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

- CPU
- Internal Memory
- External Memory

\[ \text{Block transfer I/O} \]

\[ M = \text{Size of the internal memory} \]
\[ B = \text{IO block size} \]
\[ N = \# \text{elements in the priority queue} \]

Assumptions:
\[ M \geq 23B, \quad B \geq \log_2 \frac{N}{M} \]

Complexity measures:
- Internal: \# comparisons
- External: \# I/Os
Preliminaries

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

Proof: \( \Theta(M/B) \)-ary merging

Sorting: \( N \) elements in \( O(N/B) \) blocks can be sorted with \( \Theta(N/B \cdot \log M/B \cdot N/M) \) I/Os and \( \Theta(N \log N) \) comparisons.

Proof: \( \Theta(M/B) \)-ary mergesort

(i) Sort in internal memory all sequences of \( N \) elements.
(ii) In \( \log M/B \cdot N/M \) iterations apply the above merging algorithm to obtain lists of length \( M \cdot (M/B) \).

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

Aggarwal, Vitter '88
Worst-Case Efficiency

Internal: Each priority queue operation should require \( \Theta(\log_2 N) \) comparisons.

External: The I/Os should be divided evenly among the operations, i.e., one I/O for every \( \Theta(\frac{B}{\log_2 \log B} \frac{N}{B}) \) 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

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

- **Heap + B elements/node** Wegner, Tenhola '89

  \( O(\log_2 \left( \frac{N}{B} \right)) \) 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 $\Theta(B/\log MB \frac{N}{B})$ operations, which is the best possible.

Outline of the Data Structure

- **Min** stores the overall $O(M)$ smallest elements $\rightarrow$ fast DeleteMin
- **New** stores the most recently inserted $O(M)$ elements $\rightarrow$ fast Insert [Small elements are inserted directly into Min]
- The external part stores exactly 3 elements in each block (all larger than the elements in Min)

$\rightarrow$ Sorted Linked lists of length $\sim M \cdot (\frac{M}{B})$
Operations

\( k = \Theta(n) \)

**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 \( \text{MIN} = \emptyset \) then
  - Perform Batch Delete \( k \)
  - Merge the result with NEW
  - Move the \( k \) smallest elements to MIN
- Delete and Return \( \min \text{MIN} \)

**Batch Insert \( k \)**

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

**Batch Delete \( k \)**

- Delete the \( k \) least elements from the lists in external memory

\[
\text{rank } L = \log_{2^{\frac{M}{B} \cdot \log_{2^B} \frac{N}{M}}}
\]

\#lists \( \leq \frac{M}{B} \cdot \log_{2^B} \frac{N}{M} \)
Lemma
The described data structure supports Insert and DeleteMin with amortized $\Theta(\log n)$ comparisons and $\Theta(\frac{1}{B} \log M_1 / B \frac{N}{B})$ I/Os.

[the same bounds follow by using buffer trees]

Worst-case bounds?

- Incremental list merging; similar as done by Thorup 1996
- Incremental batch operations
- Perform BatchDeleteMin, before MIN gets empty
Incremental list merging

Lists of rank i:

Lists of rank generated after the inc. merging was initiated.

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 $L_i$ a list of rank $i+1$ (promote the list).

Batch Insert:

- Create a new rank 1 list
- $V_i$: Merge Step (i)
Deletions

BatchDeleteMin\(k\) deletes and returns the \(k\) least elements of the lists.

\(\downarrow\)
The length of the lists decrease

\(\downarrow\)
The maximum rank of a list increases

Possible solution: Incremental global rebuilding

Our solution:

1) If the incremental merging of rank \(i\) finishes, but \(|E_i| < k \left(\frac{M}{3}\right)^{R-1}\) then we do not promote \(E_i\) to a rank \(i+1\) list.

2) Let \(R\) be the maximum rank of a list. If the resulting list \(|E_R| < k \left(\frac{M}{3}\right)^{R-1}\) then \(E_R\) gets rank \(R-1\)

3) If \(k\) elements are deleted from \(E_i\), we perform Merge Step (i)

4) Always perform MergeStep\(R\) after BatchDeleteMin

\(R = O\left(\log_{\frac{1}{3}} \frac{N}{M}\right)\)
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 $O(\log m/B)$ I/Os for $B$ updates?