Algorithmen / Algorithms II

Peter Sanders

Exercise:

Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:

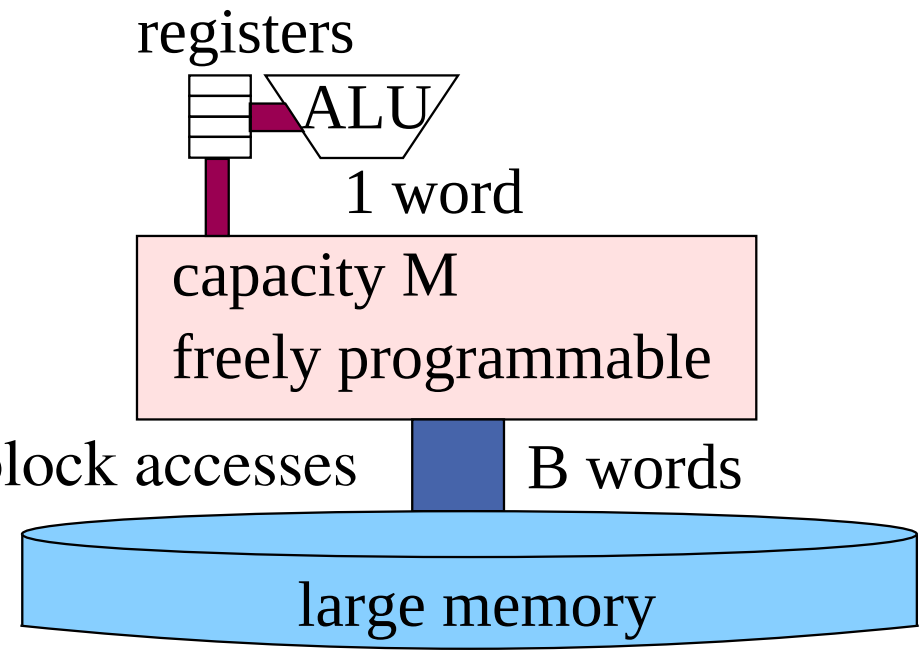
`http://algo2.iti.kit.edu/AlgorithmenII_WS20.php`

7 External Algorithms

7.1 The External Memory Model

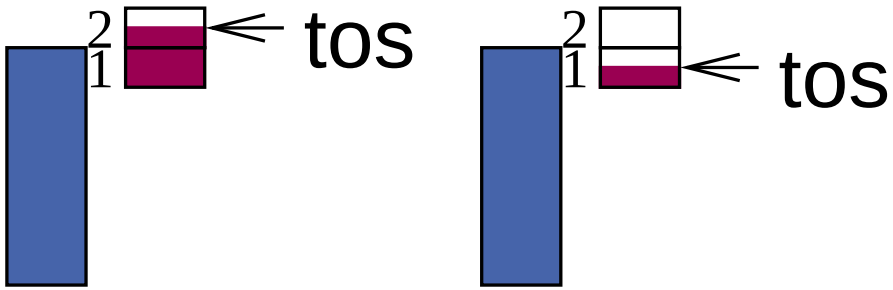
M: Fast memory of size M

B: Block size



Analysis: Count number of block accesses

7.2 External Stacks



- File containing blocks
- 2 internal buffers

push: Write into buffer if space is available.

Otherwise, write buffer one into the file, (block-level push)

and rename buffers — 1 and 2 swap places.

pop: Read from buffer if not empty.

Otherwise, read buffer one from the file. (block-level pop)

Analysis: Amortized $O(1/B)$ I/Os per operation

Exercise 1: Proof

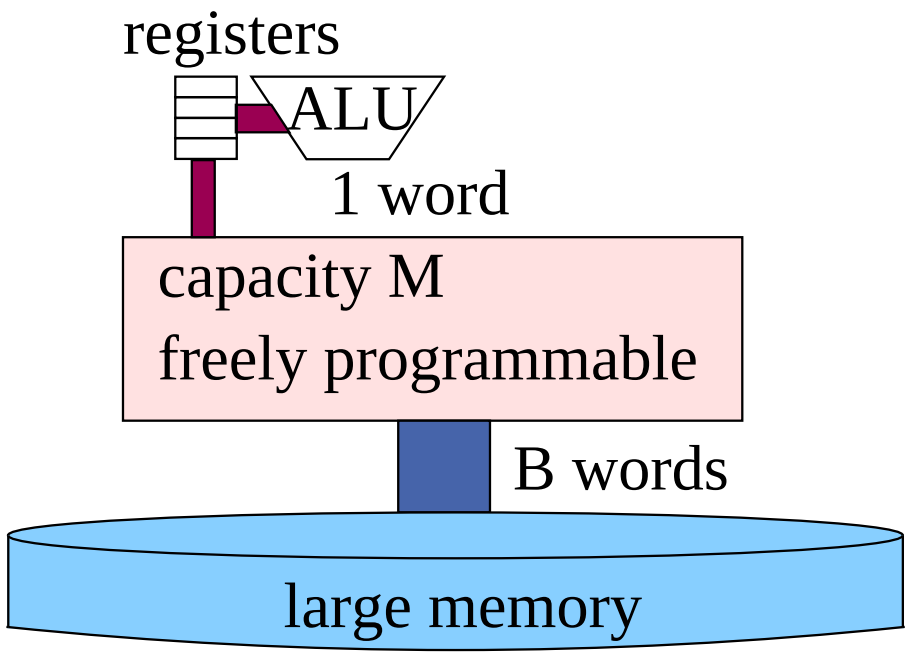
Exercise 2: Efficient implementation without unnecessary copies

7.3 External Sorting

n: Input size

M: Size of fast memory

B: Block size



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

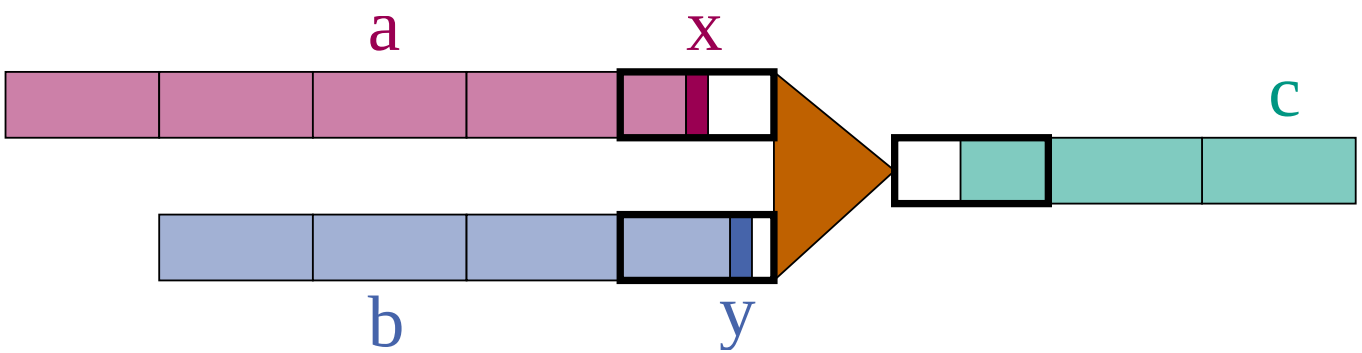
x := a.readElement // Assume emptyFile.readElement$= \infty$

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



External (Binary) Merging – I/O Analysis

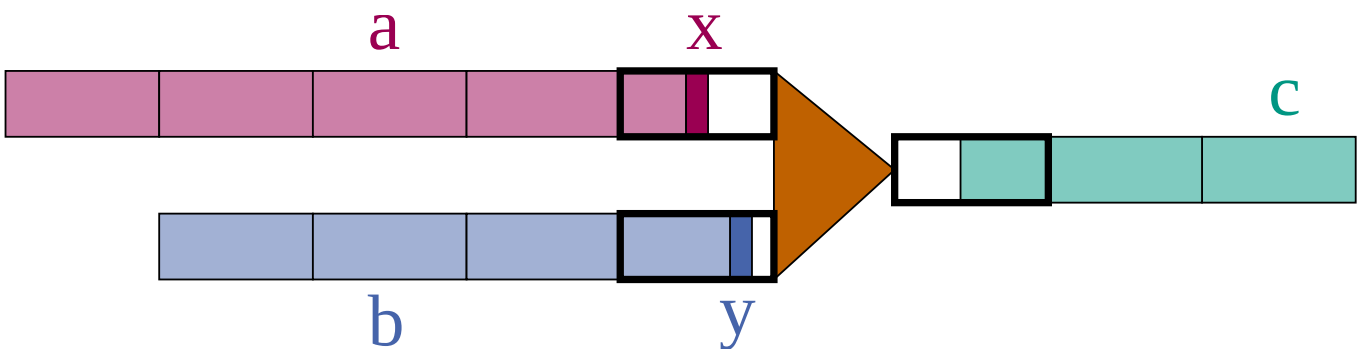
Read file a: $\lceil |a|/B \rceil \leq |a|/B + 1$

Read file b: $\lceil |b|/B \rceil \leq |b|/B + 1$

Write file c: $\lceil (|a| + |b|)/B \rceil \leq (|a| + |b|)/B + 1$

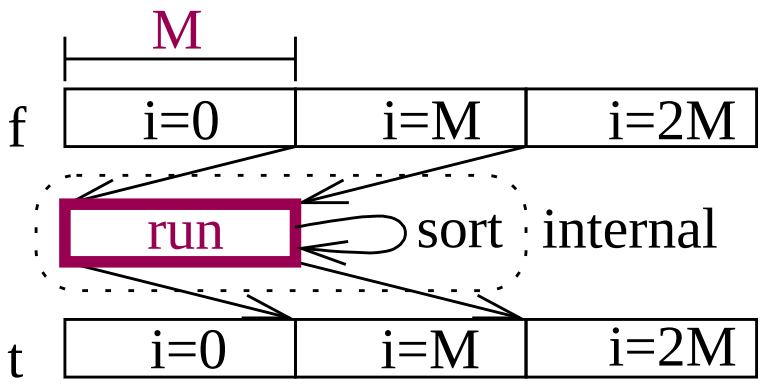
Total: $\leq 3 + 2\frac{|a| + |b|}{B} \approx 2\frac{|a| + |b|}{B}$

Condition: We need 3 buffer blocks, therefore $M > 3B$.



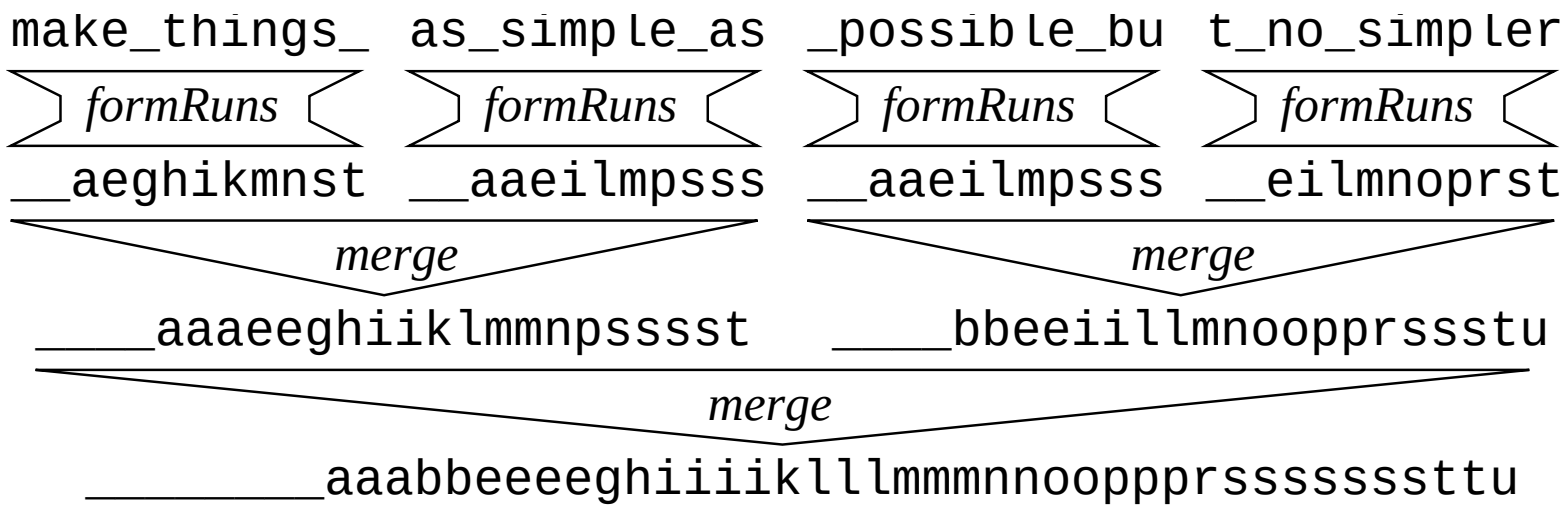
Run Formation

Sort input in chunks of size M.



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

Sorting via External Binary Merging



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

Numerical Example: PC 2019

$n = 2^{40}$ bytes

$M = 2^{34}$ bytes

$B = 2^{22}$ bytes

I/O needs 2^{-4} s

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

Idea: 7 iterations $\rightsquigarrow$ 2 iterations

Multi-way Merge

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

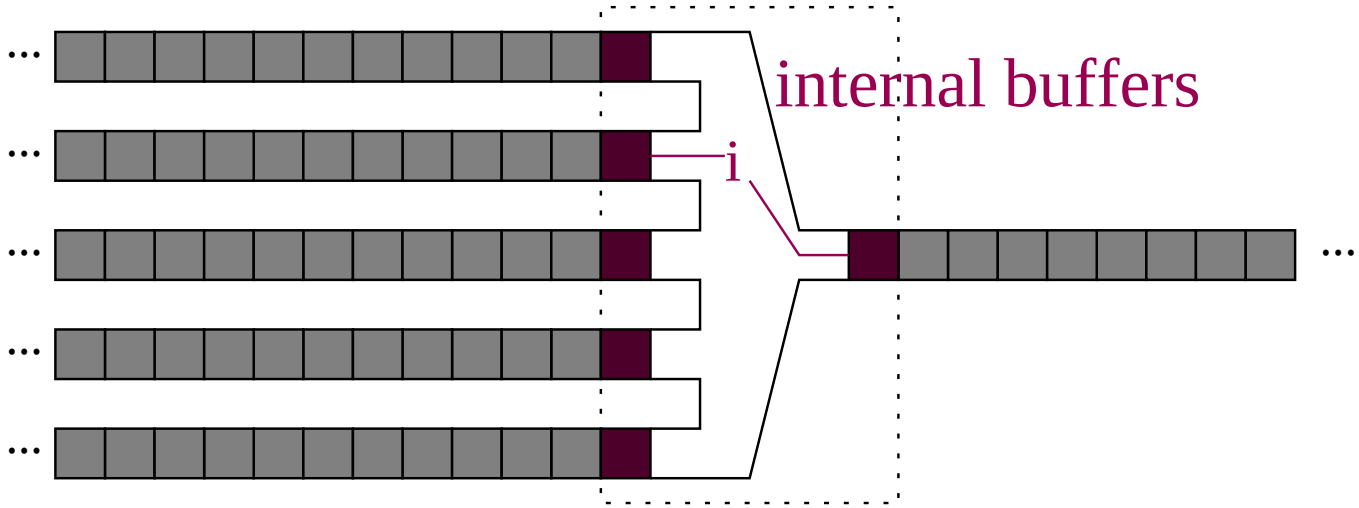
  for $i := 1$ **to** k **do** $x_i := a_i$.readElement

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

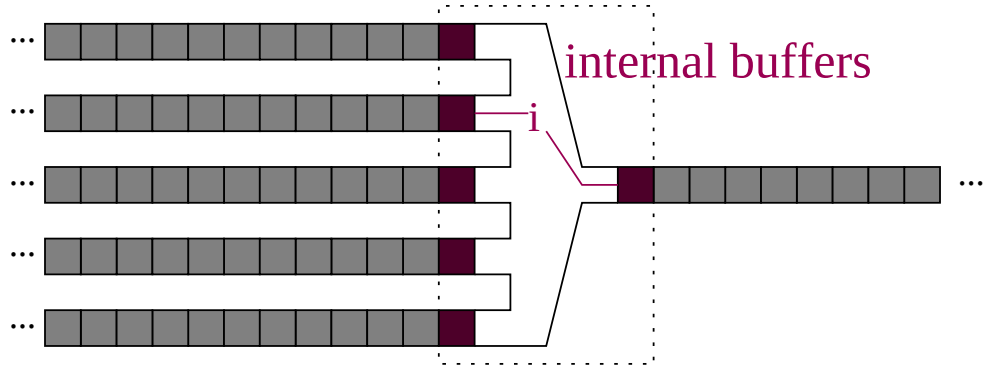
    find $i \in 1..k$ that minimizes x_i // no I/Os!, $O(\log k)$ time

    c.writeElement(x_i)

    $x_i := a_i$.readElement



Multi-way Merge – Analysis



I/Os:

Read file a_i: $\approx |a_i|/B$

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

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

Condition: We need $k+1$ buffer blocks, therefore $k+1 < M/B$

(simplified to $k < M/B$ in the following)

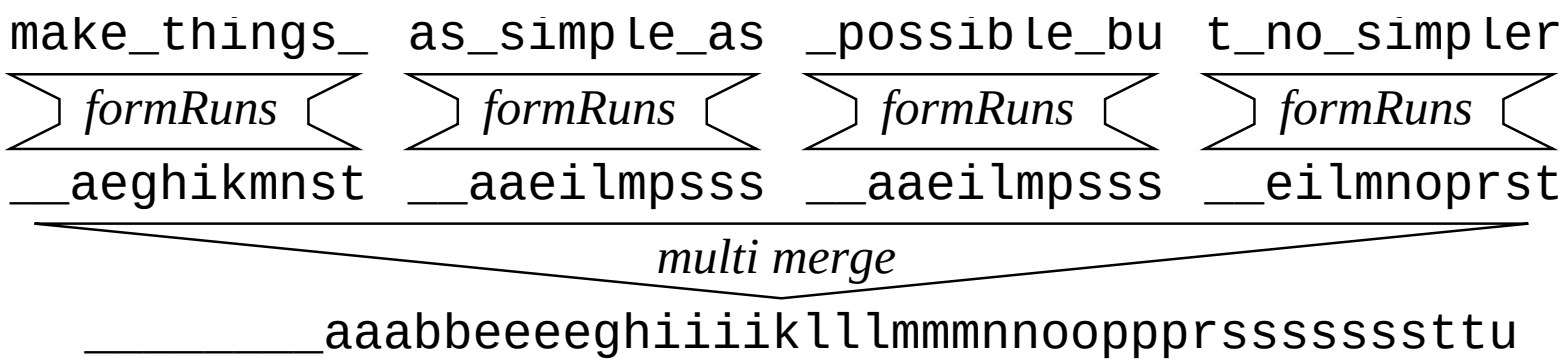
Internal work: (use priority queue!)

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

Sorting via Multi-way Merging

- Sort $\lceil n/M \rceil$ runs with M elements each. $2n/B$ I/Os
- Merge M/B runs at a time. $2n/B$ I/Os
- Repeat until only one run remains. $\times \left\lceil \log_{M/B} \frac{n}{M} \right\rceil$ Merge phases

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



Sorting via Multi-way Merging

Internal work:

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

More than one merge phase?

Not for the hierarchy of main memory and hard disk, since:

$$\underbrace{\frac{M}{B}}_{>1000} > \underbrace{\frac{\text{RAM Euro}/\text{bit}}{\text{Disk Euro}/\text{bit}}}_{\approx 150}$$

More on External Sorting

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

[Aggarwal Vitter 1988]

Upper bound $\approx \frac{2n}{DB}\left(1+\left\lceil \log_{M/B}\frac{n}{M}\right\rceil\right)$ I/Os (expected)

for D parallel disks

[Hutchinson Sanders Vitter 2005, Dementiev Sanders 2003]

Open question: deterministic?

External Priority Queues

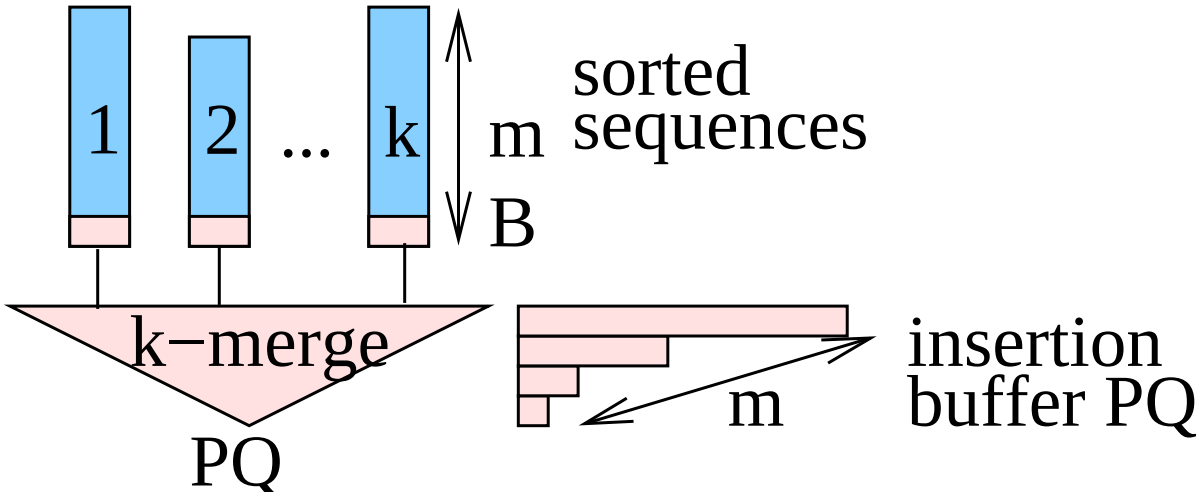
Problem: Binary heaps require

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

We would like:

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

Medium-sized PQs – $km \ll M^2/B$ Insertions



Insert: Initially into insertion buffer.

Overflow $\longrightarrow$

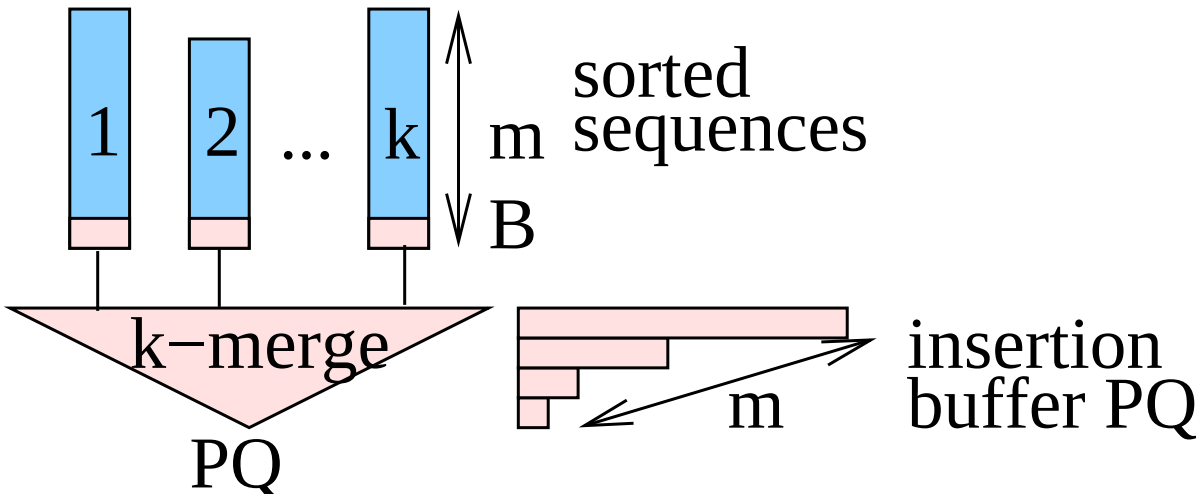
sort; flush; smallest key in merge-PQ

Delete-Min: deleteMin from the PQ with smallest $\min$

Analysis – I/Os

deleteMin: each element is read $\leq 1\times$, together with B others –

amortized $1/B$ penalty for insert.



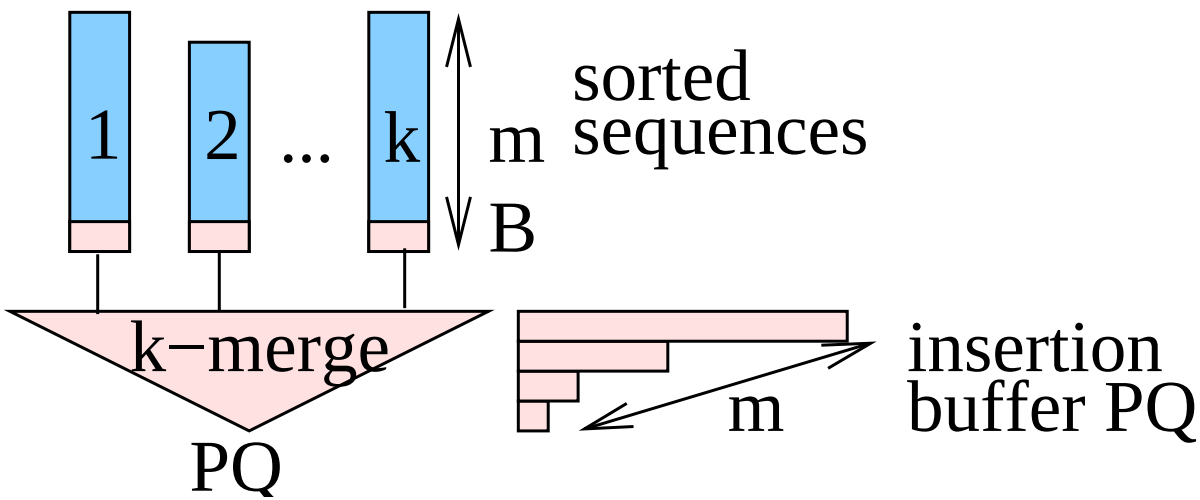
Analysis – Comparisons (Measure for Internal Work)

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

more exact argumentation: amortized $1 + \log k$ for suitable PQ

insert: $\approx m \log m$ every m Ops. Amortized $\log m$

In total, only $\log km$ amortized!



Large Queues

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

I/Os for n insertions

$O(n \log n)$ work

[Sanders 1999]

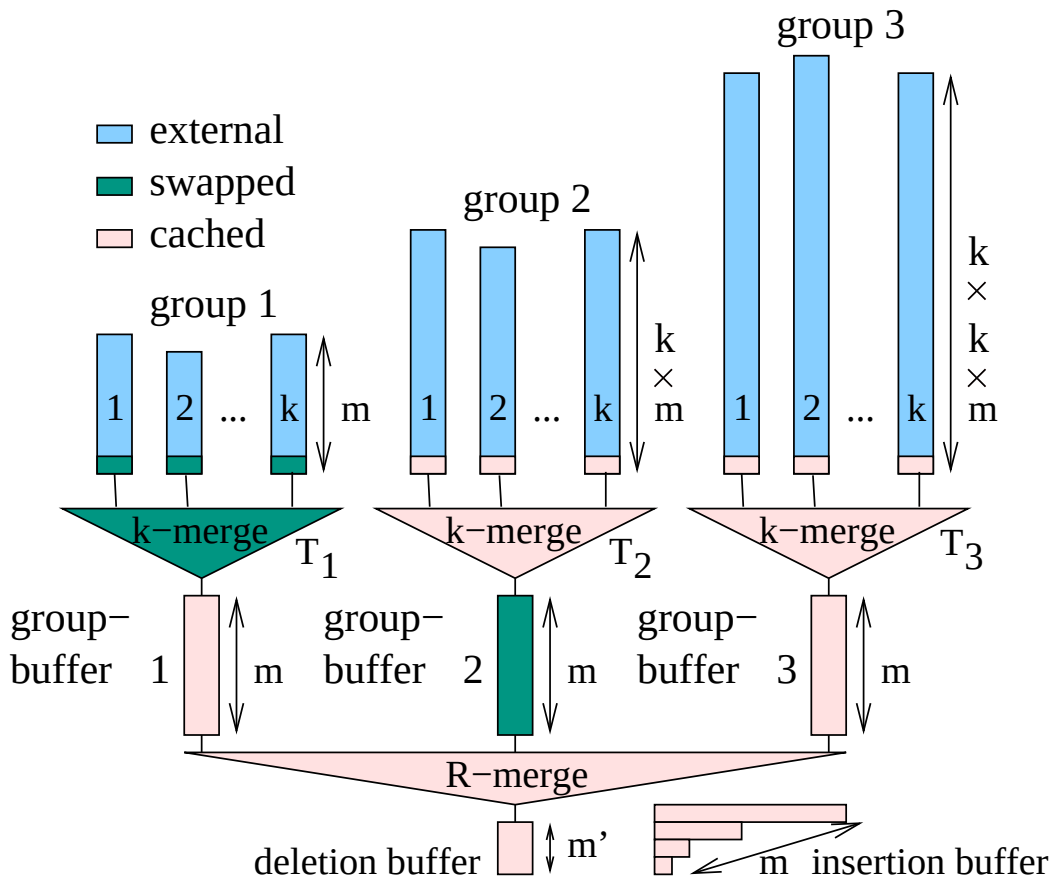
deleteMin:

"amortized for free"

Details:

Algorithm Engineering

lecture



Experiments

Keys: random 32 bit integers

Associated information: 32 dummy bits

Deletion buffer size: 32

Group buffer size: 256

Merging degree k: 128

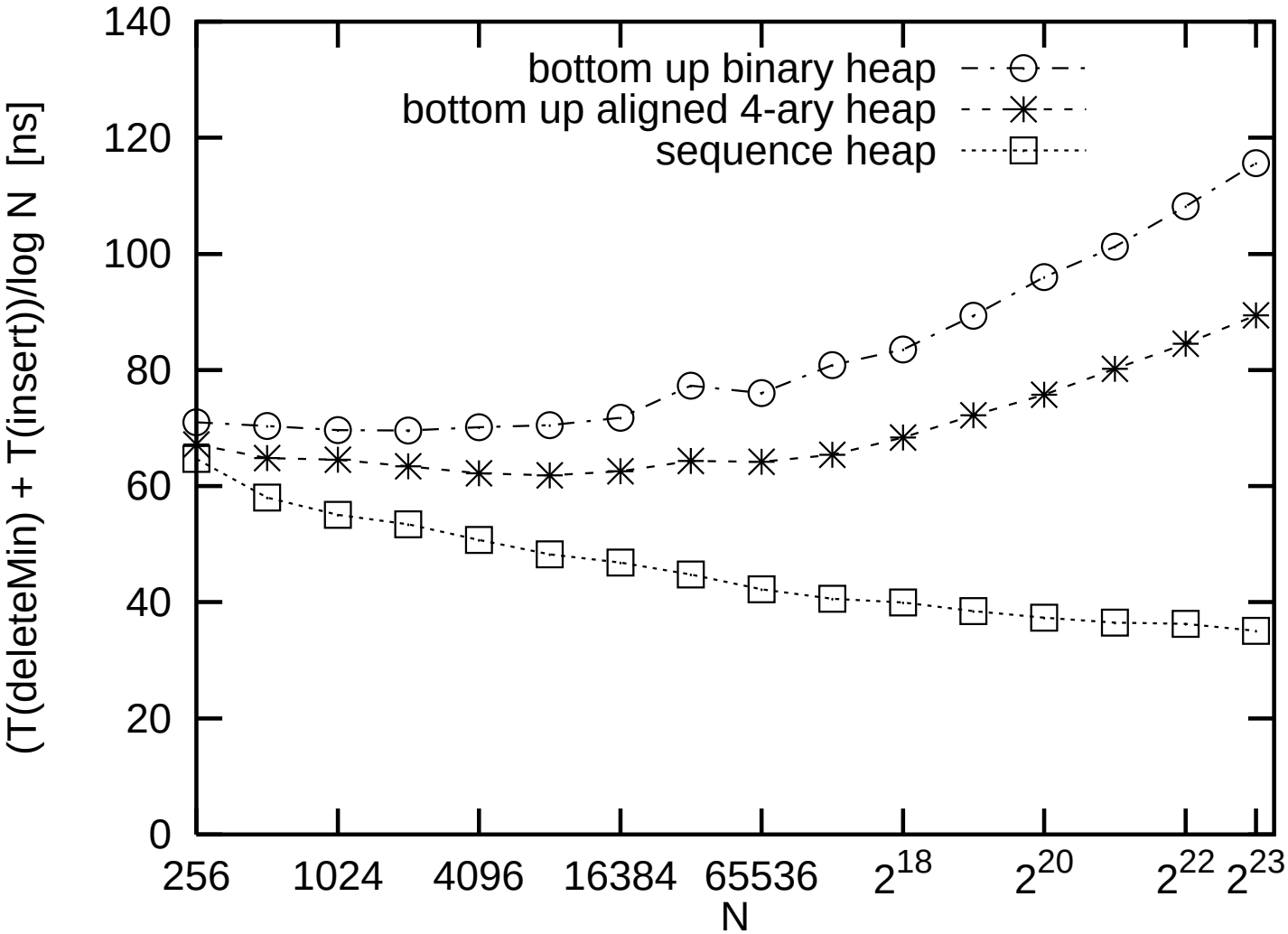
Compiler flags: Highly optimizing, nothing advanced

Operation Sequence:

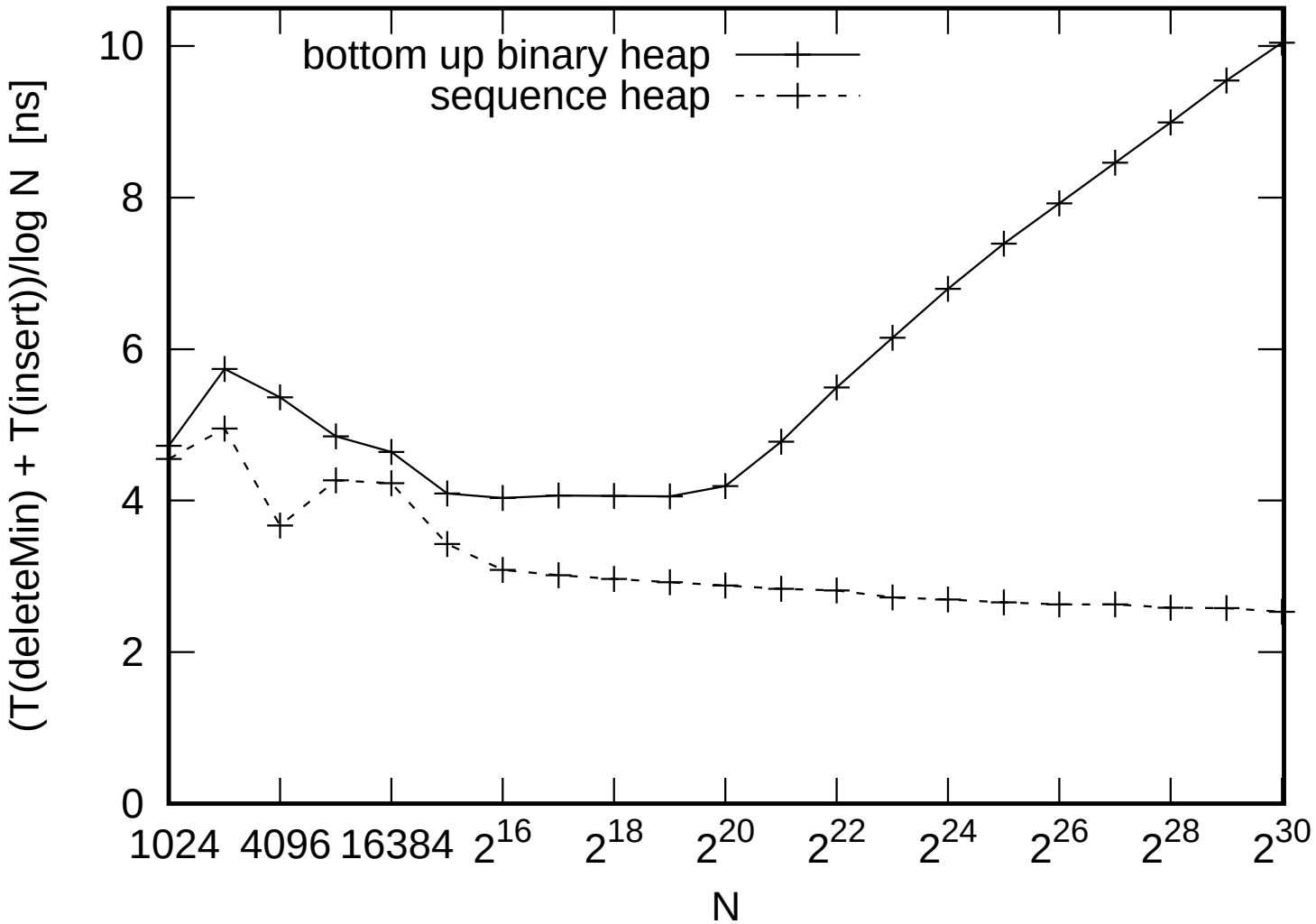
$$\big(\text{Insert-DeleteMin-Insert}\big)^N\big(\text{DeleteMin-Insert-DeleteMin}\big)^N$$

Near optimal performance on all machines tried!

Alpha-21164, 533 MHz, 1997



AMD Ryzen 1800X, 16MB L3, 3.6 GHz, 2017



Open Problem:

Faster cache-efficient PQs

Multi-way merging $\longrightarrow$ Multi-way distribution?

Factor 2–3 again?

Minimal Spanning Trees

Semi-external Kruskal

Assumption: $M = \Omega(n)$ constant amount of machine words per node

Procedure seKruskal$(G = (1..n, E))$

sort E by decreasing weight // $\text{sort}(m)$ 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 would suffice

External MST Computation

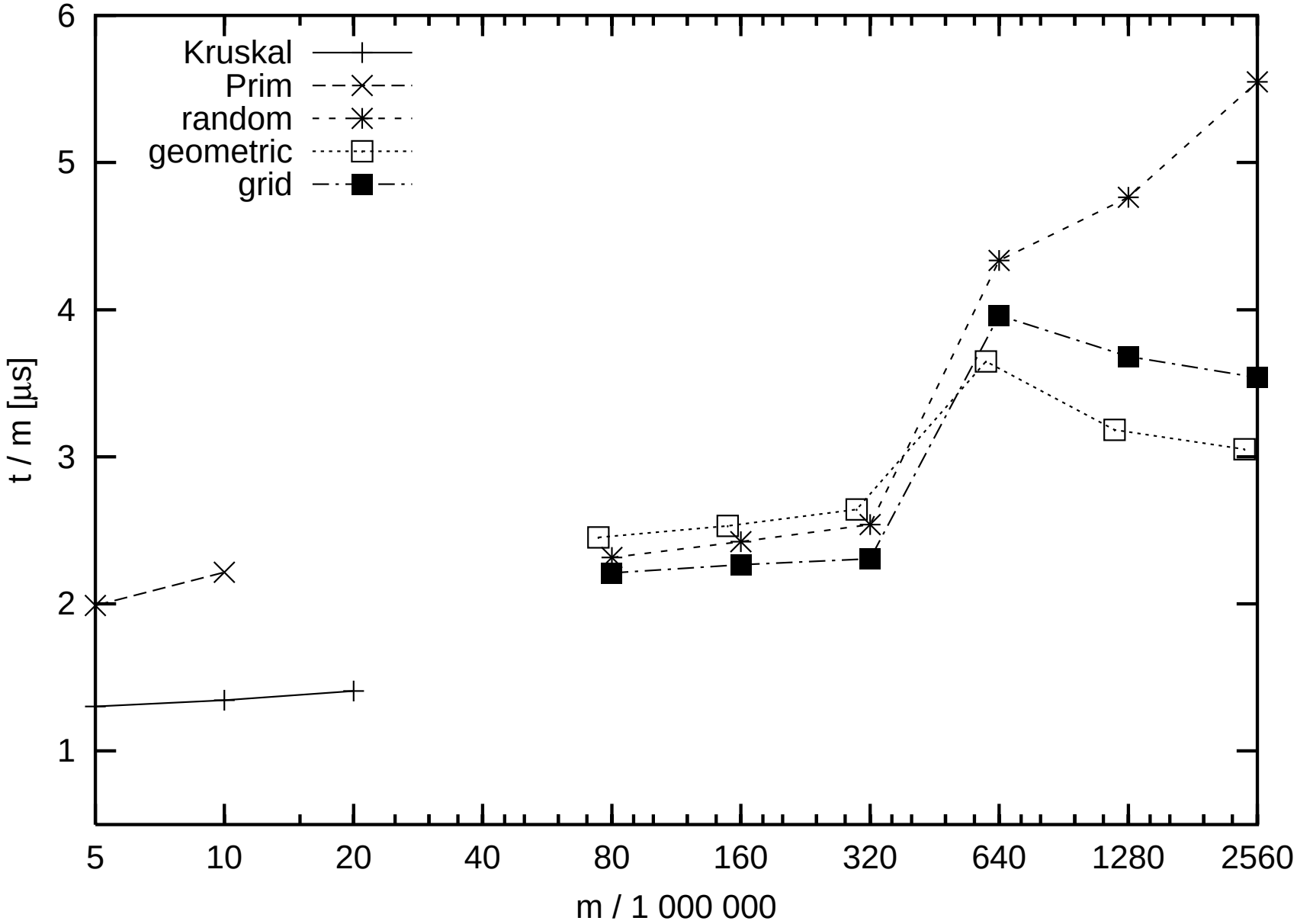
- Reduce number of nodes through Contraction of MST edges

  Details: Algorithm Engineering lecture, Sibeyn's Algorithm.

  Implementation $\approx$ Sorter $+$ ext. priority queue $+$ 1 screen of code. (STXXL library)

- Use semi-external algorithm once $n < M$.

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



More on External Algorithms – Basic Toolbox?

External Hash Tables: works, but 1 I/O per access

Search Trees: $(a, 2a)$-Trees with $a = \Theta(B) \rightsquigarrow \log_B n$ I/Os for basic operations. Bread-and-butter data structure for databases. By now also in file systems. A lot of tuning: Large leaves, caching, ...

BFS: OK for low graph diameter

DFS: Even harder. Heuristics for the semi-external case

Shortest Paths: similar to BFS