5 Sortieren & Co

Formaler

Gegeben: Elementfolge $s = \langle e_1, \ldots, e_n \rangle$

Gesucht: $s' = \langle e'_1, \ldots, e'_n \rangle$ mit

- s' ist Permutation von s.
- $e'_1 \leq \cdots \leq e'_n$ für eine lineare Ordnung '$\leq$'

Anwendungsbeispiele

- Allgemein: Vorverarbeitung
- Zum Beispiel für binäre Suche, Aufbau von Suchbäumen
- Gruppieren (Alternative Hashing?)
- Kruskals MST-Algorithmus
- Verarbeitung von Intervallgraphen (z.B. Hotelbuchungen)
- Rucksackproblem
- Scheduling, die schwersten Probleme zuerst
- Sekundärspeicheralgorithmen, z.B. Datenbank-Join

Viele verwandte Probleme. Zum Beispiel Transposition dünner Matrizen, invertierten Index aufbauen. Konversion zwischen Graphrepräsentationen.

Ergebnisüberprüfung (Checking)

Permutationseigenschaft (Sortiertheit: trivial.)

$\langle e_1, \ldots, e_n \rangle$ ist Permutation von $\langle e'_1, \ldots, e'_n \rangle$ gdw.

$$q(z) := \prod_{i=1}^{n} (z - \text{field}(\text{key}(e_i))) - \prod_{i=1}^{n} (z - \text{field}(\text{key}(e'_i))) = 0,$$

$\mathbb{F}$ sei Körper, field : Key $\longrightarrow \mathbb{F}$ sei injektiv.

Beobachtung: q hat höchstens n Nullstellen.

Auswertung an zufälliger Stelle $x \in \mathbb{F}$.

$$\mathbb{P}[q \neq 0 \wedge q(x) = 0] \leq \frac{n}{|\mathbb{F}|}$$

Monte Carlo-Algorithmus, Linearzeit.

Frage: Welchen Körper $\mathbb{F}$ nehmen wir?

5.1 Sentinels am Beispiel Sort. durch Einfügen

Procedure insertionSort(a : **Array** $[1..n]$ **of** Element)

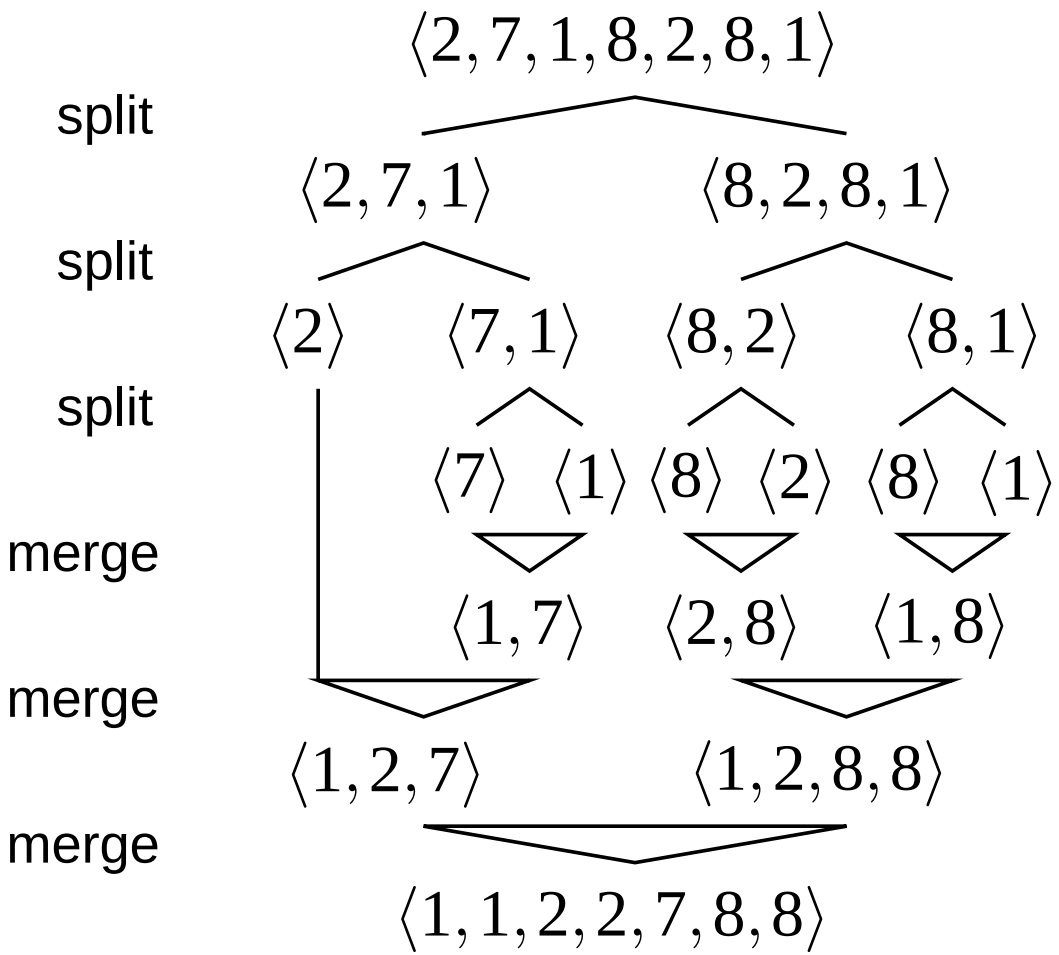
```
for i := 2 to n do
    invariant a[1] ≤ ··· ≤ a[i−1]
    // move a[i] to the right place
    e := a[i]
    if e < a[1] then                                  // new minimum
        for j := i downto 2 do  a[j] := a[j−1]
        a[1] := e
    else                                              // use a[1] as a sentinel
        for j := i downto −∞ while a[j−1] > e do  a[j] := a[j−1]
        a[j] := e
```

5.2 Mischen

Jeweils $\min(a,b)$ in die Ausgabe schieben. Zeit $O(n)$

a	b	c	operation
$\langle 1,2,7\rangle$	$\langle 1,2,8,8\rangle$	$\langle\rangle$	move a
$\langle 2,7\rangle$	$\langle 1,2,8,8\rangle$	$\langle 1\rangle$	move b
$\langle 2,7\rangle$	$\langle 2,8,8\rangle$	$\langle 1,1\rangle$	move a
$\langle 7\rangle$	$\langle 2,8,8\rangle$	$\langle 1,1,2\rangle$	move b
$\langle 7\rangle$	$\langle 8,8\rangle$	$\langle 1,1,2,2\rangle$	move a
$\langle\rangle$	$\langle 8,8\rangle$	$\langle 1,1,2,2,7\rangle$	move a
$\langle\rangle$	$\langle\rangle$	$\langle 1,1,2,2,7,8,8\rangle$	concat b

Sortieren durch Mischen (Mergesort)



Analyse: $T(n) = \mathrm{O}(n) + T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) = \mathrm{O}(n \log n)$.

5.3 Eine untere Schranke

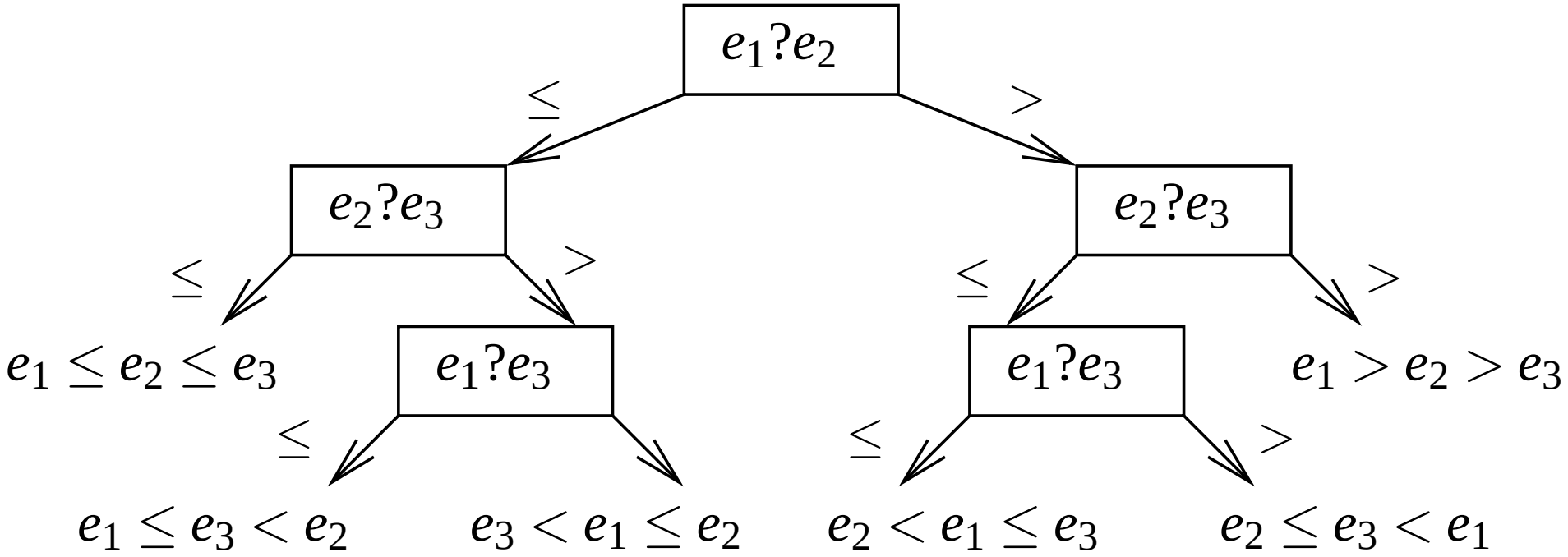
Vergleichsbasiertes Sortieren: Informationen über Elemente nur durch Zwei-Wege-Vergleich $e_i \leq e_j$?.

Satz: Deterministische vergleichsbasierte Sortieralgorithmen brauchen

$$n \log n - \mathrm{O}(n)$$

Vergleiche im schlechtesten Fall.

Baumbasierte Sortierer-Darstellung



Mindestens ein Blatt pro Permutation von $e_1, \ldots, e_n$

Ausführungszeit entspricht Tiefe

Beweis

Baum der Tiefe T hat höchstens 2^T Blätter.

$\Rightarrow 2^T \geq n!$

$\Leftrightarrow T \geq \log n! \geq \log\left(\frac{n}{e}\right)^n = n\log n - n\log e$

Randomisierung, Mittlere Ausführungszeit

Satz: immer noch $n \log n - \mathrm{O}(n)$ Vergleiche.

Beweis: nicht hier.

5.4 Quicksort

Satz: bei zufälligem Pivot: erwartet $\leq 2n \ln n$ Vergleiche.

Platzverbrauch: inplace $+\mathrm{O}(\log n)$ (Trick !)

Beweis: nicht hier

Geschicktere Pivotwahl ?

[Kaligosi Sanders 2006]: Kaum der Mühe Wert.

Branch-Mispredictions !

$\rightsquigarrow$ Sortieren ohne bedingte Sprünge [Sanders Singler 2004]

5.5 Auswahl (Selection)

Wähle das r-t größte Element.

QuickSelect $\approx$ Quicksort mit einseitiger Rekursion.

erwartete Zeit $\mathrm{O}(n)$.

Deterministischer Algorithmus in Zeit $\mathrm{O}(n)$.

(Idee: QuickSelect mit det. Pivotwahl)

[Kaligosi Mehlhorn Munro Sanders 2005]:

$(1+o(1))$untere Schranke für folgende Verallgemeinerung:

Wähle k Elemente mit vorgegebenen Rängen $r_1,\ldots,r_k$

5.6 Durchbrechen der unteren Schranke – Ganzzahliges Sortieren

Idee: Modell ändern.

Mehr mit den Schlüsseln machen als nur Vergleichen.

K Schlüssel – Eimer-Sortieren (bucket sort)

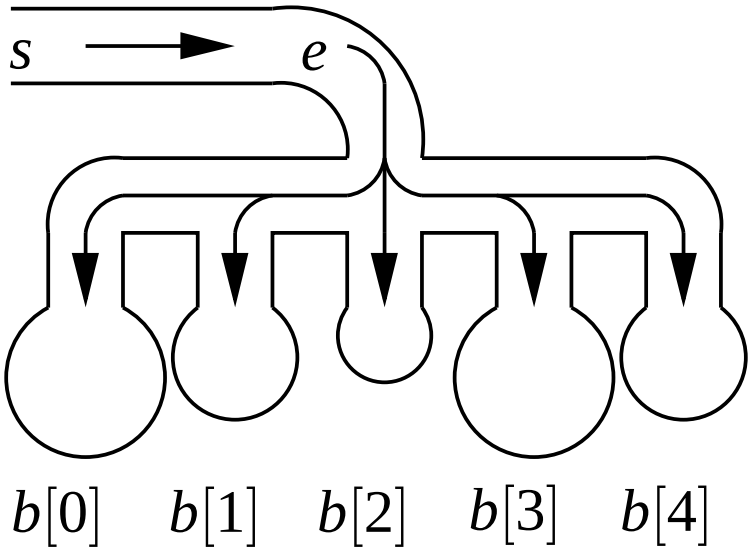
Procedure KSort(s : Sequence **of** Element)

b$=\langle\langle\rangle,\ldots,\langle\rangle\rangle$: **Array** $[0..K-1]$ **of** Sequence **of** Element

foreach $e \in s$ **do** $b[\text{key}(e)]$.pushBack(e)

s := concatenation of $b[0],\ldots,b[K-1]$

Zeit: $O(n+K)$



K^d Schlüssel – (LSD-)Radix-Sortieren

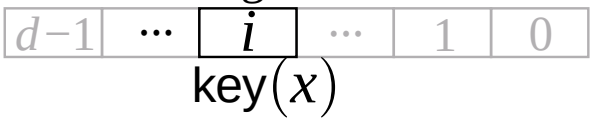
Beobachtung: KSort ist stabil, d.h.,

Elemente mit gleichem Schlüssel behalten ihre relative Reihenfolge.

Procedure LSDRadixSort(s : Sequence **of** Element)

for $i := 0$ **to** $d-1$ **do**

redefine key(x) as $(x$ **div** $K^i)$ **mod** K// x



KSort(s)

invariant s is sorted with respect to digits $i..0$

Zeit: $O(d(n+K))$

Algorithm Engineering

- Einfache schnelle 2-pass Array-Implementierung
- Nicht (ohne weiteres) inplace
- MSD-Radix-Sort: Wichtigste Ziffer zuerst.

  im Mittel Cache-effizienter aber Probleme mit schlechtestem Fall
- Kleineres K kann besser sein. (Cache-Misses, LSD-Misses)

Mehr Theorie:

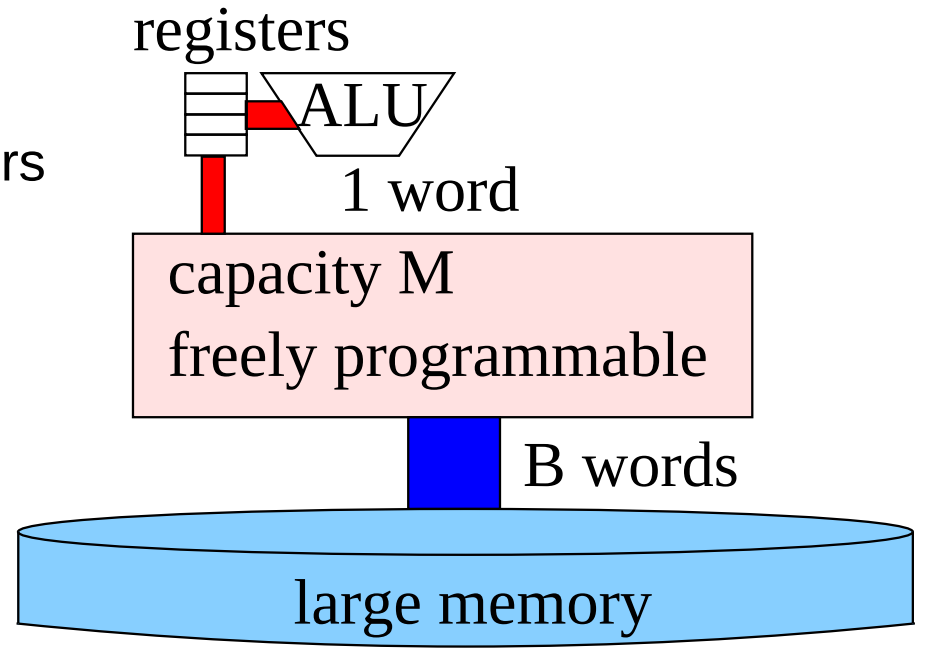
Zeit $O\left(n\sqrt{\log\log n}\right)$ (erwartet) für ganzzahlige Schlüssel, die in ein Maschinenwort passen. [Han Thorup 2002]

5.7 Externes Sortieren

n: Eingabegröße

M: Größe des schnellen Speichers

B: Blockgröße



Procedure externalMerge$(a, b, c$:File **of** Element$)$

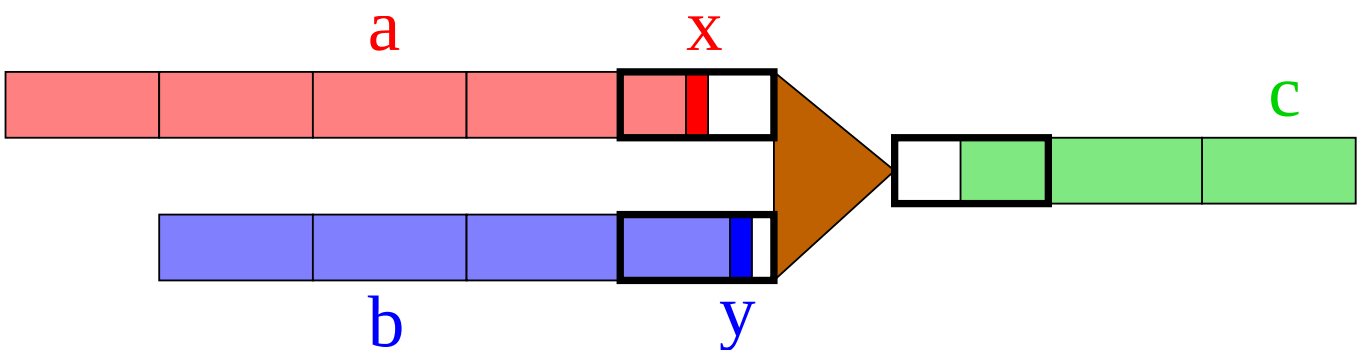
x := a.readElement // Assume emptyFile.readElement$= \infty$

y := b.readElement

for $j := 1$ **to** $|a| + |b|$ **do**

if $x \leq y$ **then** c.writeElement(x); x := a.readElement

else c.writeElement(y); y := b.readElement



Externes (binäres) Mischen – I/O-Analyse

Datei a lesen: $\lceil |a|/B \rceil \leq |a|/B + 1.$

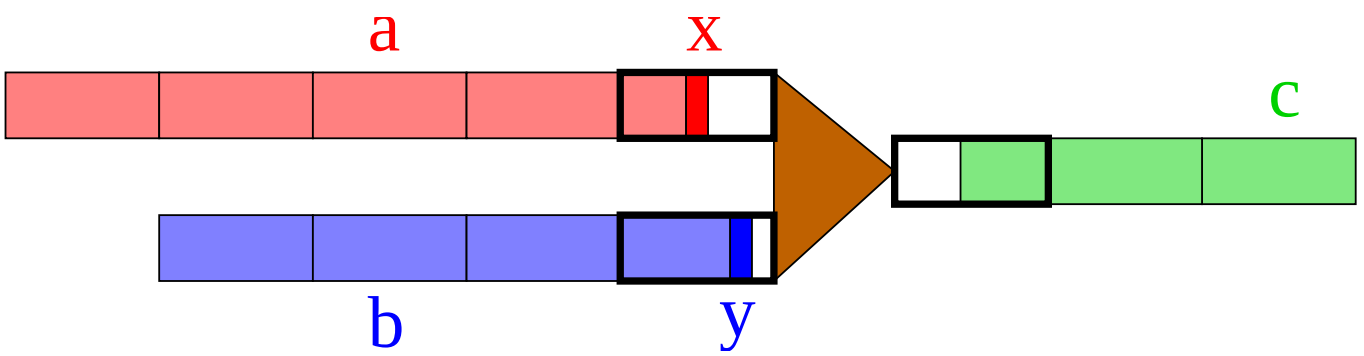
Datei b lesen: $\lceil |b|/B \rceil \leq |b|/B + 1.$

Datei c schreiben: $\lceil (|a| + |b|)/B \rceil \leq (|a| + |b|)/B + 1.$

Insgesamt:

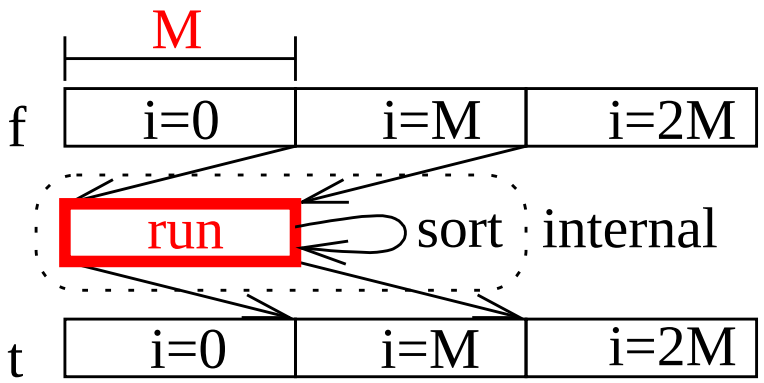
$$\leq 3 + 2\frac{|a| + |b|}{B} \approx 2\frac{|a| + |b|}{B}$$

Bedingung: Wir brauchen 3 Pufferblöcke, d.h., $M > 3B$.



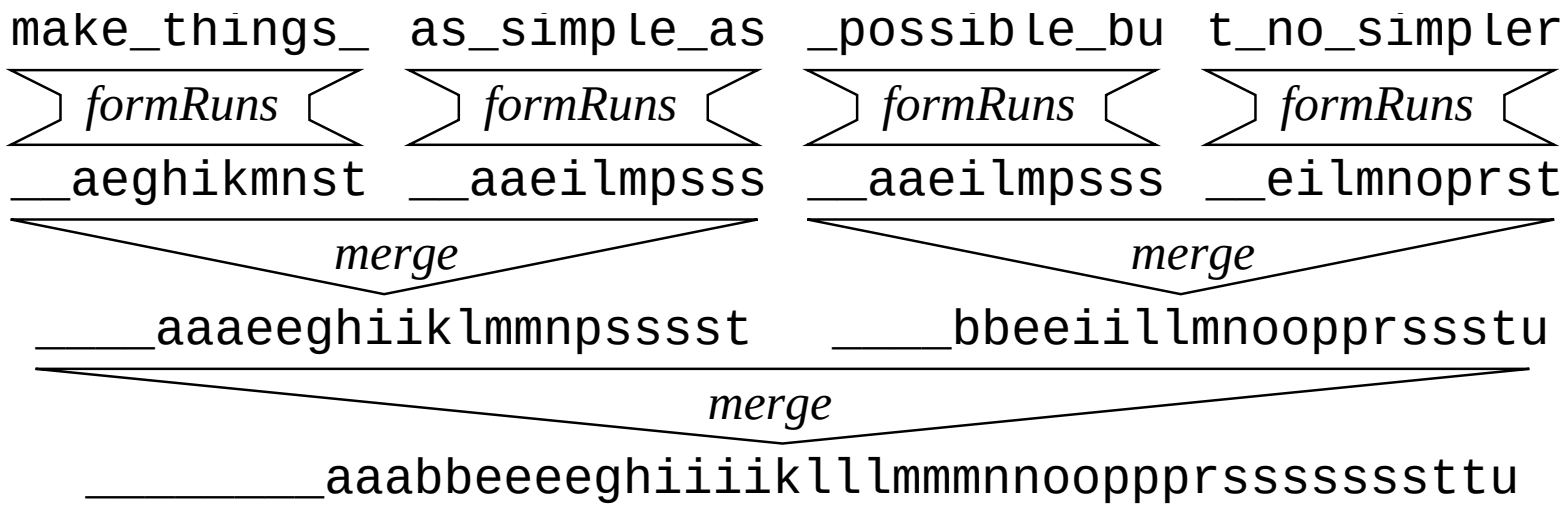
Run Formation

Sortiere Eingabeportionen der Größe M



I/Os: $\approx 2\frac{n}{B}$

Sortieren durch Externes Binäres Mischen



Procedure externalBinaryMergeSort // I/Os: $\approx$

run formation // $2n/B$

while more than one run left **do** // $\lceil \log \frac{n}{M} \rceil \times$

merge pairs of runs // 2n/B

output remaining run // $\sum: 2\frac{n}{B}\left(1+\left\lceil \log \frac{n}{M} \right\rceil\right)$

Zahlenbeispiel: PC 2007

$n = 2^{38}$ Byte

$M = 2^{31}$ Byte

$B = 2^{20}$ Byte

I/O braucht 2^{-6} s

Zeit: $2\frac{n}{B}\left(1+\left\lceil \log \frac{n}{M} \right\rceil\right) = 2 \cdot 2^{18} \cdot (1+7) \cdot 2^{-6}$ s $= 2^{16}$ s≈ 18 h

Idee: 8 Durchläufe $\rightsquigarrow$ 2 Durchläufe

Mehrwegemischen

Procedure multiwayMerge$(a_1, \ldots, a_k, c$:File **of** Element$)$

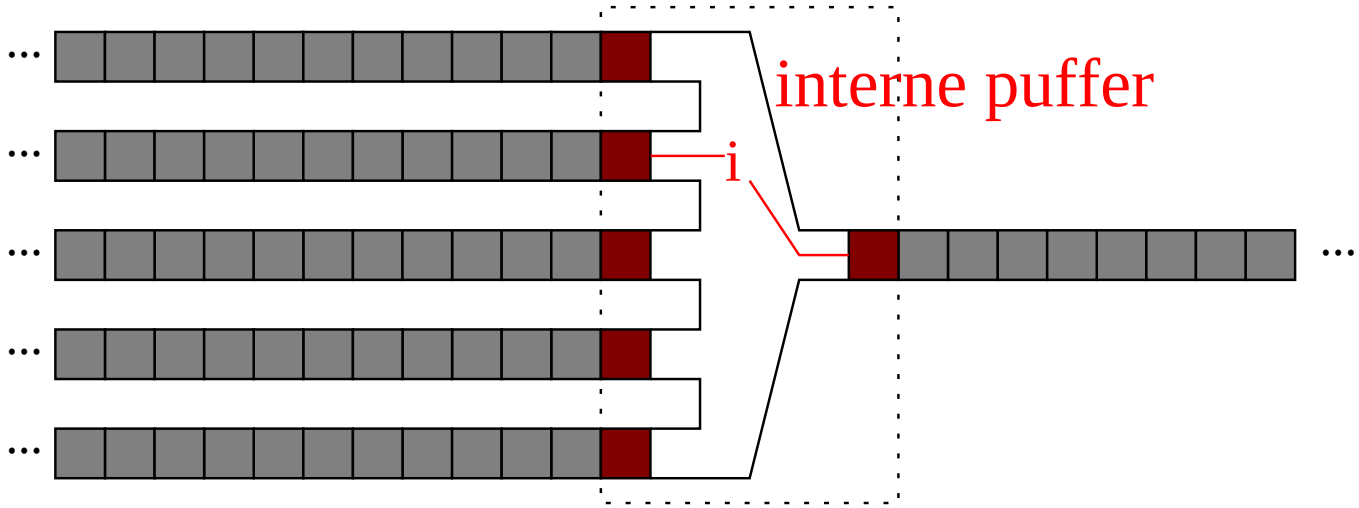
for $i := 1$ **to** k **do** $x_i := a_i$.readElement

for $j := 1$ **to** $\sum_{i=1}^{k} |a_i|$ **do**

find $i \in 1..k$ that minimizes x_i // no I/Os!, $O(\log k)$ time

c.writeElement(x_i)

$x_i := a_i$.readElement



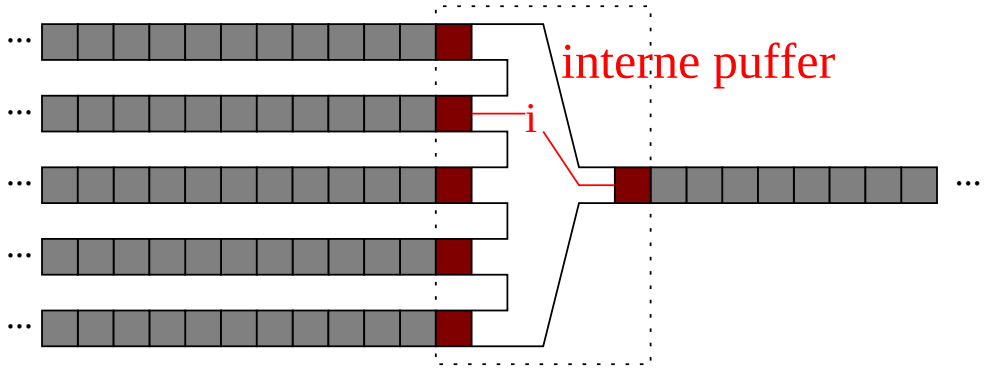
Mehrwegemischen – Analyse

I/Os: Datei a_i lesen: $\approx |a_i|/B$.

Datei c schreiben: $\approx \sum_{i=1}^{k} |a_i|/B$

Insgesamt:

$$\leq\approx 2\frac{\sum_{i=1}^{k} |a_i|}{B}$$



Bedingung: Wir brauchen k Pufferblöcke, d.h., $k < M/B$.

Interne Arbeit: (benutze Prioritätsliste !)

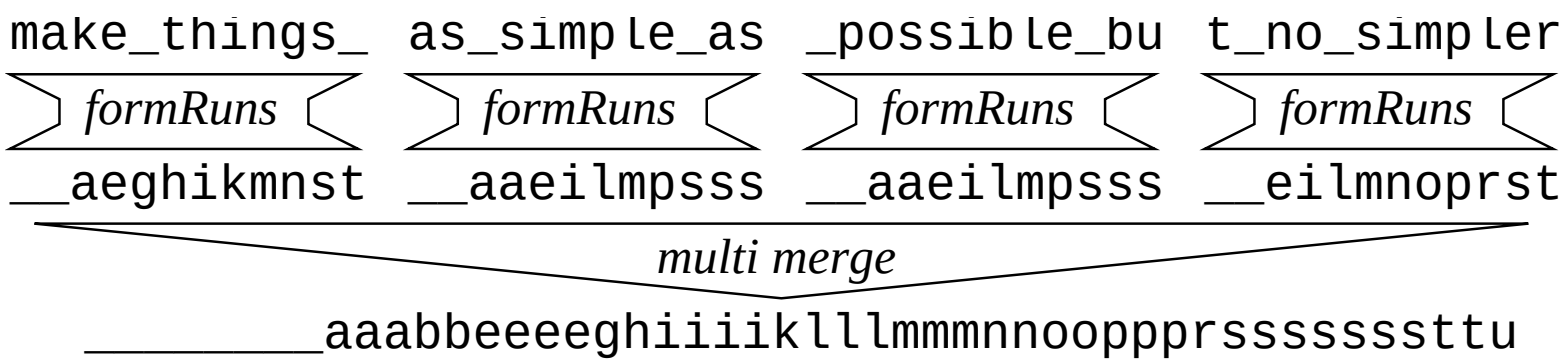
$$\mathrm{O}\left(\log k \sum_{i=1}^{k} |a_i|\right)$$

Sortieren durch Mehrwege-Mischen

- Sortiere $\lceil n/M \rceil$ runs mit je M Elementen — $2n/B$ I/Os
- Mische jeweils M/B runs — $2n/B$ I/Os
- bis nur noch ein run übrig ist — $\times \left\lceil \log_{M/B} \frac{n}{M} \right\rceil$ Mischphasen

Insgesamt

$$\text{sort}(n) := \frac{2n}{B}\left(1 + \left\lceil \log_{M/B} \frac{n}{M} \right\rceil\right) \text{ I/Os}$$



Sortieren durch Mehrwege-Mischen

Interne Arbeit:

$$\mathrm{O}\left(\overbrace{n\log M}^{\text{run formation}} + \underbrace{n\log\frac{M}{B}}_{\text{PQ access per phase}} \overbrace{\left\lceil\log_{M/B}\frac{n}{M}\right\rceil}^{\text{phases}}\right) = \mathrm{O}(n\log n)$$

Mehr als eine Mischphase?:

Nicht für Hierarchie Hauptspeicher, Festplatte.

Grund $\overbrace{\frac{M}{B}}^{>1000} > \overbrace{\frac{\text{RAM Euro/bit}}{\text{Platte Euro/bit}}}^{\approx 200}$

Mehr zu externem Sortieren

Untere Schranke $\approx \frac{2^{(?)}n}{B}\left(1+\left\lceil \log_{M/B}\frac{n}{M}\right\rceil\right)$ I/Os

[Aggarwal Vitter 1988]

Obere Schranke $\approx \frac{2n}{DB}\left(1+\left\lceil \log_{M/B}\frac{n}{M}\right\rceil\right)$ I/Os (erwartet)

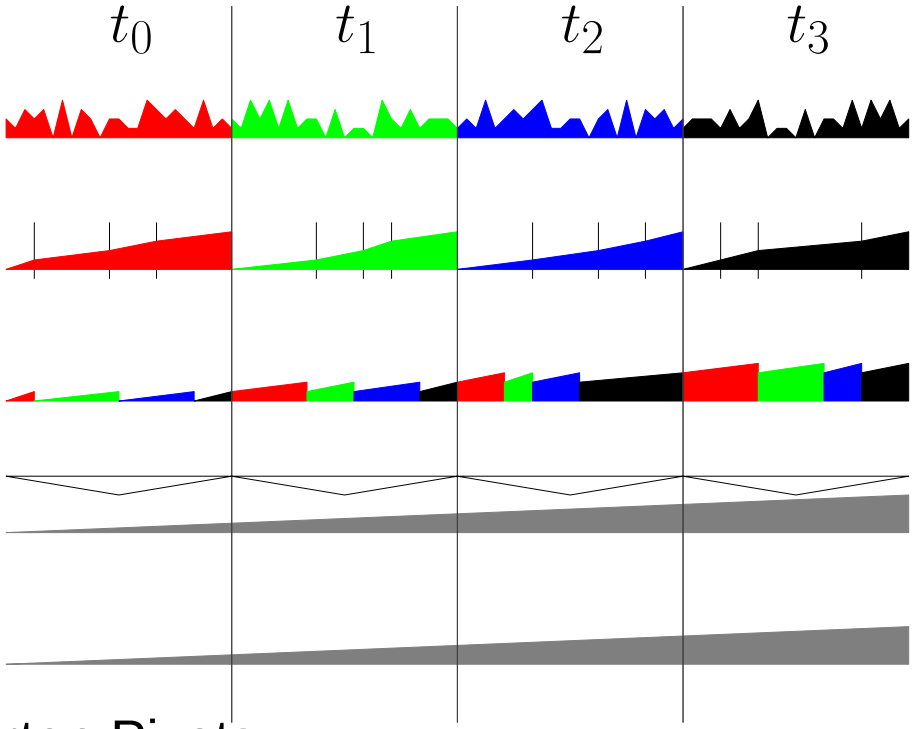
für D parallele Platten

[Hutchinson Sanders Vitter 2005, Dementiev Sanders2003]

Offene Frage: deterministisch?

Paralleles Sortieren durch Mehrwegemischen

1. p Prozessoren sortieren
   je n/p Elemente lokal
2. Finde pivots so dass
   n/p Elemente zwischen
   zwei benachbarten pivots liegen
3. Jeder Prozessor macht
   Mehrwegemischen für alle
   Elemente zwischen zwei benachbarten Pivots.



Mehr in “Parallele Algorithmen”

Messungen Spart T1 – 8 Kerne, 128 bit Elemente

[Sanders Singler 2007]