Worst-Case Efficient External-Memory Priority Queues

Gerth Stølting Brodal
MPI Saarbrücken

Joint work with

Jyrki Katajainen
University of Copenhagen

Priority Queues

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

Insert(x) - Insert element x into the set

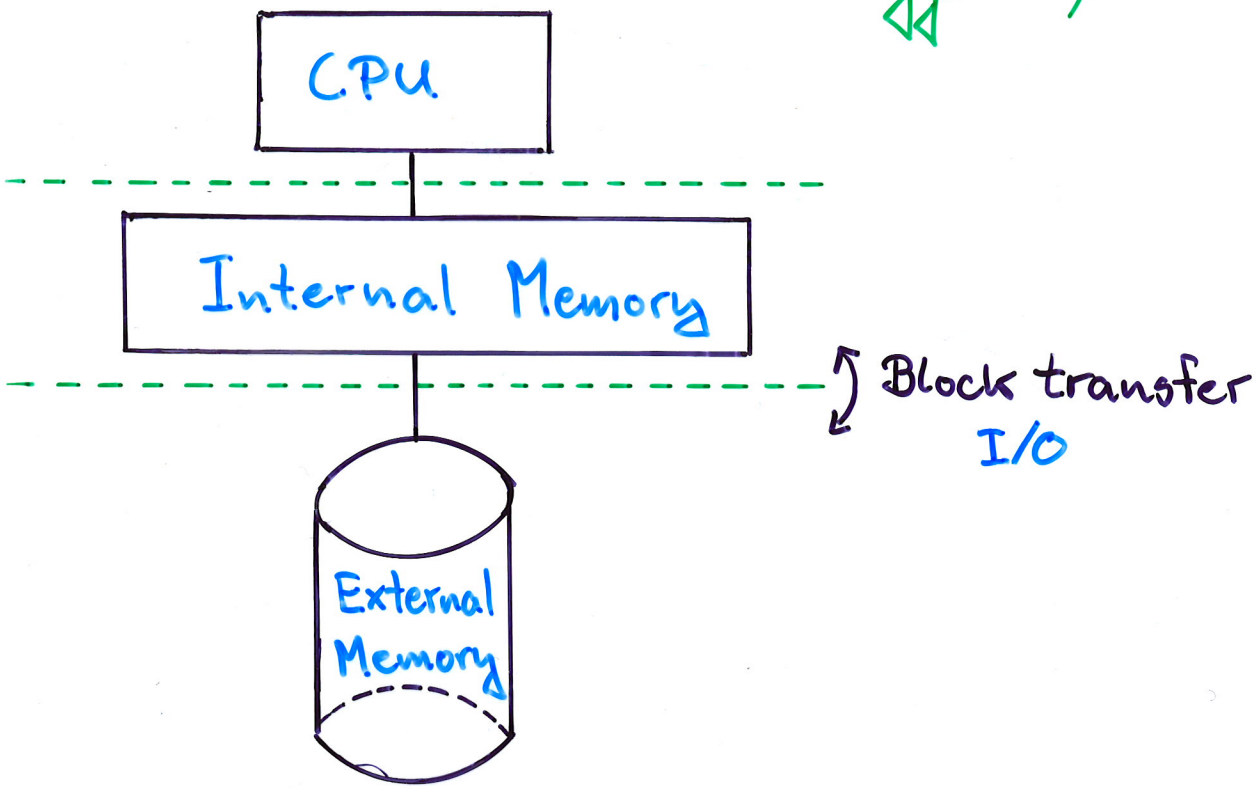
DeleteMin() - Delete and return the minimum of the set

Insert and Delete can be implemented with $\Theta(\log_2 N)$ comparisons

Williams '64

External Memory Model

Aggarwal, Vitter '88



M = Size of the internal memory
B = I/O block size
N = # elements in the priority queue

Assumptions:

$M \geq 2B$, $B \geq \log_{\frac{M}{B}} \frac{N}{M}$

Complexity measures:

Internal : # comparisons
External : # I/Os

Preliminaries

Merging: $\Theta(\frac{M}{B})$ sorted lists containing N elements stored in $O(\frac{N}{B})$ blocks can be merged into one list with $\Theta(\frac{N}{B})$ I/Os and $\Theta(N \cdot \log_2 \frac{M}{B})$ comparisons.

Proof: $\Theta(\frac{M}{B})$-ary merging

Sorting: N elements in $O(\frac{N}{B})$ blocks can be sorted with $\Theta(\frac{N}{B} \cdot \log_{M/B} \frac{N}{M})$ I/Os and $\Theta(N \log_2 N)$ comparisons

Proof: $\Theta(\frac{M}{B})$-ary mergesort

(i) Sort in internal memory all sequences of M elements.

(ii) In $\log_{M/B} \frac{N}{M}$ iterations apply the above merging algorithm to obtain lists of length $M \cdot (\frac{M}{B})^i$

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

Aggarwal, Vitter '88

Worst-Case Effeciency

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

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

External-Memory Priority Queues

- Heaps Williams '64

  The random memory accesses implies that each CPU operation causes an I/O...

- Fishspear Fischer, Patterson '94

  $\Theta(\frac{N}{B} \cdot \log_2 N)$ I/Os for N operations.
  Uses $O(1)$ push down stacks

- Heap + B elements/node Wegner, Teuhola '89

  $O(\log_2 \frac{N}{B})$ I/Os for every B'th operation.
  Optimal for $M = O(B)$

- Buffer tree Arge '95

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

- Heap with buffers Fadel et al. '97

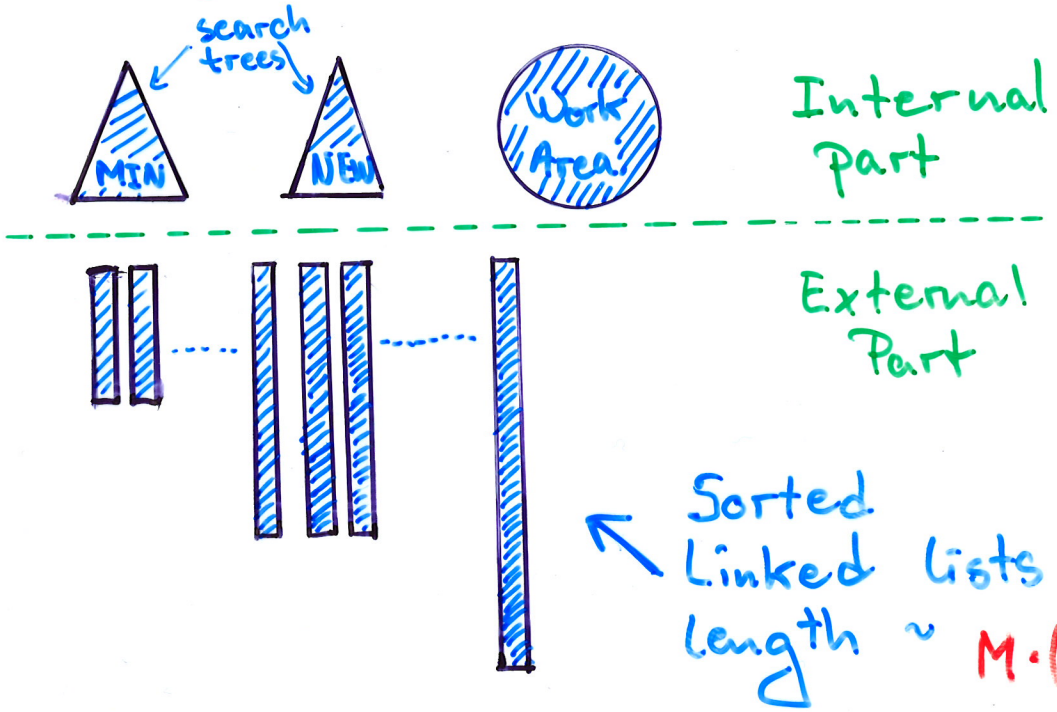
  Same bounds as for buffer trees

- No worst-case guarantee on the individual operations

New Result

DeleteMin and Insert can be done with worst-case $\Theta(\log n)$ comparisons per operation, and 1 I/O for every $\Theta(B/\log_{M/B}\frac{N}{B})$ operations, which is the best possible.

Outline of the Data Structure



- Min stores the overall $O(M)$ smallest elements → fast DeleteMin
- New stores the most recently inserted $O(M)$ elements → fast Insert
  [Small elements are inserted directly into Min]
- The external part stores exactly B elements in each block (all larger than the elements in MIN)

Operations

$K = \Theta(M)$

Insert(x)

- If x small ($\leq$ max MIN) then
  Swap x and max MIN
- Insert x into NEW
- If $|NEW| \geq K$ then
  Perform Batch Insert$_K$ with K elements from NEW

DeleteMin

- If MIN = $\emptyset$ then
  Perform Batch Delete$_K$
  Merge the result with NEW
  Move the K smallest elements to MIN
- Delete and Return min MIN

Batch Insert$_K$

- Create a new (ext. mem.) list of length K and assign it rank 1
- i := 1
- while #lists of rank i = $\frac{M}{B}$ do
  Merge the $\frac{M}{B}$ rank i lists and assign it rank i+1
  i := i+1

Batch Delete$_K$

- Delete the K least elements from the Lists in external memory

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

Lemma

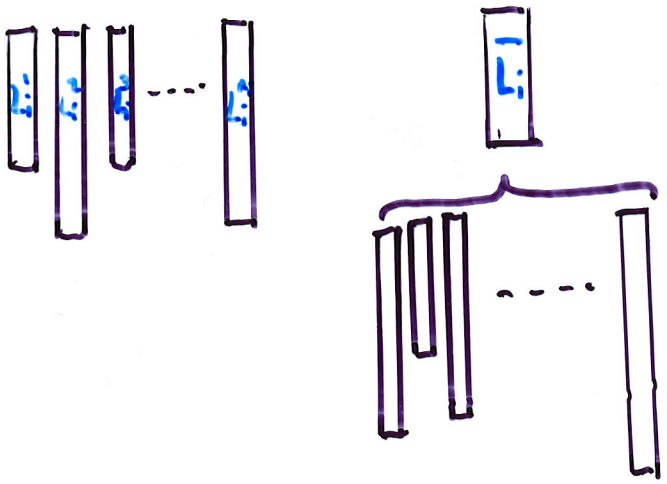
The described data structure supports Insert and DeleteMin with amortized $\Theta(\log n)$ comparisons and $\Theta(\frac{1}{B} \log_{M/B} \frac{N}{B})$ I/Os.

[the same bounds follow by using buffer trees].

Worst-case bounds?

- Incremental list merging; similar as done by Thorup '96
- Incremental batch operations
- Perform BatchDeleteMin$_K$ before MIN gets empty

Incremental list merging



Merge Step (i)

- Perform K steps of the list merging at rank i
- If the merging is finished, then make the resulting list $\overline{L_i}$ a list of rank i+1 (promote the list)

Batch Insert$_K$

- Create a new rank 1 list
- $\forall i$: Merge Step (i)

Deletions

BatchDeleteMin$_K$ deletes and returns the K least elements of the lists.

⇓ The length of the lists decrease

⇓ The maximum rank of a list 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 a rank $i+1$ list.

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 BatchDeleteMin

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

Conclusion

The First worst-case optimal external memory priority queue implementation.

Open problems

- Does "worst-case" make sense in practice?
- Applications!
- Implementation and experiments.
- Can buffer trees be deamortized, i.e., how can buffer trees support updates with worst-case $\Theta(\log_{M/B} \frac{N}{B})$ I/Os for B updates?