Worst-Case Efficient External-Memory Priority Queues

Gerth Stølting Brodal

Max–Planck–Institut für Informatik

Saarbrücken

Germany

Jyrki Katajainen

Datalogisk Institut

Københavns Universitet

Denmark

July 1998

Priority Queues

Maintain a set of N elements from a totally ordered universe under

INSERT(x) – Insert element x into the set

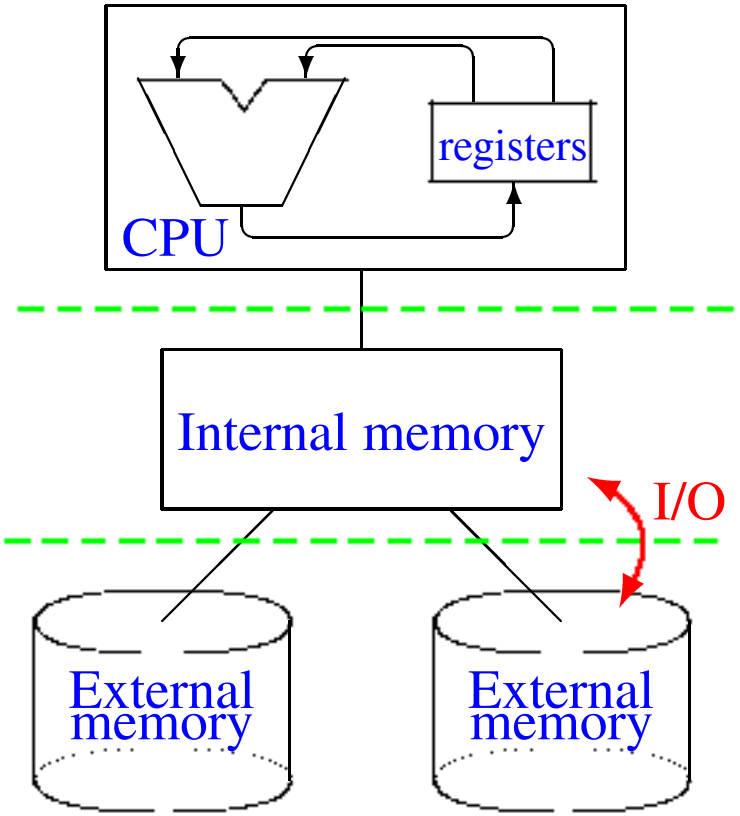
DELETEMIN$()$ – Delete and return the minimum of the set

INSERT and DELETEMIN can be implemented with $\Theta(\log_2 N)$ comparisons

– Williams, 1964

External Memory Model

– Aggarwal, Vitter, 1988



M = Size of the internal memory

B = I/O block size

N = # elements in the priority queue

Complexity measures

Internal : # comparisons

External : # I/Os

Assumptions

$$M \geq 23B\,, \quad B \geq \log_{M/B} \frac{N}{M}$$

External-Memory Merge Sorting

Theorem N elements stored in $\mathcal{O}\left(\frac{N}{B}\right)$ blocks can be sorted with $\Theta(N \log_2 N)$ comparisons and $\Theta\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$ I/Os.

Algorithm:

(1) Sort in internal memory all runs of M elements

(2) In $\log_{M/B} \frac{N}{M}$ iterations apply a $\Theta\left(\frac{M}{B}\right)$-ary merging algorithm to obtain lists of length $M \cdot \left(\frac{M}{B}\right)^i$.

The sorting algorithm is optimal w.r.t. both comparisons and I/Os.

– Aggarwal, Vitter, 1988

Worst-Case Effeciency

Internal

Each priority queue operation should require $\Theta(\log_2 N)$ comparisons in the worst-case

External

The I/Os should be divided evenly among the operations, i.e., one I/O for every $\Theta\left(B/\log_{M/B}\frac{N}{B}\right)$th operation.

External-Memory Priority Queues

Binary heap *– Williams, 1964*

– Random memory accesses imply one I/O every $\mathcal{O}(1)$ CPU operation.

Fishspear *– Fischer, Patterson, 1994*

– $\mathcal{O}\left(\frac{N}{B}\log_2 N\right)$ I/Os for N operations. Uses $\mathcal{O}(1)$ push down stacks.

Heap + B elements/node *– Wegner, Teuhola, 1989*

– $\mathcal{O}\left(\log_2 \frac{N}{B}\right)$ I/Os for every Bth operation. Optimal for $M = \mathcal{O}(B)$.

Buffer tree * *– Arge, 1995*

– $\mathcal{O}\left(\frac{N}{B}\log_{M/B}\frac{N}{M}\right)$ I/Os for N operations.

Heap with buffers * *– Fadel et al., 1997*

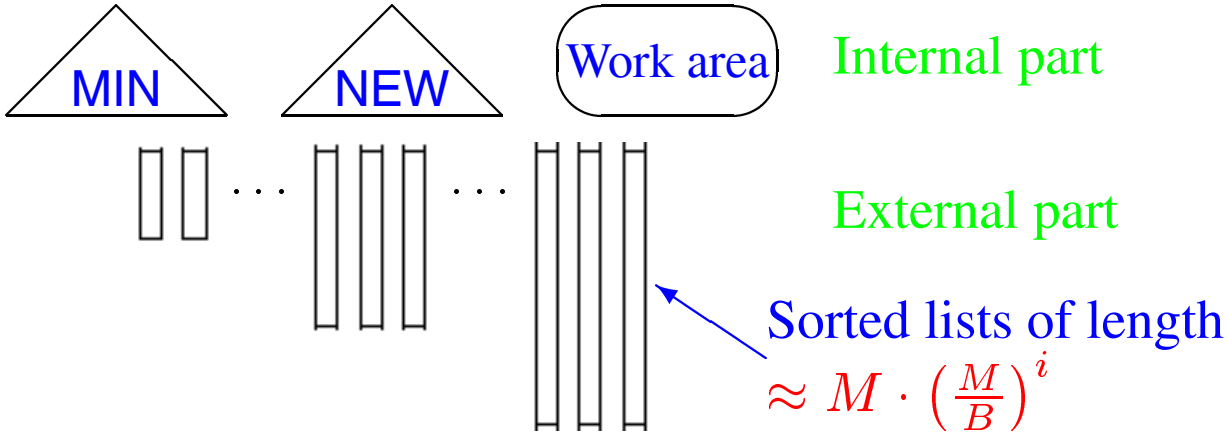
– Similar bounds as for buffer trees.

* No worst-case guarantee on the individual operations

New Result

Theorem DELETEMIN and INSERT can be done with $\Theta(\log N)$ comparisons and one I/O for every $\Theta(B/\log_{M/B}\frac{N}{B})$th operation.

Outline of the Data Structure



MIN stores the overall $\mathcal{O}(M)$ smallest elements $\rightarrow$ fast DELETEMIN.

NEW stores the most recently inserted $\mathcal{O}(M)$ elements $\rightarrow$ fast INSERT.
[small elements are inserted directly into MIN]

The external part stores exactly B elements per block
[all elemenst are larger than the elements in MIN]

Operations

$K = \Theta(M)$

INSERT(x)

– If $x \leq \max$ MIN then swap x and $\max$ MIN
– Insert x into NEW
– If $|\text{NEW}| \geq K$ then perform BATCHINSERT
with K elements from NEW.

DELETEMIN()

– If MIN $= \emptyset$ then perfrom BATCHDELETE
Merge the result with NEW
Move the K smallest elements to MIN
– Delete and return $\min$ MIN

BATCHINSERT

– Create a new list of length K and assign it rank 1
– $i := 1$
– while #list of rank $i = \frac{M}{B}$ do
Merge the lists with rank i
Assign the resulting list rank $i + 1$
$i := i + 1$

BATCHDELETE

– Fetch the K smallest elements from external
memory to internal memory

rank $L = \lfloor \log_{M/B} \frac{|L|}{B} \rfloor$, #lists $\leq \frac{M}{B} \log_{M/B} \frac{N}{M}$

Amortized Result

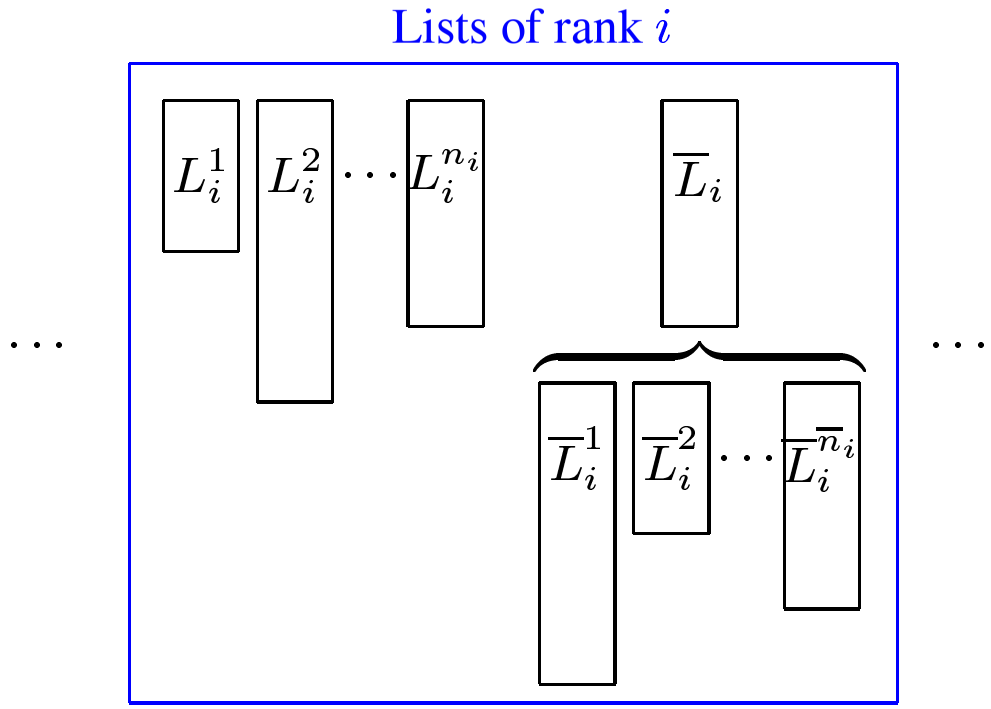
Lemma The described data structure supports INSERT and DELETEMIN with amortized $\Theta(\log N)$ comparisons and $\Theta\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os.

[Identical bounds follow by using buffer trees and external heaps]

Worst-Case Bounds?

- Incremental list merging — *– Thorup, 1996*
- Incremental batch operations
- Start BATCHDELETE before MIN gets empty.

Incremental List Merging



MERGESTEP(i)

– Perform K steps of the list merging at rank i.

– If the merging is finished, then make the resulting list $\bar{L}_i$ a list of rank $i+1$ (list promotion)

BATCHINSERT

– Create a new list with rank one

– $\forall i$: MERGESTEP(i)

Deletions

BATCHDELETE deletes and returns the K least elements of the lists.

$\Downarrow$

The length of the lists decreases.

$\Downarrow$

The maximum rank increases.

Possible solution: Incremental global rebuilding.

Our solution:

1) If the incremental merging of rank i finishes, but $|\bar{L}_i| < K\left(\frac{M}{B}\right)^i$, then we do not promote $\bar{L}_i$ to rank $i+1$.
2) Let R be the maximum rank of a list. If the resulting list $\bar{L}_R < K\left(\frac{M}{B}\right)^{R-1}$, then $\bar{L}_R$ gets rank $R-1$.
3) If K elements are deleted from $\bar{L}_i$, we perform MERGESTEP(i).
4) Always perform MERGESTEP(R) after BATCHDELETE

$\Downarrow$

$R = \mathcal{O}\left(\log_{M/B}\frac{N}{M}\right)$

Conclusion

- The first worst case external-memory priority-queue implementation.

Open Problems

- Is the definition of "worst-case" reasonable in practice ?
- Experimental studies.
- Can buffer trees be deamortized, i.e., how can buffer trees support operations with worst-case $\mathcal{O}\left(\log_{M/B}\frac{N}{B}\right)$ I/Os for B operations ?