Karlsruhe Institute of Technology

Algorithmen II

Peter Sanders, Thomas Worsch, Simon Gog

Übungen:

Demian Hespe, Yaroslav Akhremtsev

Institut für Theoretische Informatik, Algorithmik II

Web:

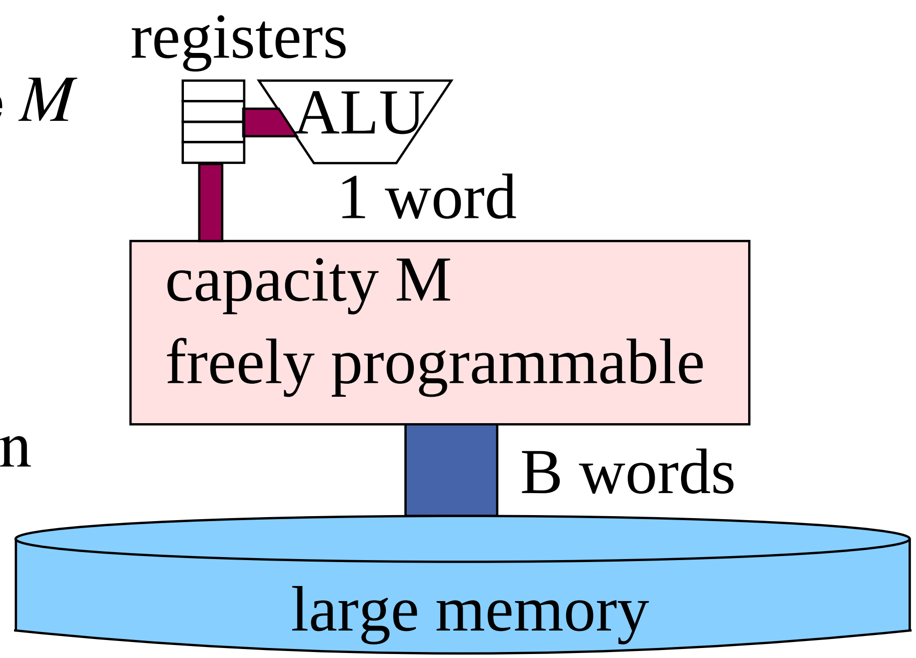
`http://algo2.iti.kit.edu/AlgorithmenII_WS17.php`

15 Externe Algorithmen

15.1 Das Sekundärspeichermodell

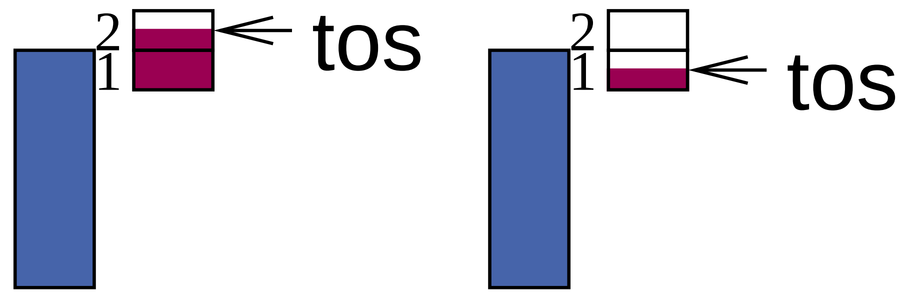
M: Schneller Speicher der Größe M

B: Blockgröße



Analyse: Blockzugriffe zählen

15.2 Externe Stapel



Datei mit Blöcken

2 interne Puffer

push: Falls Platz, in Puffer.

Sonst schreibe Puffer eins in die Datei (push auf Blockebene)

Umbenennung: Puffer 1 und 2 tauschen die Plätze

pop: Falls vorhanden, pop aus Puffer.

Sonst lese Puffer eins aus der Datei (pop auf Blockebene)

Analyse: amortisiert $O(1/B)$ I/Os pro Operation

Aufgabe 1: Beweis.

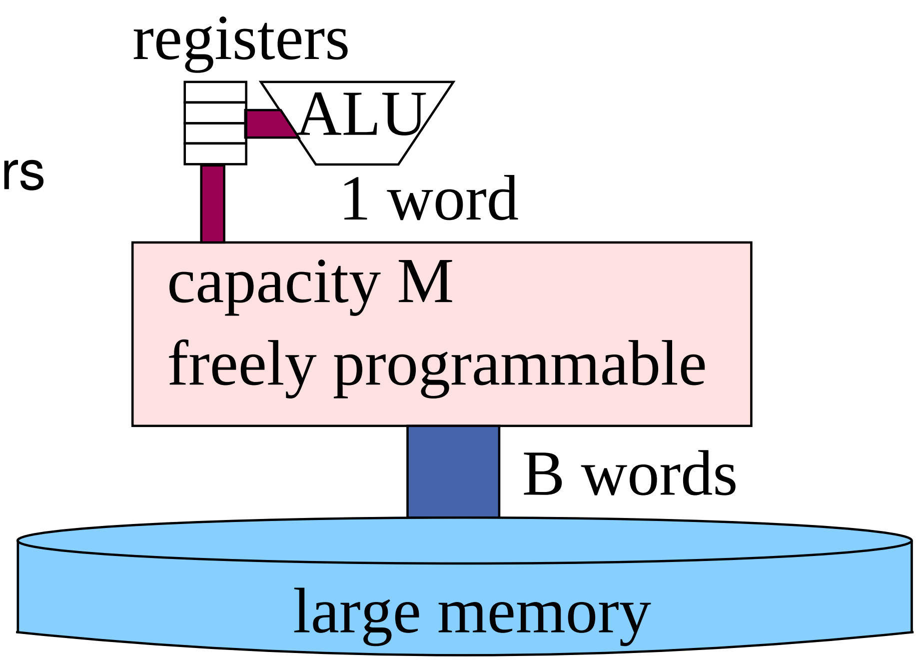
Aufgabe 2: effiziente Implementierung ohne überflüssiges Kopieren

15.3 Externes Sortieren

n: Eingabegröße

M: Größe des schnellen Speichers

B: Blockgröße



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

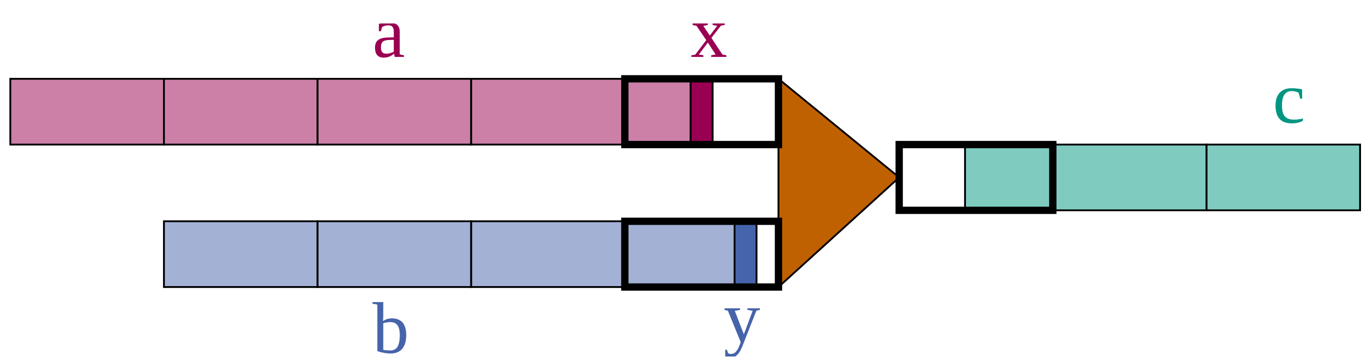
$x := a$.readElement // Assume emptyFile.readElement$= \infty$

$y := b$.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: $\lceil |a|/B \rceil \leq |a|/B + 1$.

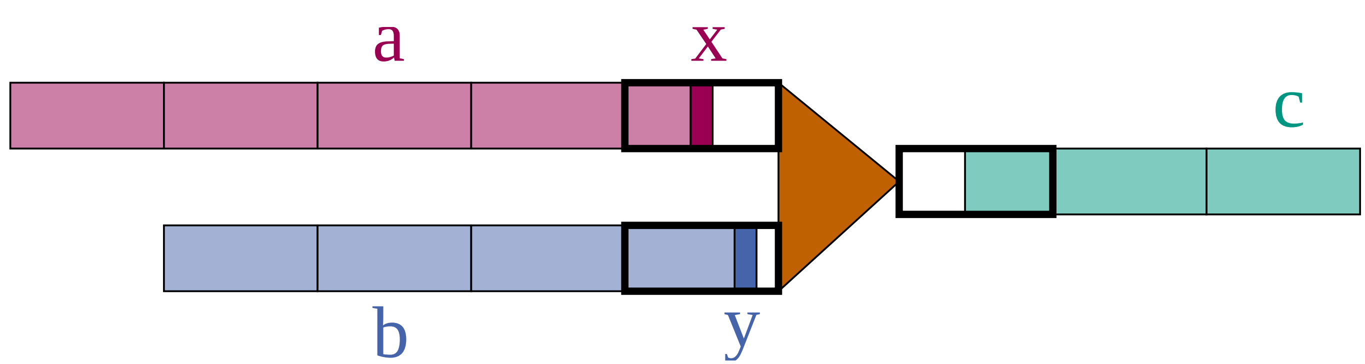
Datei b lesen: $\lceil |b|/B \rceil \leq |b|/B + 1$.

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

Insgesamt:

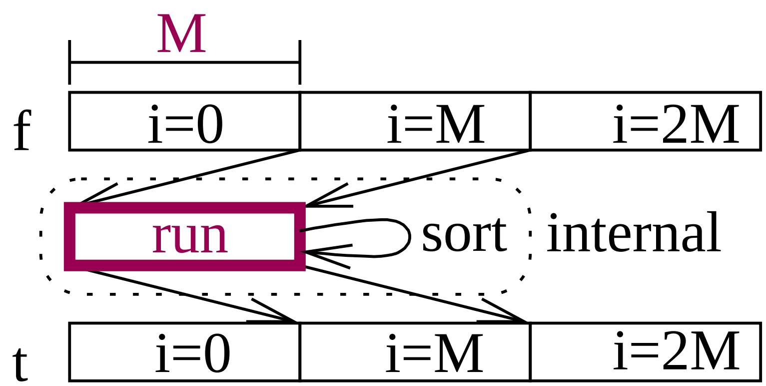
$$\leq 3 + 2\frac{|a|+|b|}{B} \approx 2\frac{|a|+|b|}{B}$$

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



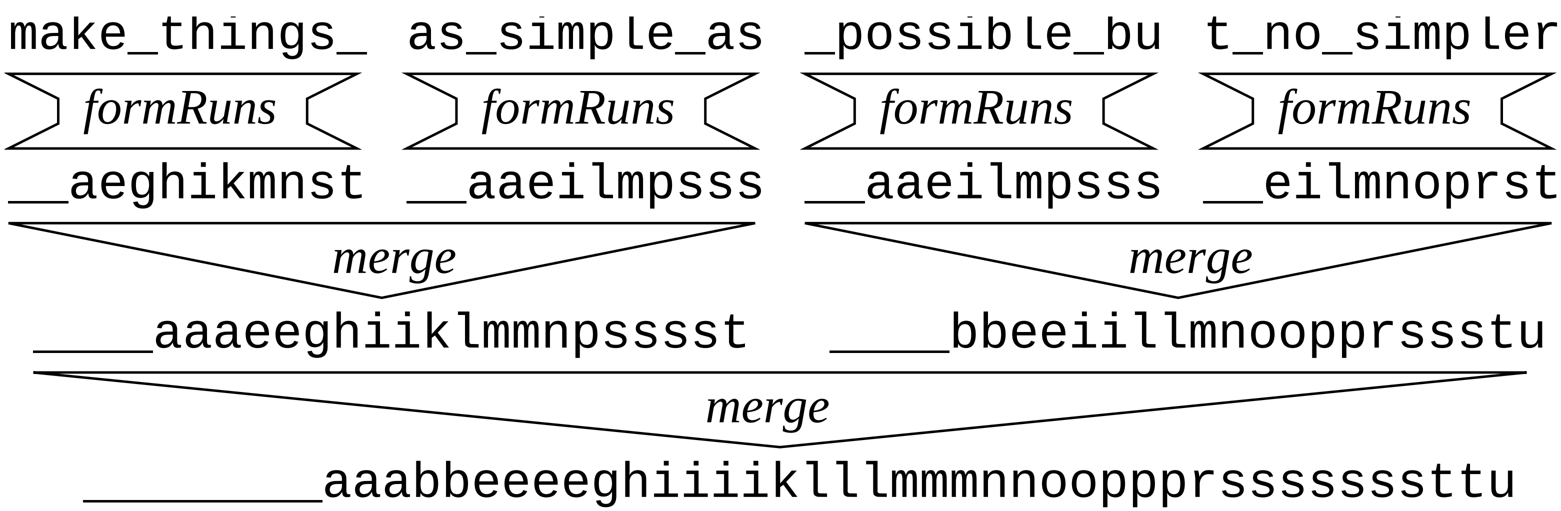
Run Formation

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** // $\left\lceil \log \frac{n}{M} \right\rceil \times$

    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 2010

$n = 2^{39}$ Byte

$M = 2^{33}$ Byte

$B = 2^{22}$ Byte

I/O braucht 2^{-4} s

Zeit: $2\frac{n}{B}\left(1+\left\lceil\log\frac{n}{M}\right\rceil\right) = 2\cdot 2^{17}\cdot(1+6)\cdot 2^{-4}$ s $= 2^{16}$ s ≈ 32 h

Idee: 7 Durchläufe $\rightsquigarrow$ 2 Durchläufe

Mehrwegemischen

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

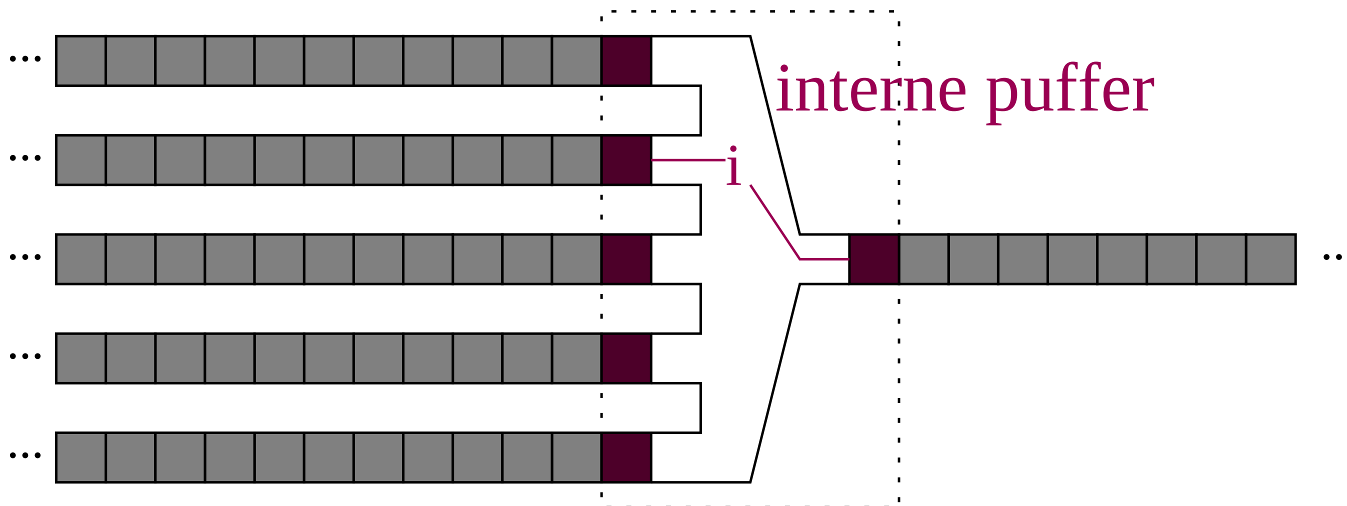
  for $i := 1$ **to** k **do** $x_i := a_i$.readElement

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

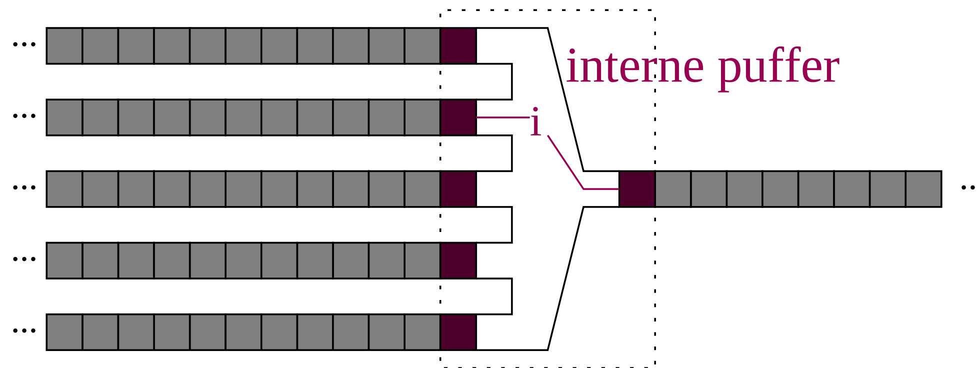
    find $i \in 1..k$ that minimizes x_i // no I/Os!, $O(\log k)$ time

    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 $k+1$ Pufferblöcke, d.h., $k+1 < M/B$

(im Folgenden vereinfacht zu $k < M/B$)

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

```
make_things_   as_simple_as   _possible_bu   t_no_simpler
  formRuns       formRuns       formRuns       formRuns
__aeghikmnst   __aaeilmpsss   __aaeilmpsss   __eilmnoprst
                        multi merge
________aaabbeeeeghiiiiklllmmmnnooppprsssssssttu
```

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.

$$\text{Grund}\quad \overbrace{\frac{M}{B}}^{>1000} > \overbrace{\frac{\text{RAM Euro}/\text{bit}}{\text{Platte Euro}/\text{bit}}}^{\approx 400}$$

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?

Externe Prioritätslisten

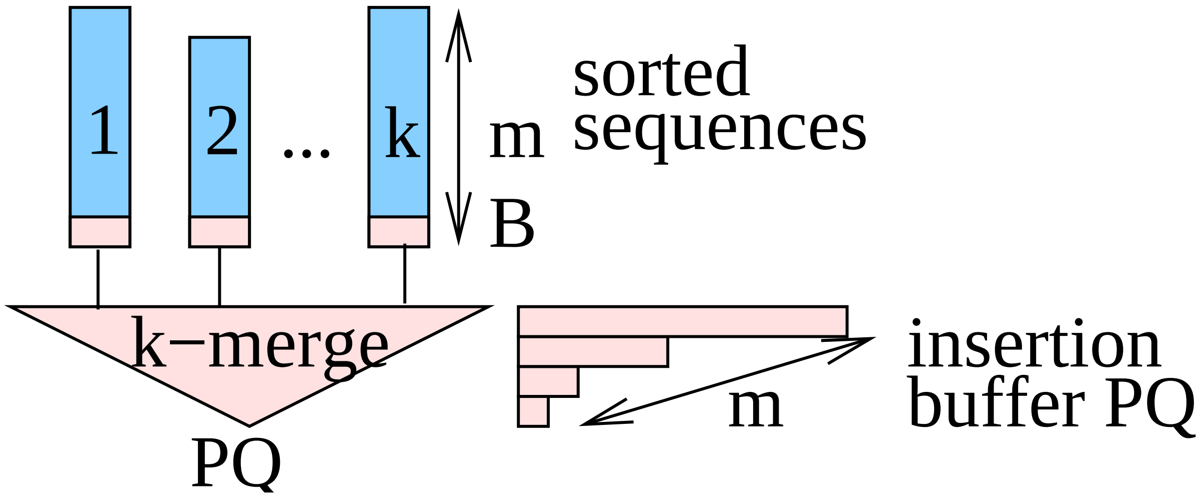
Problem: Binary heaps brauchen

$\Theta\left(\log \frac{n}{M}\right)$ I/Os pro deleteMin

Wir hätten gerne:

$\Theta\left(\frac{1}{B} \log_{M/B} \frac{n}{M}\right)$ I/Os amortisiert

Mittelgroße PQs – $km \ll M^2/B$ Einfügungen



Insert: Anfangs in insertion buffer.

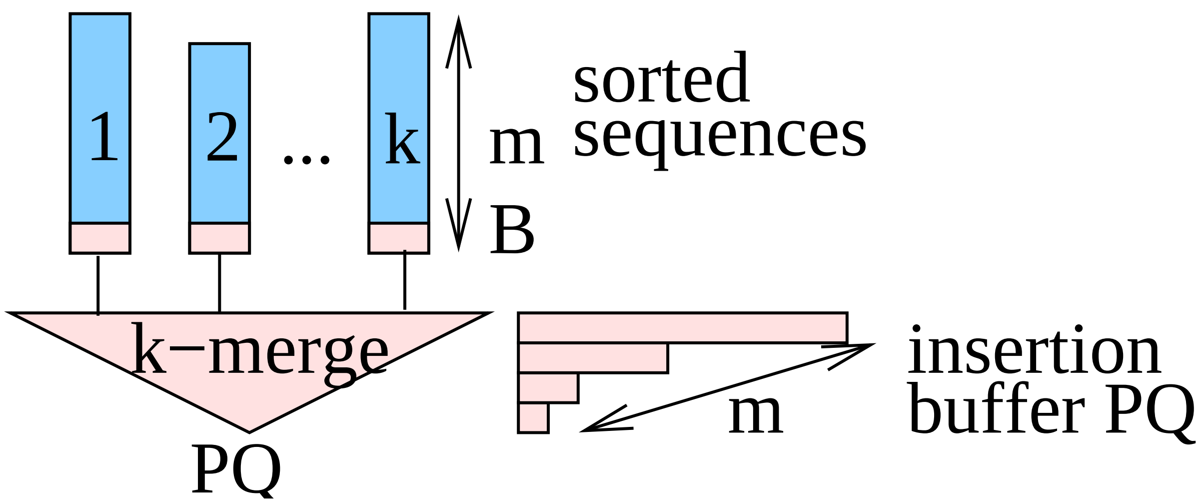
Überlauf $\longrightarrow$

sort; flush; kleinster Schlüssel in merge-PQ

Delete-Min: deleteMin aus der PQ mit kleinerem $\min$

Analyse – I/Os

deleteMin: jedes Element wird $\leq 1\times$ gelesen, zusammen mit B anderen – amortisiert $1/B$ penalty für insert.



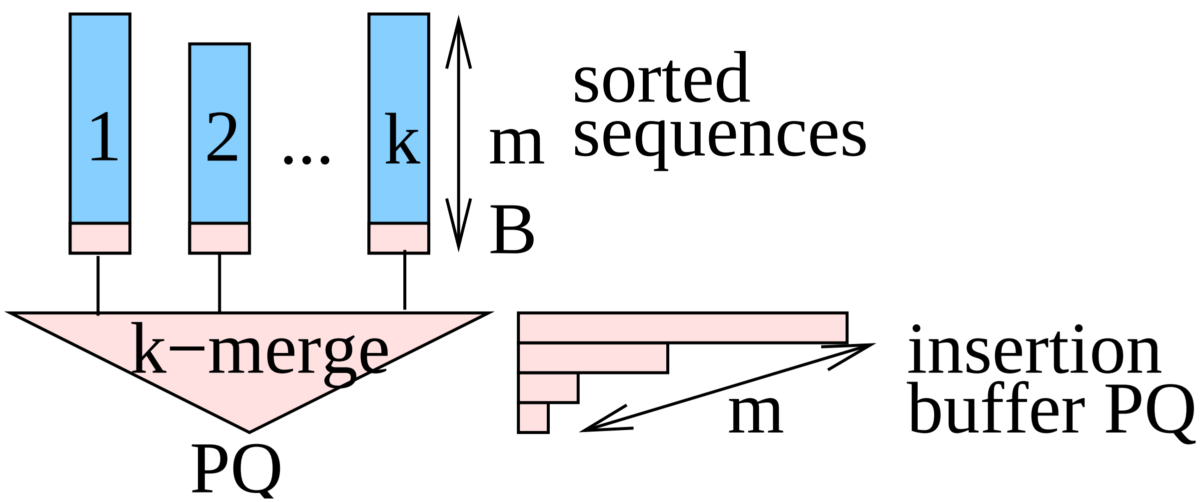
Analyse – Vergleiche (Maß für interne Arbeit)

deleteMin: $1 + \mathrm{O}(\max(\log k, \log m)) = \mathrm{O}(\log m)$

genauere Argumentation: amortisiert $1 + \log k$ bei geeigneter PQ

insert: $\approx m \log m$ alle m Ops. Amortisiert $\log m$

Insgesamt nur $\log km$ amortisiert !



Große Queues

$$\approx \frac{2n}{B}\left(1+\left\lceil \log_{M/B}\frac{n}{M}\right\rceil\right)$$

I/Os für n Einfügeoperationen

$O(n \log n)$ Arbeit.

[Sanders 1999].

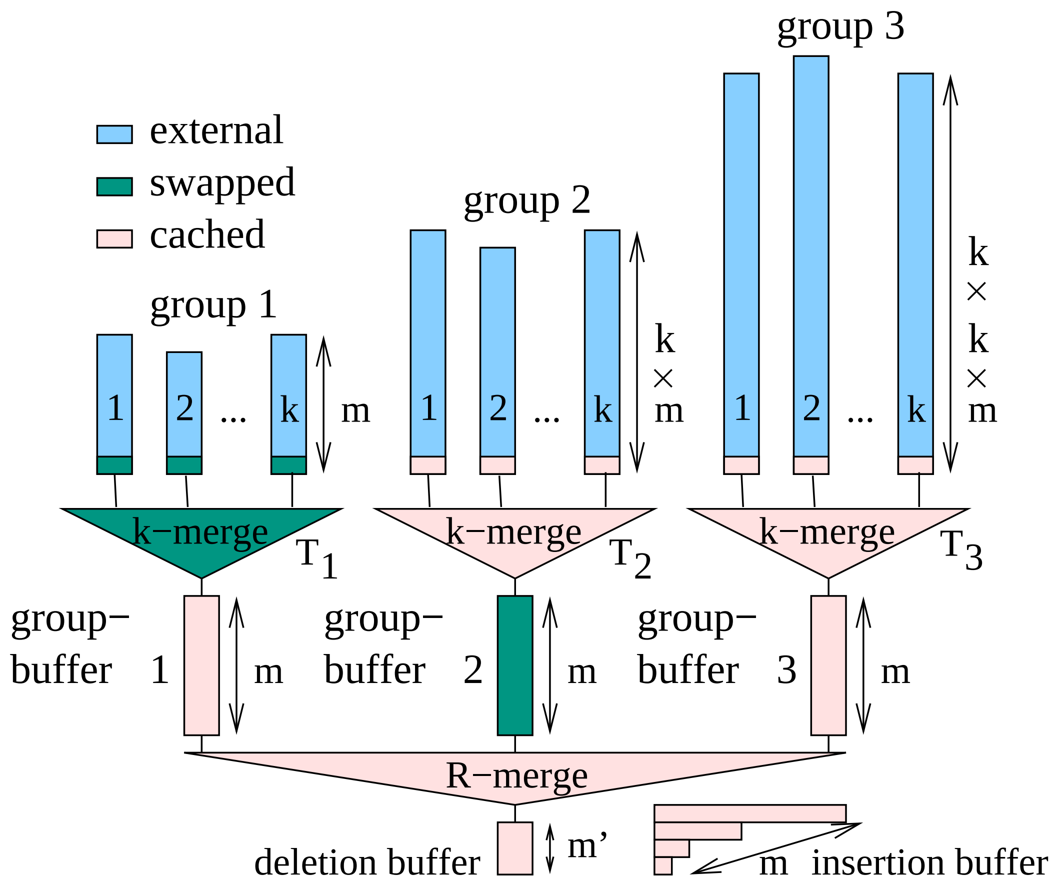
deleteMin:

“amortisiert umsonst”.

Details:

Vorlesung

Algorithm Engineering.



Experiments

Keys: random 32 bit integers

Associated information: 32 dummy bits

Deletion buffer size: 32

Group buffer size: 256

Merging degree k: 128

Near optimal
: performance on
all machines tried!

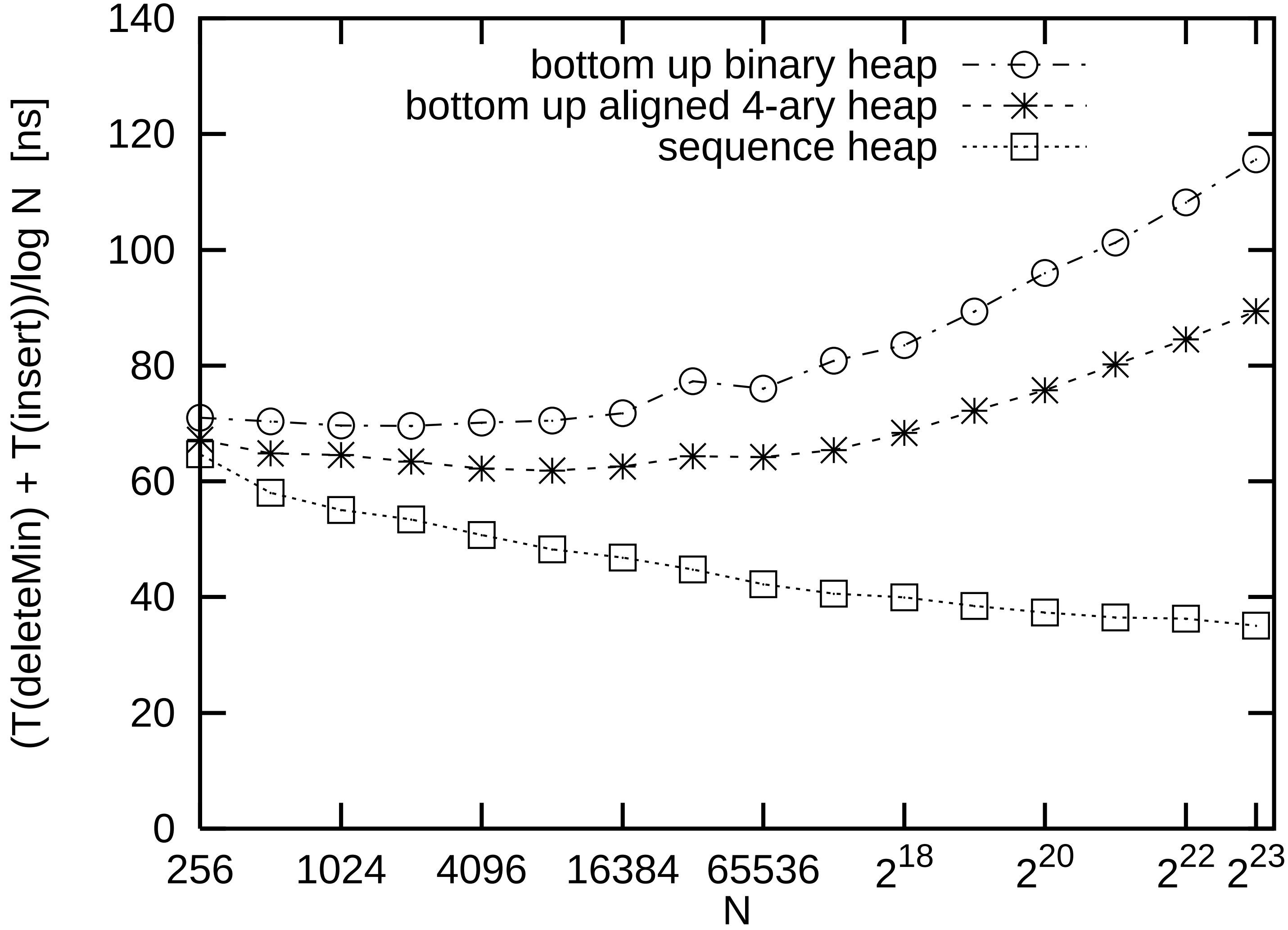
Compiler flags: Highly optimizing, nothing advanced

Operation Sequence:

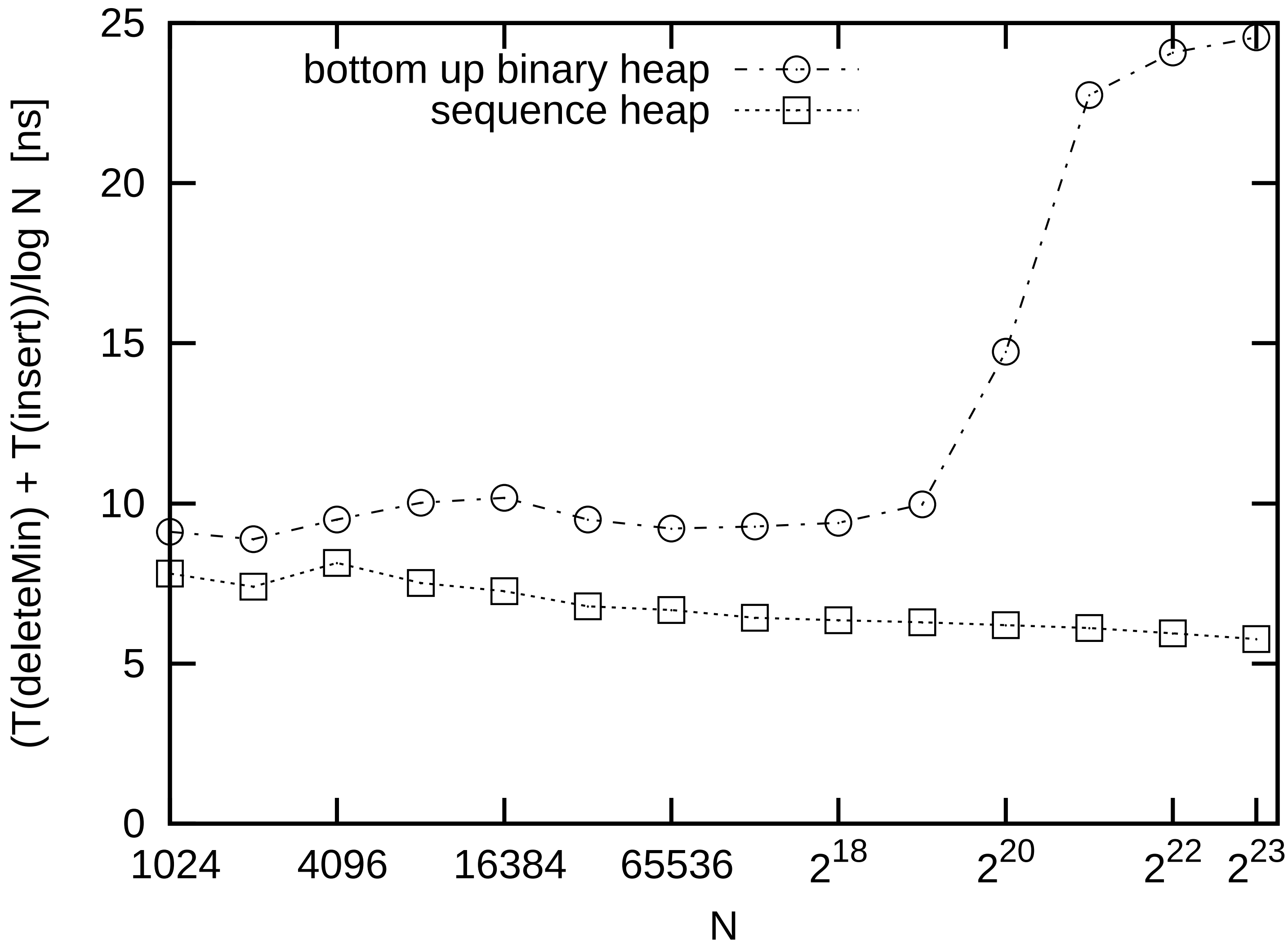
$$(\text{Insert-DeleteMin-Insert})^N(\text{DeleteMin-Insert-DeleteMin})^N$$

Near optimal performance on all machines tried!

Alpha-21164, 533 MHz



Core2 Duo Notebook, 1.??? GHz



Minimale Spannbäume

Semiexterner Kruskal

Annahme: $M = \Omega(n)$ konstant viele Maschinenworte pro Knoten

Procedure seKruskal$(G = (1..n, E))$
  sort E by decreasing weight // sort(m) I/Os
  Tc : UnionFind(n)
  foreach $(u, v) \in E$ in ascending order of weight **do**
    if Tc.find$(u) \neq$ Tc.find(v) **then**
      output $\{u, v\}$
      Tc.union(u, v) // link reicht auch

Externe MST-Berechnung

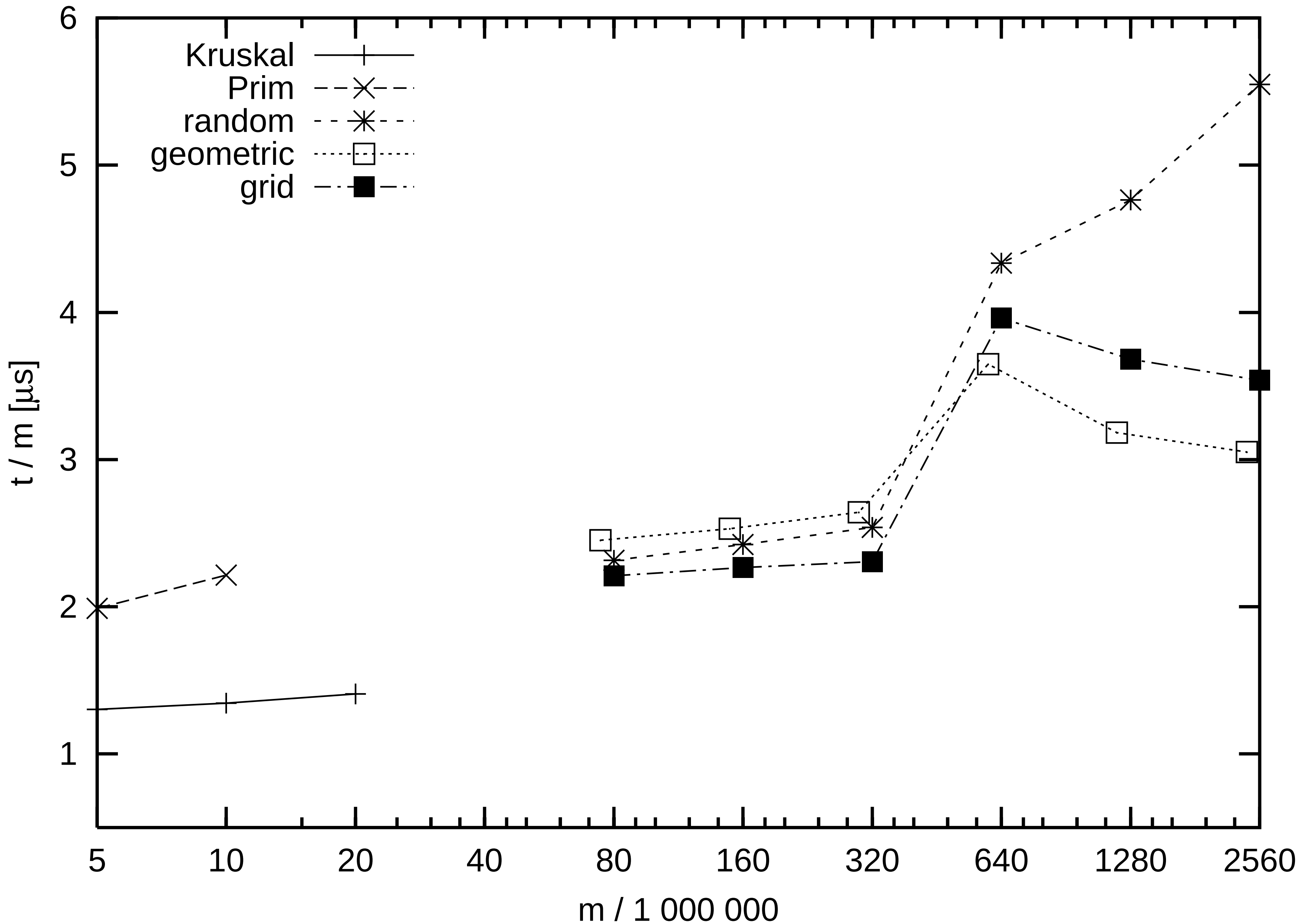
- Reduziere Knotenzahl mittels Kontraktion von MST-Kanten

  Details: Vorlesung Algorithm Engineering, Sibeyn's Algorithmus.

  Implementierung $\approx$ Sortierer $+$ ext. Prioritätsliste $+$ 1 Bildschirmseite. (STXXL Bibliothek)

- benutze semiexternen Algorithmus sobald $n < M$.

Beispiel, Sibeyn's algorithm, $m \approx 2n$



Mehr zu externen Algorithmen – Basic Toolbox ?

Externe Hashtabellen: geht aber 1 I/O pro Zugriff

Suchbäume: $(a, 2a)$-Bäume mit $a = \Theta(B) \rightsquigarrow \log_B n$ I/Os für Basisoperationen. Brot-und-Butter-Datenstruktur für Datenbanken. Inzwischen auch in Dateisystemen. Viel Tuning: Große Blätter, Caching, ...

BFS: OK bei kleinem Graphdurchmesser

DFS: noch schwieriger. Heuristiken für den semiexternen Fall

kürzeste Wege: ähnlich BFS.