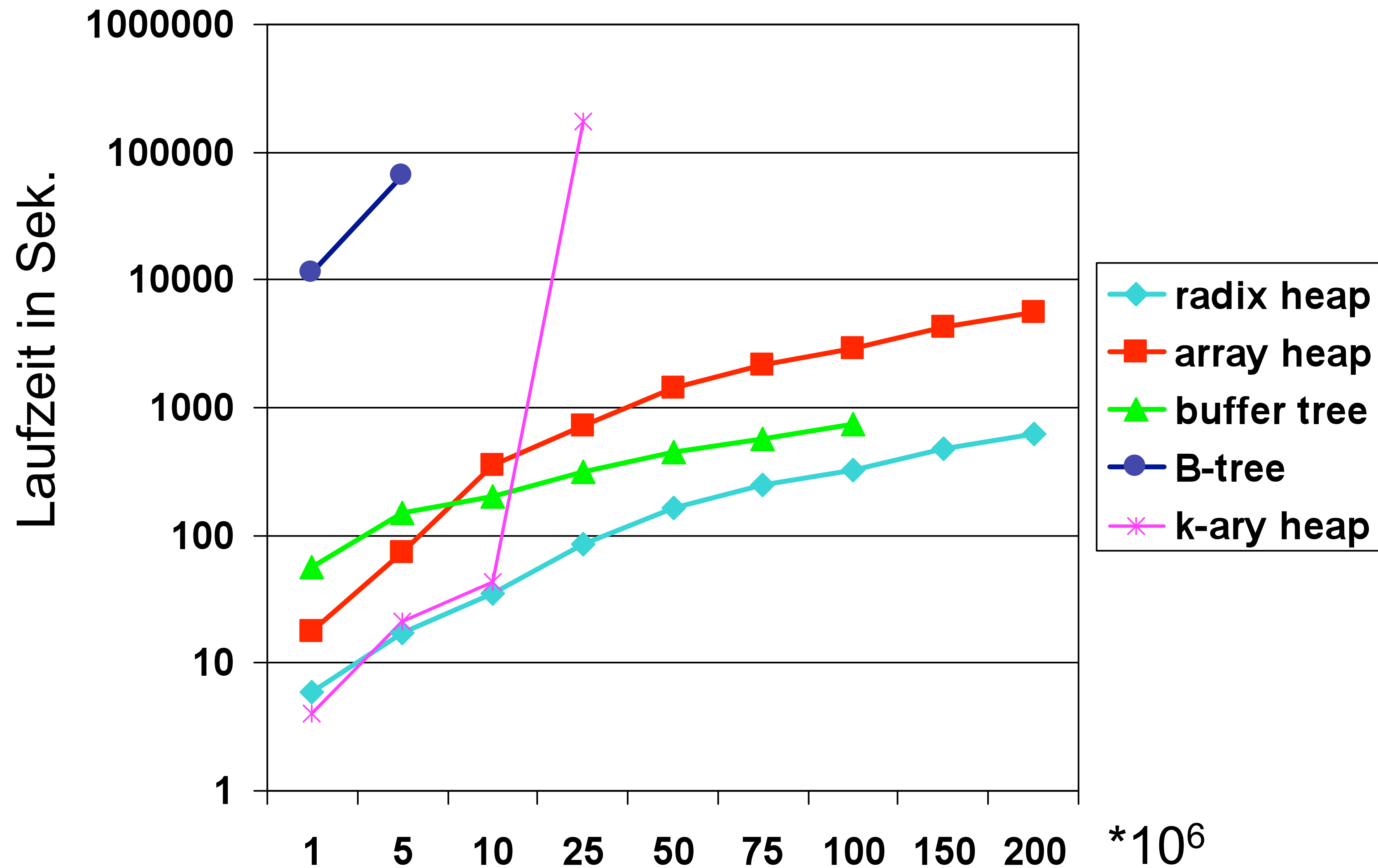
- MM: 256 Mbytes (nur $M=16$ Mbytes genutzt)
- lokale 9 GBytes fast-wide SCSI disk
- $B=32$ kbytes

Experimentelles Setup

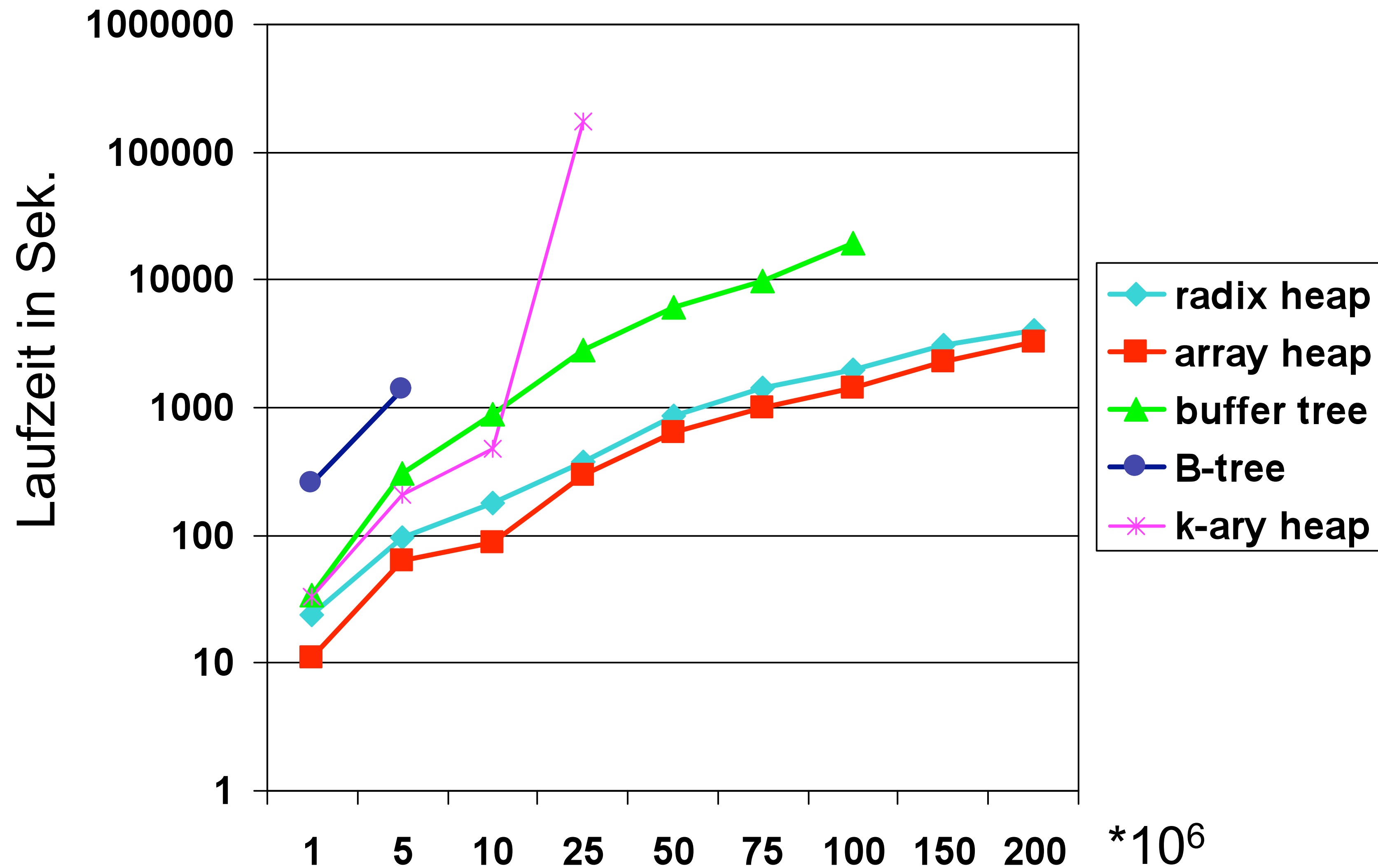
Experimentreihen:

1. **Insert-All-Delete-All:** zunächst N Insert, dann N Del_Min
3. **Intermixed:** zunächst N=20 Mio Insert, dann gemischte Insert mit prob=1/3 und Del_Min Operationen mit prob=2/3
4. **Dijkstra´s shortest-path**: simuliere Dijkstra in MM für große Graphen und teste die hierbei produzierte Sequenz von Insert und Del_Mins.

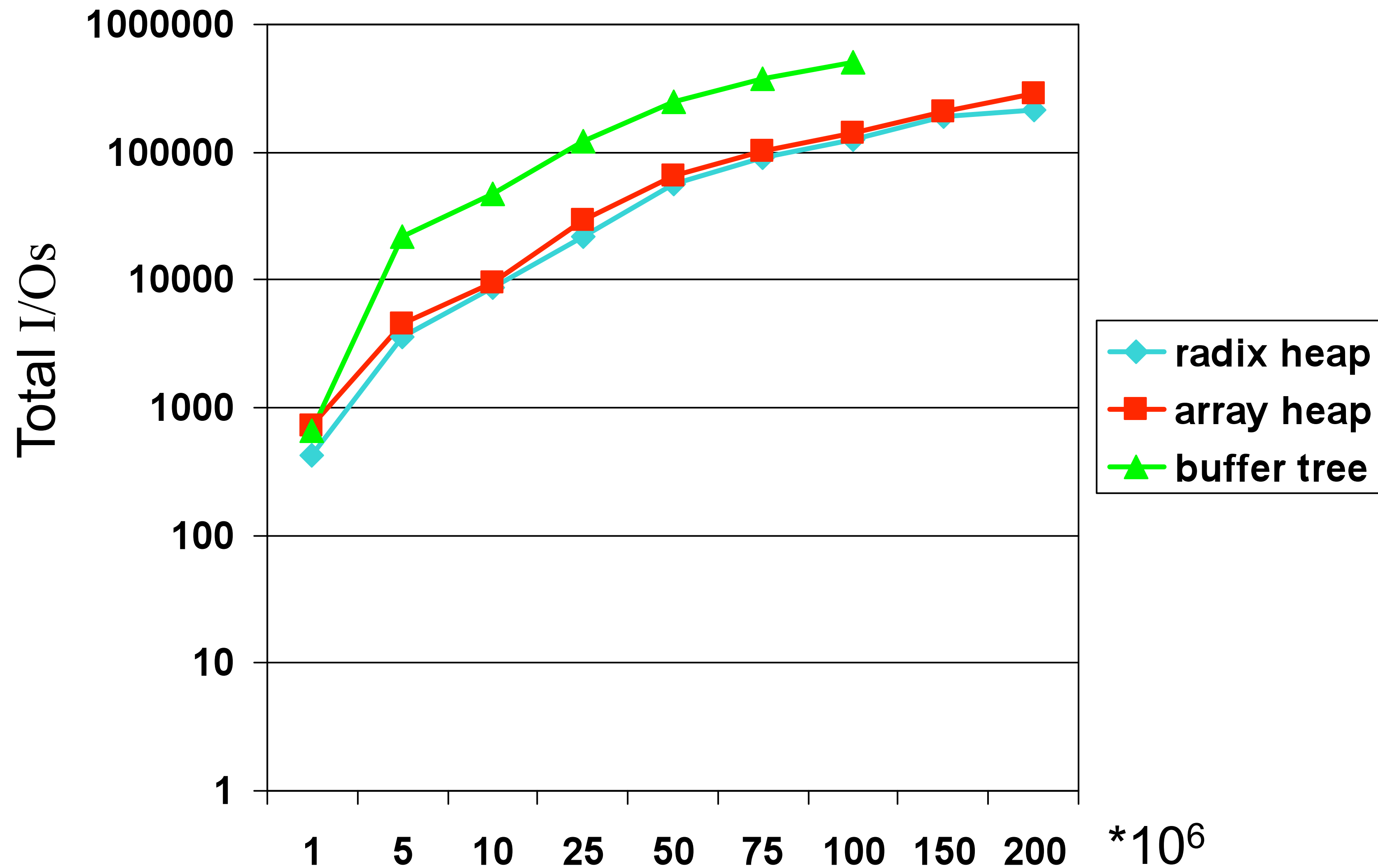
Laufzeit Insert-All-Delete-All: Insert



Laufzeit Insert-All-Delete-All: Del_Min

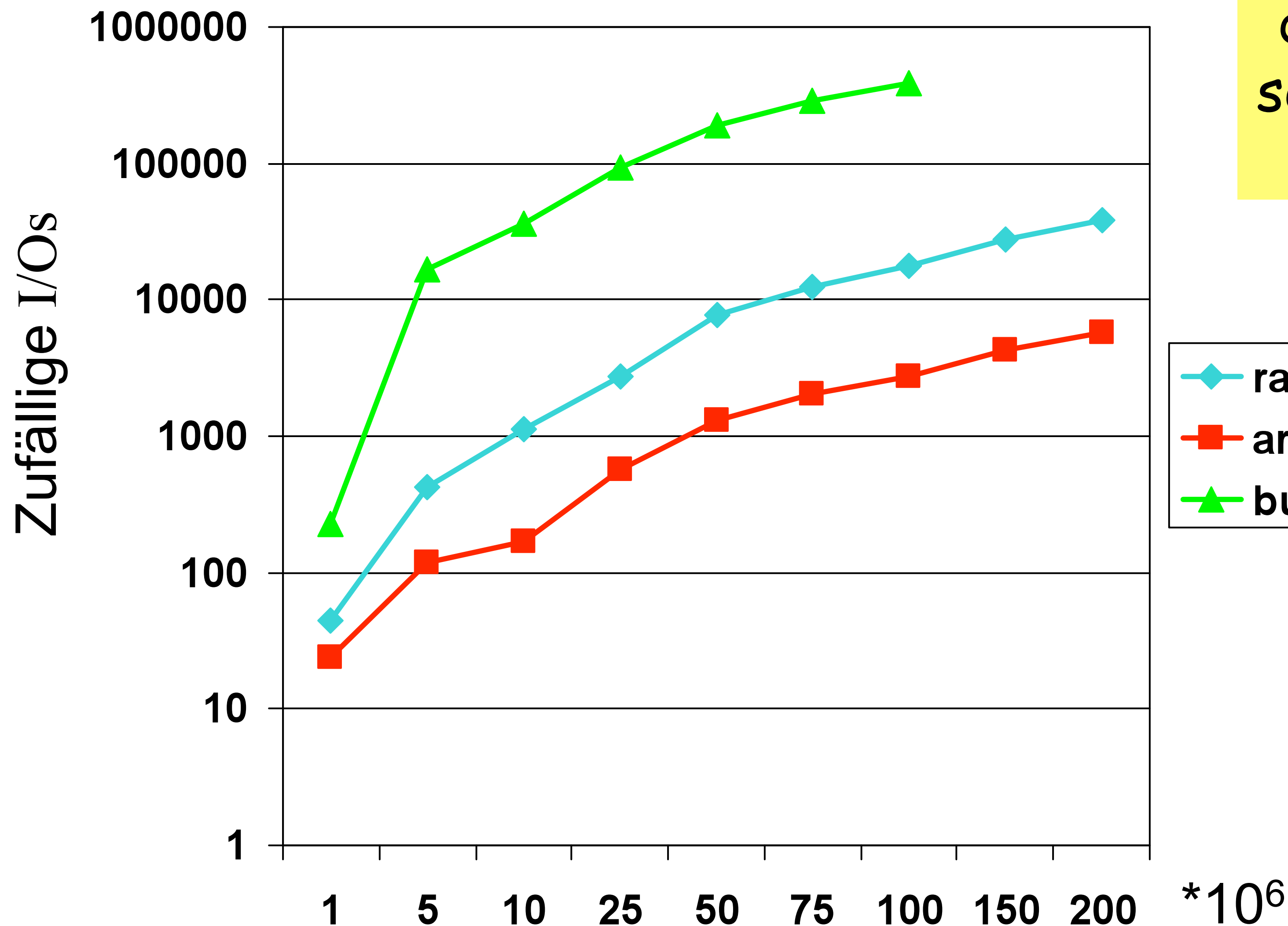


I/Os Insert-All-Delete-All: Total I/Os

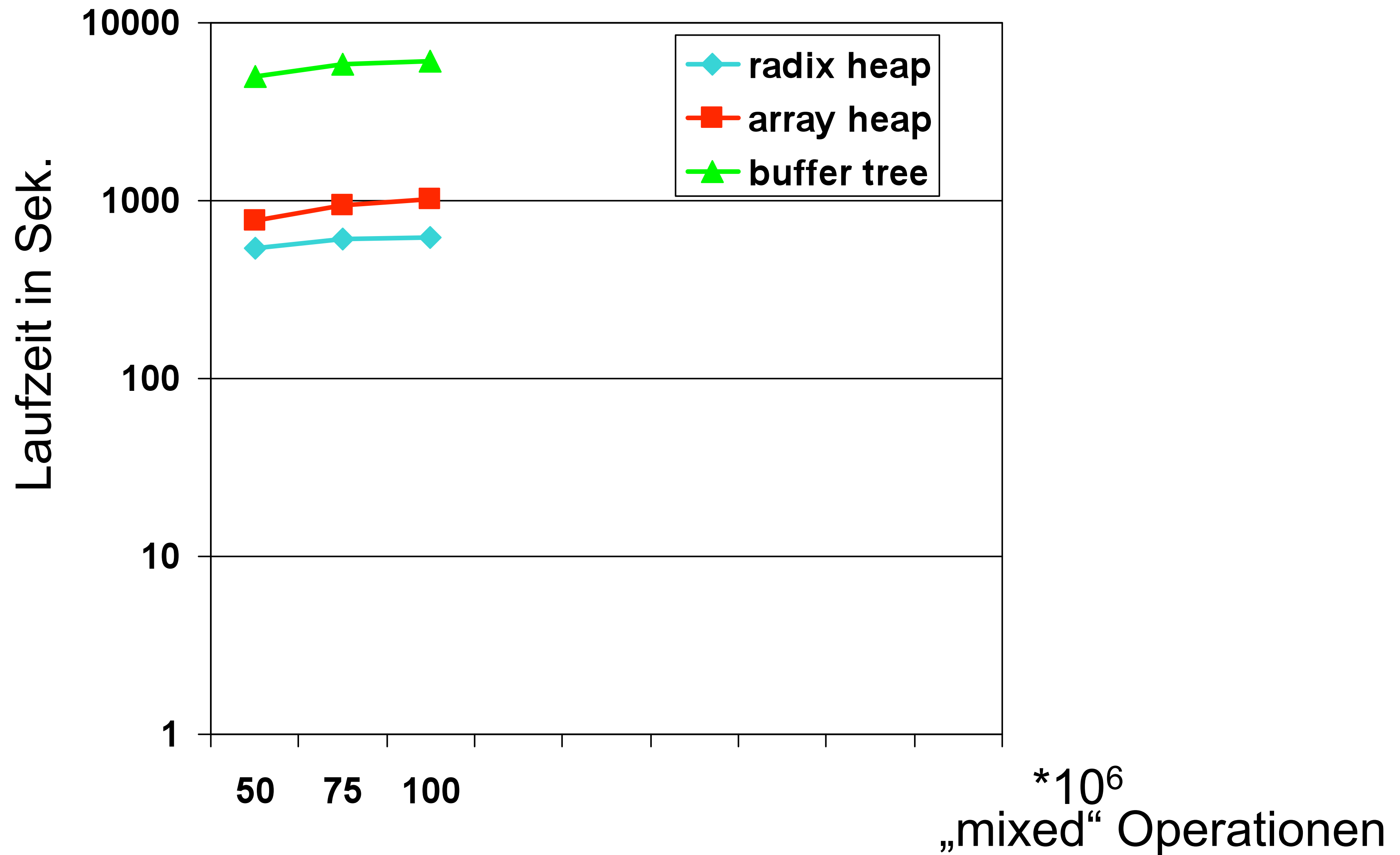


I/Os Insert-All-Delete-All: Random I/Os

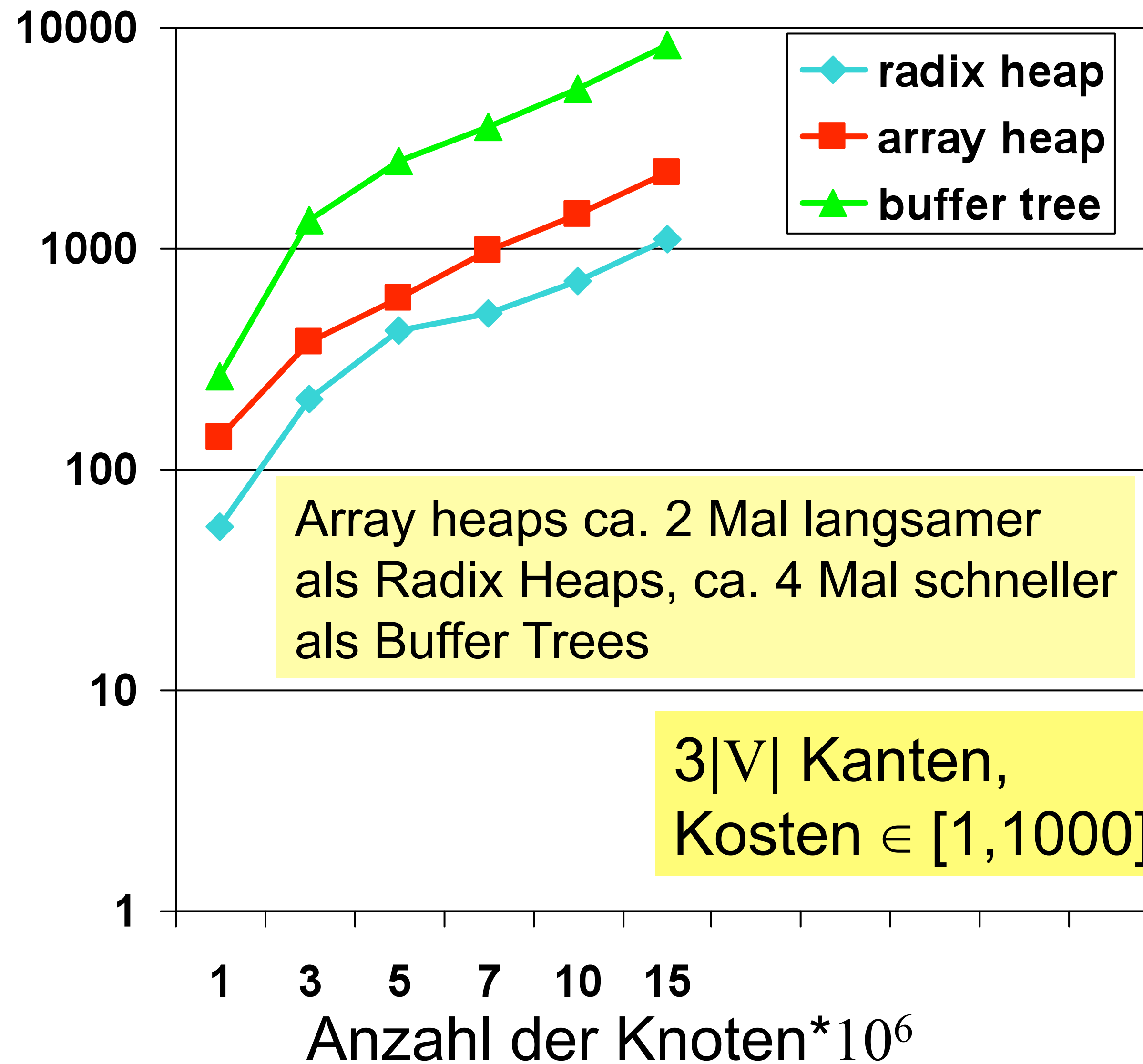
d.h. nicht-sequentielle I/Os



Laufzeit Intermixed



Laufzeit Dijkstra



ENDE