Priority queue data type

A min-oriented priority queue supports the following core operations:

- **MAKE-HEAP():** create an empty heap.
- **INSERT(\(H, x\)):** insert an element \(x\) into the heap.
- **EXTRACT-MIN(\(H\)):** remove and return an element with the smallest key.
- **DECREASE-KEY(\(H, x, k\)):** decrease the key of element \(x\) to \(k\).

The following operations are also useful:

- **IS-EMPTY(\(H\)):** is the heap empty?
- **FIND-MIN(\(H\)):** return an element with smallest key.
- **DELETE(\(H, x\)):** delete element \(x\) from the heap.
- **UNION(\(H_1, H_2\)):** replace heaps \(H_1\) and \(H_2\) with their union.

**Note.** Each element contains a key (duplicate keys are permitted) from a totally-ordered universe.

Priority queue applications

**Applications.**

- A* search.
- Heapsort.
- Online median.
- Huffman encoding.
- Prim’s MST algorithm.
- Discrete event-driven simulation.
- Network bandwidth management.
- Dijkstra’s shortest-paths algorithm.
- ...
Complete binary tree

Binary tree. Empty or node with links to two disjoint binary trees.

Complete tree. Perfectly balanced, except for bottom level.

Height of complete binary tree with $n$ nodes is $\lceil \log_2 n \rceil$.

Pf. Height increases (by 1) only when $n$ is a power of 2.

Binary heap

Binary heap. Heap-ordered complete binary tree.

Heap-ordered. For each child, the key in child $\leq$ key in parent.

Explicit binary heap

Pointer representation. Each node has a pointer to parent and two children.
- Maintain number of elements $n$.
- Maintain pointer to root node.
- Can find pointer to last node or next node in $O(\log n)$ time.
Implicit binary heap

**Array representation.** Indices start at 1.
- Take nodes in level order.
- Parent of node at $k$ is at $\lfloor k / 2 \rfloor$.
- Children of node at $k$ are at $2k$ and $2k + 1$.

---

**Binary heap demo**

**Heap ordered**

---

**Binary heap: insert**

**Insert.** Add element in new node at end; repeatedly exchange new element with element in its parent until heap order is restored.

---

**Binary heap: extract the minimum**

**Extract min.** Exchange element in root node with last node; repeatedly exchange element in root with its smaller child until heap order is restored.
**Binary heap: decrease key**

**Decrease key.** Given a handle to node, repeatedly exchange element with its parent until heap order is restored.

**decrease key of node x to 11**

**Binary heap: analysis**

**Theorem.** In an implicit binary heap, any sequence of $m$ \textsc{insert}, \textsc{extract-min}, and \textsc{decrease-key} operations with $n$ \textsc{insert} operations takes $O(m \log n)$ time.

**Pf.**
- Each heap op touches nodes only on a path from the root to a leaf; the height of the tree is at most $\log_2 n$.
- The total cost of expanding and contracting the arrays is $O(n)$.

**Theorem.** In an explicit binary heap with $n$ nodes, the operations \textsc{insert}, \textsc{decrease-key}, and \textsc{extract-min} take $O(\log n)$ time in the worst case.

**Binary heap: find-min**

**Find the minimum.** Return element in the root node.

**Binary heap: delete**

**Delete.** Given a handle to a node, exchange element in node with last node; either swim down or sink up the node until heap order is restored.

**delete node x or y**
**Binary heap: union**

**Union.** Given two binary heaps $H_1$ and $H_2$, merge into a single binary heap.

**Observation.** No easy solution: $\Omega(n)$ time apparently required.

[Diagram of two binary heaps $H_1$ and $H_2$ merged into a single binary heap]

---

**Binary heap: heapify**

**Theorem.** Given $n$ elements, can construct a binary heap containing those $n$ elements in $O(n)$ time.

**Pf.**
- There are at most $\lceil n/2^{h+1} \rceil$ nodes of height $h$.
- The amount of work to sink a node is proportional to its height $h$.
- Thus, the total work is bounded by:
  \[
  \sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil h \leq \sum_{h=0}^{\lfloor \log_2 n \rfloor} nh/2^h \leq 2n
  \]

**Corollary.** Given two binary heaps $H_1$ and $H_2$ containing $n$ elements in total, can implement UNION in $O(n)$ time.

---

**Priority queues performance cost summary**

<table>
<thead>
<tr>
<th>operation</th>
<th>linked list</th>
<th>binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>IS-EMPTY</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>FIND-MIN</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Priority queues performance cost summary

Q. Reanalyze so that \textsc{Extract-Min} and \textsc{Delete} take \(O(1)\) amortized time?

<table>
<thead>
<tr>
<th>operation</th>
<th>linked list</th>
<th>binary heap</th>
<th>binary heap †</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{Make-Heap}</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>\textsc{IsEmpty}</td>
<td>(O(1))</td>
<td>(O(1))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>\textsc{Insert}</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>\textsc{Extract-Min}</td>
<td>(O(n))</td>
<td>(O(\log n))</td>
<td>(O(1)) †</td>
</tr>
<tr>
<td>\textsc{Decrease-Key}</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>\textsc{Delete}</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(O(1)) †</td>
</tr>
<tr>
<td>\textsc{Union}</td>
<td>(O(1))</td>
<td>(O(n))</td>
<td>(O(n))</td>
</tr>
<tr>
<td>\textsc{Find-Min}</td>
<td>(O(n))</td>
<td>(O(1))</td>
<td>(O(1))</td>
</tr>
</tbody>
</table>

† amortized

Complete \(d\)-ary tree

Binary tree. Empty or node with links to \(d\) disjoint \(d\)-ary trees.

Complete tree. Perfectly balanced, except for bottom level.

Fact. The height of a complete \(d\)-ary tree with \(n\) nodes is \(\leq \lceil \log_d n \rceil\).

Multiway heap: insert

Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

Running time. Proportional to height = \(O(\log_d n)\).
Multiway heap: extract the minimum

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in largest child until heap order is restored.

Running time. Proportional to \( d \times \text{height} = O(d \log_d n) \).

---

Multiway heap: decrease key

Decrease key. Given a handle to an element \( x \), repeatedly exchange it with its parent until heap order is restored.

Running time. Proportional to height = \( O(\log_d n) \).

---

Priority queues performance cost summary

<table>
<thead>
<tr>
<th>operation</th>
<th>linked list</th>
<th>binary heap</th>
<th>d-ary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>IS-EMPTY</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>INSERT</td>
<td>( O(1) )</td>
<td>( O(\log n) )</td>
<td>( O(\log_d n) )</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>( O(n) )</td>
<td>( O(\log n) )</td>
<td>( O(d \log_d n) )</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>( O(1) )</td>
<td>( O(\log n) )</td>
<td>( O(\log_d n) )</td>
</tr>
<tr>
<td>DELETE</td>
<td>( O(1) )</td>
<td>( O(\log n) )</td>
<td>( O(d \log_d n) )</td>
</tr>
<tr>
<td>UNION</td>
<td>( O(1) )</td>
<td>( O(\log n) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>FIND-MIN</td>
<td>( O(n) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>
Priority queues performance cost summary

<table>
<thead>
<tr>
<th>operation</th>
<th>linked list</th>
<th>binary heap</th>
<th>d-ary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>ISEMPTY</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log_d n)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log_d n)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log_d n)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log_d n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>FIND-MIN</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

**Goal.** $O(\log n)$ INSERT, DECREASE-KEY, EXTRACT-MIN, and UNION.

---

Binomial tree properties

**Def.** A binomial tree of order $k$ is defined recursively:
- Order 0: single node.
- Order $k$: one binomial tree of order $k - 1$ linked to another of order $k - 1$.

**Properties.** Given an order $k$ binomial tree $B_k$,
- Its height is $k$.
- It has $2^k$ nodes.
- It has $\binom{k}{i}$ nodes at depth $i$.
- The degree of its root is $k$.
- Deleting its root yields $k$ binomial trees $B_{k-1}, \ldots, B_0$.

**Pf.** [by induction on $k$]
**Binomial heap**

**Def.** A binomial heap is a sequence of binomial trees such that:
- Each tree is min-heap ordered.
- There is either 0 or 1 binomial tree of order $k$.

**Binomial heap representation**

**Binomial trees.** Represent trees using left-child, right-sibling pointers.

**Roots of trees.** Connect with singly-linked list, with degrees decreasing from left to right.

**Binomial heap properties**

**Properties.** Given a binomial heap with $n$ nodes:
- The node containing the min element is a root of $B_0$, $B_1$, ..., or $B_k$.
- It contains the binomial tree $B_i$ iff $b_i = 1$, where $b_i \cdot b_{i-1} b_{i-2} \cdots b_0$ is binary representation of $n$.
- It has $\leq \lceil \log_2 n \rceil + 1$ binomial trees.
- Its height $\leq \lceil \log_2 n \rceil$.

**Binomial heap: union**

**Union operation.** Given two binomial heaps $H_1$ and $H_2$, (destructively) replace with a binomial heap $H$ that is the union of the two.

**Warmup.** Easy if $H_1$ and $H_2$ are both binomial trees of order $k$.
- Connect roots of $H_1$ and $H_2$.
- Choose node with smaller key to be root of $H$.
**Binomial heap: union**

**Union operation.** Given two binomial heaps $H_1$ and $H_2$, (destructively) replace with a binomial heap $H$ that is the union of the two.

**Solution.** Analogous to binary addition.

**Running time.** $O(\log n)$.

**Pf.** Proportional to number of trees in root lists $\leq 2 \left( \left\lfloor \log_2 n \right\rfloor + 1 \right)$.  □
Binomial heap: extract the minimum

**Extract-min.** Delete the node with minimum key in binomial heap $H$.
- Find root $x$ with min key in root list of $H$, and delete.

![Diagram of Extract-min](image)

Binomial heap: decrease key

**Decrease key.** Given a handle to an element $x$ in $H$, decrease its key to $k$.
- Suppose $x$ is in binomial tree $B_k$.
- Repeatedly exchange $x$ with its parent until heap order is restored.

**Running time.** $O(\log n)$.

![Diagram of Decrease-key](image)

Binomial heap: delete

**Delete.** Given a handle to an element $x$ in a binomial heap, delete it.
- **DECREASE-KEY**($H$, $x$, $-\infty$).
- **DELETE-MIN**($H$).

**Running time.** $O(\log n)$.

![Diagram of Delete](image)
Binomial heap: insert

**Insert.** Given a binomial heap $H$, insert an element $x$.

- $H' \leftarrow $ MAKE-HEAP($x$).
- $H' \leftarrow $ INSERT($H'$, $x$).
- $H \leftarrow $ UNION($H'$, $H$).

**Running time.** $O(\log n)$.

---

Binomial heap: sequence of insertions

**Insert.** How much work to insert a new node $x$?

- If $n = \ldots .0$, then only 1 credit.
- If $n = \ldots .01$, then only 2 credits.
- If $n = \ldots .011$, then only 3 credits.
- If $n = \ldots .0111$, then only 4 credits.

**Observation.** Inserting one element can take $\Omega(\log n)$ time.

**Theorem.** Starting from an empty binomial heap, a sequence of $n$ consecutive INSERT operations takes $O(n)$ time.

**Pf.** $(n / 2)(1) + (n / 4)(2) + (n / 8)(3) + \ldots \leq 2 n$. \[ \sum_{i=1}^{k} \frac{i}{2^i} = 2 - \frac{k}{2^k} - \frac{1}{2^{k-1}} \leq 2 \]

---

Binomial heap: amortized analysis

**Theorem.** In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE KEY is $O(\log n)$.

**Pf.** Define potential function $\Phi(H_i) = \text{trees}(H_i) = \#$ trees in binomial heap $H_i$.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap $H_i$.

**Case 1.** [INSERT]

- Actual cost $c_i = \text{number of trees merged} + 1$.
- $\Delta \Phi = \Phi(H_i) - \Phi(H_{i-1}) = \text{number of trees merged} - 1$.
- Amortized cost $= \hat{c}_i = c_i + \Phi(H_i) - \Phi(H_{i-1}) = 2$.

---

Binomial heap: amortized analysis

**Theorem.** In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE KEY is $O(\log n)$.

**Pf.** Define potential function $\Phi(H_i) = \text{trees}(H_i) = \#$ trees in binomial heap $H_i$.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap $H_i$.

**Case 2.** [DECREASE-KEY]

- Actual cost $c_i = O(\log n)$.
- $\Delta \Phi = \Phi(H_i) - \Phi(H_{i-1}) = 0$.
- Amortized cost $= \hat{c}_i = c_i = O(\log n)$.

---
Binomial heap: amortized analysis

**Theorem.** In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE-KEY is $O(\log n)$.

**Pf.** Define potential function $\Phi(H_i) = \text{trees}(H_i) = \# \text{ trees in binomial heap } H_i$.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap $H_i$.

**Case 3.** [EXTRACT-MIN or DELETE]

- Actual cost $c_i = O(\log n)$.
- $\Delta \Phi = \Phi(H_i) - \Phi(H_{i+1}) \leq \log n$.
- Amortized cost $\hat{c}_i = c_i + \Phi(H_i) - \Phi(H_{i+1}) = O(\log n)$.

**Priority queues performance cost summary**

<table>
<thead>
<tr>
<th>operation</th>
<th>linked list</th>
<th>binary heap</th>
<th>binomial heap</th>
<th>binomial heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>ISEMPTY</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>FIND-MIN</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Hopeless challenge. $O(1)$ INSERT, DECREASE-KEY and EXTRACT-MIN. Why?

Challenge. $O(1)$ INSERT and DECREASE-KEY, $O(\log n)$ EXTRACT-MIN.