Fibonacci Heaps

6.046 Design and Analysis of Algorithms

Charles E. Leiserson

MIT EECS

Spring 2011
These lecture slides were adapted with permission from materials originally developed by Kevin Wayne of Princeton University in Spring 2007 for COS 423 Theory of Algorithms, which in turn were based on the following textbook materials:

- Chapter 20 of Introduction to Algorithms (second edition) by Cormen, Leiserson, Rivest, and Stein;
- Chapter 9 of The Design and Analysis of Algorithms by Dexter Kozen.
Theorem. Let $T$ be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting $A$ to $V - A$. Then $(u, v) \in T$ (assuming all edge weights are distinct).
Prim’s Algorithm for Minimum Spanning Trees

Idea: Maintain $V - A$ as a priority queue $Q$. Key each vertex in $Q$ with the weight of the least-weight edge connecting it to a vertex in $A$.

$v.key = \infty$ for all $v \in V$
$s.key = 0$ for some arbitrary $s \in V$
$Q \leftarrow V$ // $|V|$ INSERT’s into $Q$
while $Q \neq \emptyset$
    $u = $ EXTRACT-MIN($Q$)
    for each $v \in Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.key = w(u, v)$ // DECREASE-KEY
            $v.\pi = u$

At the end, $\{(v, v.\pi)\}$ forms the MST.
Example of Prim’s algorithm

- $\epsilon A$
- $\epsilon V - A$
Example of Prim’s algorithm

\[ \in A \]
\[ \notin V \setminus A \]
Example of Prim’s algorithm
Example of Prim’s algorithm

\[ \epsilon A \]
\[ \in V - A \]
Example of Prim’s algorithm

\[
\begin{align*}
\varepsilon & \in A \\
\notin & \in V - A
\end{align*}
\]
Example of Prim’s algorithm
Example of Prim’s algorithm
Example of Prim’s algorithm

\[ \in A \]

\[ \in V - A \]
Example of Prim’s algorithm

∈ A
∈ V – A

Diagram showing a graph with nodes and edges labeled with weights.
Example of Prim’s algorithm

\[ \in A \]
\[ \in V - A \]
Example of Prim’s algorithm
Example of Prim’s algorithm

∈ A
∈ V – A
Example of Prim’s algorithm

∈ $A$

∈ $V - A$
Analysis of Prim

\begin{align*}

\Theta(V) & \left\{ \begin{array}{l}
 v.key &= \infty \text{ for all } v \in V \\
s.key &= 0 \text{ for some arbitrary } s \in V
\end{array} \right. \\
Q & \leftarrow V \quad // |V| \text{ INSERT's into } Q \\
\text{while } Q & \neq \emptyset \\
& \quad u = \text{EXTRACT-MIN}(Q) \\
& \quad \text{for each } v \in \text{Adj}[u] \\
& \quad \quad \text{if } v \in Q \text{ and } w(u, v) < v.key \\
& \quad \quad \quad v.key = w(u, v) \quad // \text{DECREASE-KEY} \\
& \quad \quad v.\pi = u
\end{align*}

\text{Handshaking Lemma } \Rightarrow \Theta(E) \text{ implicit } \text{DECREASE-KEY's.}

\text{Total time} \\
\Theta(V) \cdot T_{\text{INSERT}} + \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}
Priority Queues

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary Heap</th>
<th>Fibonacci Heap*</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>O(1)</td>
<td>O(lg (n))</td>
<td>O(1)</td>
</tr>
<tr>
<td>EXTRACT–MIN</td>
<td>O((n))</td>
<td>O(lg (n))</td>
<td>O(lg (n))</td>
</tr>
<tr>
<td>DECREASE–KEY</td>
<td>O((n))</td>
<td>O(lg (n))</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

\(n = \# \text{ elements in queue} \quad \quad \quad \text{*amortized}

Hopeless challenge: O(1)–time for all of INSERT, EXTRACT–MIN, and DECREASE–KEY. Why?
# Priority Queues and Prim

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary Heap</th>
<th>Fibonacci Heap*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSERT</strong></td>
<td>O(1)</td>
<td>O(lg (n))</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>EXTRACT–MIN</strong></td>
<td>O((n))</td>
<td>O(lg (n))</td>
<td>O(lg (n))</td>
</tr>
<tr>
<td><strong>DECREASE–KEY</strong></td>
<td>O((n))</td>
<td>O(lg (n))</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

\(n = \# \text{ elements in queue}\)  \(^*\)amortized

Time for Prim’s algorithm:

\[ \Theta(V) \cdot T_{\text{INSERT}} + \Theta(V) \cdot T_{\text{EXTRACT–MIN}} + \Theta(E) \cdot T_{\text{DECREASE–KEY}} \]

Linked list: \(O(V^2)\) worst case

Binary heap: \(O(E \lg V)\) worst case

Fibonacci heap: \(O(E + V \lg V)\) worst case

Fibonacci heaps were invented by Fredman and Tarjan in 1986.
Fibonacci–Heap Data Structure

Fibonacci heap
- Set of heap–ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

Heap $H$

Roots

Heap-ordered tree
Fibonacci-Heap Data Structure

Fibonacci heap
- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

Finding the minimum takes O(1) time
Fibonacci–Heap Data Structure

Fibonacci heap
- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

used to keep heaps bushy

Heap $H$

$min$

marked
Notation

- $n$ = number of nodes in heap.
- $\deg(x)$ = number of children of node $x$.
- $D(n)$ = max degree of any node in any heap with $n$ elements.
- $\text{trees}(H)$ = number of trees in heap $H$.
- $\text{marks}(H)$ = number of marked nodes in heap $H$.

$trees(H) = 5 \quad marks(H) = 3 \quad n = 14 \quad \deg = 3$
Potential Function

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

potential of heap \( H \)

\[ \text{trees}(H) = 5 \quad \text{marks}(H) = 3 \quad \Phi(H) = 5 + 2 \cdot 3 = 11 \]
**INSERT**

- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

**INSERT(21)**

```
| 17 | 24 | 23 | 7 |

17: 30 26 35
24: 46
23: 7
7: 3

min: 3
```

Heap H
**INSERT**

- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

**INSERT(21)**

*Heap $H'$*
Analysis of INSERT

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

potential of heap \( H \)

\[ \hat{c} = c + \Phi(H') - \Phi(H) \]

amortized cost

Actual cost: \( c = O(1) \)

Change in potential: \( \Phi(H') - \Phi(H) = +1 \)

Amortized cost: \( \hat{c} = O(1) \)
EXTRACT–MIN
Linking

**Linking operation** on two trees whose roots have the same degree. Make the root with the larger key be a child of the root with the smaller key.

![Diagram of trees](image)

**Tree** $T_1$, **tree** $T_2$, **tree** $T'$
**EXTRACT-MIN: Step 1**

**EXTRACT-MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.
**EXTRACT−MIN: Step 1**

**EXTRACT−MIN**

- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram showing the process of EXTRACT−MIN](image)
**EXTRACT–MIN: Step 2**

**EXTRACT–MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**

- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram of a tree structure with nodes labeled 7, 24, 23, 17, 18, 52, 41, 30, 26, 46, 35, 39, 44. The degree distribution is shown in a table with 0, 1, 2, and 3 columns. The current node is marked with a red arrow. The min node is indicated.]
**EXTRACT−MIN: Step 2**

**EXTRACT−MIN**

- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram](attachment:image.png)
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**
- Remove the `min` element; meld its children into the root list; update the `min` pointer.
- Consolidate trees so that no two roots have the same degree.
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**
- Remove the \textit{min} element; meld its children into the root list; update the \textit{min} pointer.
- Consolidate trees so that no two roots have the same degree.

**Diagram:**
- The diagram shows a tree structure with nodes labeled from 7 to 52.
- The node labeled 23 is linked into the node labeled 17.
- The \textit{min} element is indicated by a small black dot.
- The \textit{current} element is indicated by a red arrow.

**Legend:**
- \textit{degree}:
  - 0: 1
  - 1: 2
  - 2: 3
  - 3: 4

**Note:**
- The diagram illustrates the process of link 23 into 17.
**EXTRACT-MIN**: Step 2

**EXTRACT-MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

---

Link 17 into 7

Diagram showing a tree with nodes 7, 24, 17, 18, 52, 41, 30, 26, 46, 35, 23, 39, and 44. The tree is being consolidated by removing the min element and melding its children into the root list. The current node is highlighted with an arrow.
**EXTRACT-MIN**: Step 2

**EXTRACT-MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram of a binary heap with nodes labeled 24, 7, 18, 52, 41, 26, 46, 17, 30, 39, 44, 35, 23. The *min* element is 24, and the current node is 7. An arrow labeled "degree" points to a table with columns 0, 1, 2, 3. The diagram shows how 24 is linked into the root list at node 7.]
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**
- Remove the \textit{min} element; meld its children into the root list; update the \textit{min} pointer.
- Consolidate trees so that no two roots have the same degree.
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**

- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

```
   current
   
   7
   
   26
   24 17 30
   
   35

   degree
   0 1 2 3
   • • • •
```

Consolidate trees so that no two roots have the same degree.
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.
**Extract–Min: Step 2**

**Extract–Min**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

Link 41 into 18

```plaintext
min  

degree  

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

current

7  

24 26

17 23

30 46

39

18 52

41

44
```
**EXTRACT−MIN: Step 2**

**EXTRACT−MIN**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram showing a binary tree with nodes labeled 7, 24, 17, 30, 26, 46, 23, 35, 52, 41, 39, 18, 44. The *min* element is 7, and the root list is updated accordingly. The degree of each node is shown in a table.](image-url)
**EXTRACT-MIN: Step 2**

**EXTRACT-MIN**

- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.

![Diagram](image)
**Extract-Min: Done**

**Extract-Min**
- Remove the *min* element; meld its children into the root list; update the *min* pointer.
- Consolidate trees so that no two roots have the same degree.
## Analysis of EXTRACT-MIN

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

*potential of heap H*

\[ \hat{c} = c + \Phi(H') - \Phi(H) \]

*amortized cost*

### Actual cost: \( O(D(n)) + O(\text{trees}(H)) \)
- \( O(D(n)) \) to meld *min's children into root list.*
- \( O(D(n)) + O(\text{trees}(H)) \) to update *min.*
- \( O(D(n)) + O(\text{trees}(H)) \) to consolidate trees.

### Change in potential: \( O(D(n)) - \text{trees}(H) \)
- \( \text{trees}(H') \leq D(n) + 1 \), since no two trees have same degree.
- \( \Phi(H') - \Phi(H) \leq D(n) + 1 - \text{trees}(H) \).

### Amortized cost: \( O(D(n)) \)
DECREASE–KEY
Strategy for **DECREASE-KEY**

Deceasing the key of node \( x \)
- If heap order is not violated, just decrease \( x.key \).
- Otherwise, cut \( x \) off from its parent \( x.p \) and meld \( x \)'s tree into the root list.
  - If \( x.p \) is not marked, mark it.
  - If \( x.p \) is marked, cut \( x.p \) recursively, meld into the root list, and unmark.
**DECREASE-KEY: Case 1a**

Case 1. [heap order not violated]
- Decrease $x.key$.
- Change heap $min$ pointer (if necessary).

Decrease-Key($x$, 29) from 46.
**DECREASE–KEY: Case 1b**

Case 1. [heap order not violated]
- Decrease $x$.key.
- Change heap $min$ pointer (if necessary).

Decrease–Key($x$, 29) from 46.
Decrease-Key: Case 2a, Step 1

Case 2a. [heap order violated]
- Decrease x.key.
- Cut x (from x.p), meld x into root list, and unmark.
- If x.p is unmarked (hasn't yet lost a child), mark it.
  Otherwise, cut x.p, meld x.p into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Decrease-Key(x, 15) from 29.
**DECREASE-KEY: Case 2a, Step 2**

Case 2a. [heap order violated]
- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it.
  Otherwise, cut $x.p$, meld $x.p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Decrease-Key($x$, 15) from 29.
**DECREASE-KEY: Case 2a, Step 2**

Case 2a. [heap order violated]
- Decrease \(x.key\).
- Cut \(x\) (from \(x.p\)), meld \(x\) into root list, and unmark.
- If \(x.p\) is unmarked (hasn't yet lost a child), mark it.
  Otherwise, cut \(x.p\), meld \(x.p\) into root list, and unmark (and do so recursively for all ancestors that lose a second child).

\[x\]

![Diagram](image-url)

**Decrease-Key(x, 15) from 29.**
**DECREASE–KEY: Case 2a, Step 3**

Case 2a. [heap order violated]
- Decrease \(x\).key.
- Cut \(x\) (from \(x.p\)), meld \(x\) into root list, and unmark.
- If \(x.p\) is unmarked (hasn't yet lost a child), mark it.
  Otherwise, cut \(x.p\), meld \(x.p\) into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Decrease–Key(\(x\), 15) from 29.
**DECREASE–KEY: Case 2b, Step 1**

Case 2b. [heap order violated]
- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it. Otherwise, cut $x.p$, meld $x.p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Decrease–Key($x$, 5) from 35.
**DECREASE–KEY: Case 2b, Step 2**

Case 2b. [heap order violated]
- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it. Otherwise, cut $p$, meld $p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

![Diagram of a heap with nodes labeled with values and edges showing parent-child relationships.](image)

**Decrease–Key($x$, 5) from 35.**
**DECREASE–KEY: Case 2b, Step 2**

Case 2b. [heap order violated]

- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it. Otherwise, cut $x.p$, meld $x.p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

\[
\text{Decrease–Key}(x, 5) \text{ from } 35.
\]
DECREASE-KEY: Case 2b, Step 3

Case 2b. [heap order violated]
- Decrease x.key.
- Cut x (from x.p), meld x into root list, and unmark.
- If x.p is unmarked (hasn't yet lost a child), mark it. Otherwise, cut x.p, meld x.p into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Decrease-Key(x, 5) from 35.
DECREASE–KEY: Case 2b, Step 3

Case 2b. [heap order violated]

- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it.

Otherwise, cut $p$, meld $p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

\[
\text{min} \quad x \quad x.p
\]

Decrease–Key($x$, 5) from 35.
**DECREASE–KEY: Case 2b, Step 3**

Case 2b. [heap order violated]
- Decrease $x.key$.
- Cut $x$ (from $x.p$), meld $x$ into root list, and unmark.
- If $x.p$ is unmarked (hasn't yet lost a child), mark it. Otherwise, cut $p$, meld $p$ into root list, and unmark (and do so recursively for all ancestors that lose a second child).

\[
\text{Decrease–Key}(x, 5) \text{ from 35.}
\]
**DECREASE–KEY: Case 2b, Step 3**

**Case 2b.** [heap order violated]

- Decrease \(x\).key.
- Cut \(x\) (from \(x.p\)), meld \(x\) into root list, and unmark.
- If \(x.p\) is unmarked (hasn't yet lost a child), mark it. Otherwise, cut \(p\), meld \(p\) into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Don't mark parent if it is a root.

Decrease–Key\((x, 5)\) from 35.
Analysis of **DECREASE-KEY**

\[
\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)
\]

*potential of heap $H$*

\[
\hat{c} = c + \Phi(H') - \Phi(H)
\]

*amortized cost*

**Actual cost:** $O(s)$, where $s$ is the number of cuts.
- $O(1)$ time for changing the key.
- $O(1)$ time for each of $s$ cuts, plus melding into root list.

**Change in potential:** $O(1) - s$
- $\text{trees}(H') = \text{trees}(H) + s$.
- $\text{marks}(H') \leq \text{marks}(H) - s + 2$.
- $\Phi(H') - \Phi(H) \leq s + 2(-s + 2) = 4 - s$.

**Amortized cost:** $O(1)$
Analysis
Analysis Summary

**INSERT:** \( O(1) \)

**EXTRACT-MIN:** \( O(D(n)) \) amortized

**DECREASE-KEY:** \( O(1) \) amortized

Key lemma: \( D(n) = O(\lg n) \).
Degree Lemma

Lemma. Fix a point in time. Let \( x \) be a node, and let \( y_1, y_2, \ldots, y_k \) denote its children in the order in which they were linked to \( x \). Then we have

\[
\text{degree}(y_i) \geq \begin{cases} 
0 & \text{if } i = 1, \\
i - 2 & \text{if } i \geq 2.
\end{cases}
\]

Proof.

- When \( y_i \) was linked into \( x \), \( x \) had at least \( i - 1 \) children \( y_1, \ldots, y_{i-1} \).
- Since only trees of equal degree are linked, at that time \( \text{degree}(y_i) = \text{degree}(x) \geq i - 1 \).
- Since then, \( y_i \) has lost at most one child.
- Thus, right now \( \text{degree}(y_i) \geq i - 2 \), or \( y_i \) would have been cut. \( \blacksquare \)
Degree Lemma

**Lemma.** Fix a point in time. Let $x$ be a node, and let $y_1, y_2, ..., y_k$ denote its children in the order in which they were linked to $x$. Then we have

$$\text{degree}(y_i) \geq \begin{cases} 0 & \text{if } i = 1, \\ i - 2 & \text{if } i \geq 2. \end{cases}$$

Let $T_k$ be the smallest tree of degree $k$ satisfying this property.
The *Fibonacci numbers* are the sequence \( \langle 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots \rangle \), where each number is the sum of the previous two.

**Recurrence:**

\[
F_0 = 0, \\
F_1 = 1, \\
F_n = F_{n-1} + F_{n-2} \text{ for } n > 1.
\]

The sequence is named after Leonardo di Pisa (1170–1250 A.D.), also known as Fibonacci, a contraction of *filius Bonaccii* —“son of Bonaccio.” Fibonacci’s 1202 book *Liber Abaci* introduced the sequence to Western mathematics, although it had previously been discovered by Indian mathematicians.
Trees and Fibonacci

Lemma. Let $T_k$ be the smallest tree of degree $k$ such that for all nodes $x \in T$ with children $y_1, y_2, \ldots, y_k$, we have

$$
\text{degree}(y_i) \geq \begin{cases} 
0 & \text{if } i = 1, \\
 i - 2 & \text{if } i \geq 2.
\end{cases}
$$

Then $|T_k| = F_{k+2}$.

Proof. By induction on $k$. 
Bounding the Degree

Fibonacci fact: \( F_{k+2} \geq \phi^k \), where \( \phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \) is the golden ratio.

Corollary. \( D(n) \leq \log_\phi n = O(\lg n) \). □
## Priority Queues: Asymptotic Performance

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary Heap</th>
<th>Binomial Heap</th>
<th>Fibonacci Heap*</th>
<th>Relaxed Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>IS-EMPTY</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>1</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
<td>lg ( n )</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>1</td>
<td>( n )</td>
<td>lg ( n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>FIND-MIN</strong></td>
<td>( n )</td>
<td>1</td>
<td>lg ( n )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( n = \# \text{ elements in priority queue} \)

*amortized