Towards Ultimate Binary Heaps

Jyrki Katajainen$^{1,2}$

Amr Elmasry$^3$

1 University of Copenhagen
2 Jyrki Katajainen and Company
3 Alexandria University
Binary heaps

\[ n = 8 \]

\[
\begin{array}{cccccccc}
0 & 2 & 3 & 4 & 6 & 7 & 51 \\
8 & 1 & 2 & 3 & 65 & 10 & 75 & 46 \\
75 & 46 & 12 & 75 & 80 \\
80 & \\
\end{array}
\]

left-child \( (i) \)

\[
\text{return } 2i + 1
\]
	right-child \( (i) \)

\[
\text{return } 2i + 2
\]

parent \( (i) \)

\[
\text{return } \lfloor (i - 1)/2 \rfloor
\]

construct \( () \)

\[
\text{for } (i = \text{parent}(n - 1); i \geq 0; --i) \\
\text{sift-down}(i)
\]

minimum \( () \)

\[
\text{return } a_0
\]

insert \( (x) \)

\[
a_n = x \\
\text{sift-up}(n) \\
n += 1
\]

extract-min \( () \)

\[
\text{min} = a_0 \\
n -= 1 \\
a_0 = a_n \\
\text{sift-down}(0) \\
\text{return min}
\]
Model of computation

Available

- An **infinite** array $a$ suitable for storing elements
- $O(1)$ other memory locations for storing elements
- $O(1)$ variables (counters, indices, bit strings of length $\lceil \lg(1+n) \rceil$)

![workspace diagram](image)

Requirement

- If the data structure stores $n$ elements, these elements must be kept in the first $n$ locations of $a$. 
2014: The 50th anniversary of binary heaps

[Williams 1964]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>$O(1)$</td>
<td>0</td>
</tr>
<tr>
<td>insert</td>
<td>$O(\log n)$</td>
<td>$\log n + O(1)$</td>
</tr>
<tr>
<td>extract-min</td>
<td>$O(\log n)$</td>
<td>$2\log n + O(1)$</td>
</tr>
</tbody>
</table>

[Gonnet & Munro 1986]

<table>
<thead>
<tr>
<th>Operation</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>$\log \log n \pm O(1)$ sufficient and necessary</td>
</tr>
<tr>
<td>extract-min</td>
<td>$\log n + \log^* n \pm O(1)$ sufficient and necessary</td>
</tr>
</tbody>
</table>

Assumptions

1. Single heap
2. Perfectly heap ordered before and after each operation
3. Operations are memoryless.
Multi-ary heaps

\[ n = 8 \]

\[ k = 4 \]

\[ \begin{array}{cccccccc}
26 & 75 & 80 & 5 & 6 & 7 & 12 & 8 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array} \]

\[ a = \begin{array}{cccccccc}
8 & 12 & 10 & 75 & 46 & 26 & 75 & 80 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array} \]

first-child\((i)\)

return \( k \times i + 1 \)

parent\((i)\)

return \( \lfloor (i - 1)/k \rfloor \)

[Johnson 1975]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>insert</td>
<td>( O(\log_k n) )</td>
</tr>
<tr>
<td>extract-min</td>
<td>( O(k \log_k n) )</td>
</tr>
</tbody>
</table>

\( n \): \# elements
\( k \): arity
Open—but solved later (see the last slide)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>$O(1)$</td>
<td>0</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extract-min</td>
<td>$O(\lg n)$</td>
<td>$\lg n + O(1)$</td>
</tr>
</tbody>
</table>

in-place as binary heaps

**Sorting**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>$O(1)$</td>
<td>0</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extract-min</td>
<td>$\Omega(\lg n)$</td>
<td>$\lg n - O(1)$ necessary</td>
</tr>
</tbody>
</table>

$n$: # elements
Related work

[Carlsson, Munro & Poblete 1988]

queue of pennants
\( \text{extract-min}: 3 \lg n + \log^* n \) element comparisons

[Elmasry, Jensen & Katajainen 2008]

multipartite priority queue
\( O(n) \) extra words

[Edelkamp, Elmasry & Katajainen 2012]

engineered weak heaps
\( n + O(w) \) extra bits

\( n \): # elements
\( w \): word size in bits
**Our main result**

The objective can be achieved in the amortized sense

\[ \ell \leq \log^2(n_0)/4 \]
\[ r \leq n_0 \]
\[ n = n_0 + \ell + r \]

**What is special?**

- Partial heap
- Buffer
- Backlog
- Sizes:
  - \( n_0 \)
  - \( \ell \)
  - \( r \)

- Heap-order violations
- Some large elements kept outside
Insert

\[ \ell \leq \log^2(n_0)/4 \]
\[ r \leq n_0 \]
\[ n = n_0 + \ell + r \]

components:
- partial heap
- buffer
- backlog

elements:
- \(a_0a_1\ldots a_n\)

sizes:
- \(n_0\)
- \(\ell\)
- \(r\)

\(\text{insert}(x)\)

\(a_n \leftarrow a_{n_0} + \ell\)
\(a_{n_0 + \ell} \leftarrow x\)
\(\text{if } (\ell = 0)\)
\(k \leftarrow \lceil \log(n_0)/2 \rceil\)
\(k\text{-ary-heap-insert}(a, n_0, \ell, a_{n_0 + \ell})\)
\(\ell++\)
\(n++\)
\(\text{if } (\ell = k^2)\)
\(\text{bulk-insert}(a, n, n_0, \ell)\)
Bulk-insert

Use Floyd's heap-construction algorithm in the red area: call \textit{sift-down} at each node

I part: $O(\ell)$ nodes; $O(\ell)$ total work as for Floyd's algorithm

II part: $O(\lg(n_0))$ \textit{sift-down} calls
⇒ $O(\lg^2(n_0))$ total work

∴ amortized $O(1)$ work per \textit{insert}
Extract-min

Minimum in the buffer

\(k\text{-ary-heap-extract-min}(a, n_0, \ell)\)
\(a_{n_0 + \ell} \leftarrow a_n\)
\(\ell --\)
\(n --\)

Minimum in the heap

extract-root\((a, n, n_0, \ell)\)
\(n --\)

\(\ell \leq \lg^2(n_0)/4\)
\(r \leq n_0\)
\(n = n_0 + \ell + r\)

Extract-min (a, n, n₀, ℓ)

if (\(n₀ + ℓ = n\))
  rebuild-structure\((a, n, n₀, ℓ)\)

\(i \leftarrow 0\)

while (\(i \leq \text{parent}(n₀ - 1)\))
  \(a_\text{left-child}(i) < a_\text{right-child}(i)\)
  \(a_i \leftarrow a_\text{left-child}(i)\)
  \(i \leftarrow \text{left-child}(i)\)
else
  \(a_i \leftarrow a_\text{right-child}(i)\)
  \(i \leftarrow \text{right-child}(i)\)
\(a_i \leftarrow a_{n-1}\)
Deamortization of insertions

The objective can **almost** be achieved in the worst case.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extract-min</td>
<td>$O(\log n)$</td>
<td>$\log n + \log^* n + O(1)$</td>
</tr>
</tbody>
</table>

Role of the buffers
- buffer$_2$ takes care of the inserted elements
- buffer$_1$ is incrementally sub-merged into the heap by performing a constant amount of work per insert
# Online sorting

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case time</th>
<th># element comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>construct</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>extract-min</td>
<td>$O(\lg n)$</td>
<td>$\lg n + O(1)$</td>
</tr>
</tbody>
</table>

→ Read the details from the paper

[Elmasry 2003]

$O(n)$ extra bits

$n$: # elements
Note added on 15 April, 2013

Stefan Edelkamp came with the missing link that helped us to solve the problem. We are in the process of writing the paper “Optimal in-place heaps”. The new construction shows that the both lower bounds proved by Gonnet & Munro can be bypassed.