<table>
<thead>
<tr>
<th></th>
<th>Binary Heap (worst case)</th>
<th>Binomial Heap (worst case)</th>
<th>Fibonacci Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make-Heap</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Minimum</td>
<td>$\Theta(1)$</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Extract-Min (Union)</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(lgn)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

**Mergeable Heaps**

**Union($H_1, H_2$)** Creates and returns a new heap containing all nodes from heaps $H_1$ and $H_2$

**Binary Heaps:** $\text{Union} = \Theta(n)$ worst case

**Binomial Heaps:** $\text{Union} = O(\log n)$ worst case

**Fibonacci Heaps:** $\text{Union} = \Theta(1)$ amortized

Other operations:

Extract-Min maintains partial ordering over keys. This is useful for many graph algorithms.
Binomial Heaps

A binomial heap is a set of binomial trees.
A binomial tree $B_K$ is an ordered tree such that

$$B_0$$

$$B_1$$

$$B_2$$

$$B_3$$

$B_k$ properties

1. There are $2^k$ nodes
2. Height of tree = $k$
3. There are exactly $\binom{k}{i}$ nodes at depth $i$ (this is why the tree is called a "binomial" tree)
   
   Review: this is $\frac{k!}{i!(k-i)!}$
4. Root has degree $k$ (children) and its children are $B_{k-1}, B_{k-2}, \ldots, B_0$ from left to right
Prove properties by induction on $k$

**Base Case:** Holds for $B_0$.

**Assume:** Holds for $B_0 \ldots B_{k-1}$.

1. $B_k$ is 2 copies of $B_{k-1}$, so $2^{k-1} + 2^{k-1} = 2^k$ nodes.
2. Depth of $B_k$ is one greater than maximum depth of $B_{k-1}$.
   Add one more level: height = $(k-1) + 1 = k$.
4. True for children $B_{k-1}, B_{k-2}, \ldots, B_0$ from left to right.
   $B_{k-1}$ is left child of $B_k$, root is also root of $B_{k-1}$ (minus left child), so
   degrees are $B_{k-1}, B_{k-2}, \ldots, B_0$.
   The root of $B_k$ is a $B_{k-1}$ with one more child (the left child), so root
   of $B_k$ has degree $(k-1) + 1 = k$.

---

**Binomial Heap Properties**

1. Each binomial tree is heap-ordered (key($x$) $\geq$ key(parent($x$))).
   This is the opposite of previous heap properties.
2. There never exist two or more trees with the same degree in the heap.

A binomial heap with $n$ nodes has at most $\lceil \lg n \rceil + 1$ binomial trees.

$n$ in binary = $< b_k, b_{k-1}, \ldots, b_0 >$ bits

$k = \lceil \lg n \rceil$, $n = \sum_{i=0}^{\lceil \lg n \rceil} b_i 2^i$

There is a one-to-one mapping between the binary representation and
binomial trees in a binomial heap.
If $b_i = 1$, then $B_i$ is in the heap
Recall that there are $2^i$ nodes in $B_i$

At most $\lceil \lg n \rceil + 1$ bits are needed to express $n$ base 2

---

**Example: Binomial Heap $H$**

![Binomial Heap Diagram]

$B_0 \rightarrow B_1 \rightarrow B_2$ (_________ nodes)
$B_0 \rightarrow B_2 \rightarrow B_5$ (_________ nodes)

---

**Operations**

Make-Heap() ($\Theta(1)$)

Minimum(H) ($O(\lg n)$)
   - Find minimum of roots of binomial trees in $H$
Operations

Union($H_1$, $H_2$)

Union($H_1$, $H_2$)

$H =$ new heap containing trees of $H_1$ and $H_2$ merged in non-decreasing order by degree of root

; Similar to Merge used in MergeSort

; $O(\lg n)$: at most two roots of each degree,

; $O(\lg n)$ possible degrees

; No more than 2 $B_i$ trees in $H$ at this point

; Could be 3 after linking two $B_{i-1}$ trees together

prev-$x =$ NIL ; three-tree window

x = head(H) ; look for:

\[ \begin{array}{c}
 x & next-x & j > i \\
 B_i & B_i & B_j
\end{array} \]

next-$x =$ sibling($x$)

while next-$x \neq$ NIL

if degree($x$) $\neq$ degree(next-$x$) or

degree($x$) = degree(next-$x$) = degree(sibling(next-$x$))

then move window right by one

else if key($x$) $\leq$ key(next-$x$)

then:
Running time = $O(\lg n)$ if $n = n_1 + n_2$ nodes in H.

---

Example

---

Operations

Insert($H$, $x$)
Insert\((H, x)\)
\[H' = x\]
\[H = \text{Union}(H, H')\]

Running time = \(O(\lg n)\)

Extract-Min\((H)\)

Extract-Min\((H)\)
Find root \(x\) with minimum key in \(H\); \(O(\lg n)\)
Remove \(x\) from \(H\); \(\Theta(1)\)
\(H' = \) children of \(x\) in reverse order; \(O(\lg n)\)
because children are \(B_{k-1}, B_{k-2}, \ldots, B_0\)
\(\text{Union}(H, H')\); \(O(\lg n)\)

Running time = \(O(\lg n)\)

---

**Operations**

Decrease-Key\((H, x, k)\), where \(k \leq \text{key}(x)\)

Decrease-Key\((H, x, k)\)
\[\text{key}(x) = k\]
while parent\((x) \neq \text{NIL}\) and \(\text{key}(x) < \text{key}(\text{parent}(x))\)
\[; ; ; "\text{bubble}" \text{ new key up}\]
swap\((\text{key}(x), \text{key}(\text{parent}(x)))\)
x = parent\((x)\)

\[\text{Max depth} = \lfloor \lg n \rfloor\]
Running time = \(O(\lg n)\)
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Binomial Heap (worst case)</th>
<th>Fibonacci Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make-Heap</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Minimum</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Extract-Min</td>
<td>$\Theta(\lg n)$</td>
<td>$O(\lg n)$</td>
</tr>
<tr>
<td>Union</td>
<td>$O(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$\Theta(\lg n)$</td>
<td>$O(\lg n)$</td>
</tr>
</tbody>
</table>

Delete(H, x)

$\text{Decrease-Key}(H, x, -\infty)$ ; $O(\lg n)$
$\text{Extract-Min}(H)$ ; $O(\lg n)$

Running time = $O(\lg n)$

**Fibonacci Heaps**

- If nodes are never removed, then yields $\Theta(1)$ performance
- Not designed for efficient search
Structure of Fibonacci Heaps

A **Fibonacci Heap** is a set of heap-ordered trees. Trees are not ordered binomial trees, because

1. Children of a node are unordered
2. Deleting nodes may destroy binomial construction

Node Structure:

```
+---+---+
| parent |
+---+---+
| key |
+---+---+
| mark |
+---+---+
| degree |
+---+---+
| left | right |
+---+---+
| child |
+---+---+
     | any child |
```

The field “mark” is True if the node has lost a child since the node became a child of another node.

The field “degree” contains the number of children of this node.

The structure contains a doubly-linked list of sibling nodes.

---

**Heap Structure**

\[ \text{min}(H) \] pointer to node in root list having smallest key in heap \( H \)

\[ n(H) \] number of nodes in heap \( H \)
Potential Function

\[ \Phi(H) = t(H) + 2m(H) \]
\[ t(H) = \# \text{trees in root list of heap } H \]
\[ m(H) = \# \text{mark nodes in heap } H \]

Example

\[ t(H) = 4, \quad m(H) = 2, \quad \Phi(H) = _{-} \]

Empty heap, \( \Phi(H_0) = 0, \Phi(H_i) \geq 0 \)
\[ \sum_{i=1}^{n} c_i \] is an upper bound on \( \sum_{i=1}^{n} c_i \)

Maximum Degree

\( D(n) = \) upper bound on degree of a node in a Fibonacci Heap with \( n \) nodes
By showing $D(n) = O(lg n)$, we can constrain running times for node removal.
1. Make node root
2. Delete
3. Add $O(lg n)$ children to root list

---

**Mergeable Heap Operations**

Make-Heap, Insert, Minimum, Extract-Min, Union

These always yield unordered binomial trees; thus, they maintain the binomial tree properties.

1. $2^k$ nodes
2. $k = \text{height of tree}$
3. $\binom{k}{i}$ nodes at depth $i$
4. Unordered binomial tree $U_k$ has root with degree $k$ greater than any other node. Children are trees $U_0, U_1, \ldots, U_{k-1}$ in some order.

For $n$-node Fibonacci Heap, $D(n)$ is largest if all nodes are in one tree. The maximum degree is at depth=1, $\binom{k}{1} = k$ for tree with $2^k$ nodes. If $n = 2^k$, then $k = lg n$

$$D(n) \leq k = lg n$$
$$D(n) = O(lg n)$$
Strategy

- Do not merge trees until necessary
- Merging done in Extract-Min, where new minimum is needed

Operations

Make-Heap

Make-Heap()
allocate(H)
min(H) = NIL
n(H) = 0

Analysis:
t(H) = m(H) = 0
Φ(H) = t(H) + 2m(H) = 0
\( \hat{c}_i = c_i = O(1) \)
Amortized cost equals actual cost.

Operations

Insert

Insert(H, x)
set x’s fields appropriately
add x to root list of H ; O(1)
reset min(H) if needed  
n(H) = n(H) + 1

Analysis:  
H = initial heap with t(H) trees and m(H) marked nodes  
H’ = new heap, t(H’) = t(H) + 1, m(H’) = m(H)  
\[ \hat{c}_i = c_i + \Phi(H') - \Phi(H) \]
\[ = O(1) + [t(H) + 1 + 2m(H)] - [t(H) + 2m(H)] \]
\[ = O(1) + 1 = O(1) \]

Operations  
Minimum  
Minimum(H)  
return min(H)

Analysis:  
H = H’  
\[ \Phi(H) = \Phi(H') \]
\[ \hat{c}_i = c_i = O(1) \]

Operations  
Union  
Union(\(H_1, H_2\))  
H = new heap whose root list contains roots from \(H_1\) and \(H_2\)
\[ n(H) = n(H_1) + n(H_2) \]
\[ \min(H) = \min(H_1) \]
if \((\min(H_1) = \text{NIL}) \) or \((\min(H_2) \neq \text{NIL} \) and \(\min(H_2) < \min(H_1)\))
then \(\min(H) = \min(H_2)\)

Analysis:
\[ t(H) = t(H_1) + t(H_2) \]
\[ m(H) = m(H_1) + m(H_2) \]

\[
\hat{c}_i = c_i + \Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\
= O(1) + [t(H_1) + t(H_2) + 2(m(H_1) + m(H_2))] \\
- [t(H_1) + 2m(H_1) + t(H_2) + 2m(H_2)] \\
= O(1) + 0 \\
= O(1)
\]

**Operations**

**Extract-Min**

**Extract-Min(H)**
\[ z = \min(H) \]
add \(z\)’s children to root list \(; O(D(n(H)))\)
remove \(z\) from root list
if root list \(\neq \{\}\)
then Consolidate(H) \(; O(D(n(H)))\)
else \(\min(H) = \text{NIL}\)
n(H) = n(H) - 1
Consolidate

Consolidate(H)
while two trees in H (T1, T2) have same degree
change root list to following using Link(H, T2, T1):

for i = 0 to D(n(H))
if tree T of degree i has root-key < min(H)
then min(H) = T

Example

Click mouse to advance to next frame.

Analysis

\[
\begin{align*}
n(H) &= n \\
\text{length(rootlist)} &\leq D(n) + t(H) - 1 \\
T(\text{while loop}) &\leq D(n) + t(H) \\
c_i &= O(D(n) + t(H)) \\
\Phi(H) &= t(H) + 2m(H) \\
\Phi(H') &\leq D(n) + 1 + 2m(H)
\end{align*}
\]
\[ \dot{c}_i = c_i + (D(n) + 1 + 2m(H)) - (t(H) + 2m(H)) \\
= O(D(n) + t(H)) + D(n) + 1 - t(H) \\
= O(D(n)) \]

Assuming adjustment of potential coefficients to dominate coefficients hidden in \( O(t(H)) \).

---

**Operations**

**Decrease-Key**

Decrease-Key\((H, x, k)\)

\[ \text{key}(x) = k \]

\[ p = \text{parent}(x) \]

if \( p \neq \text{NIL} \) and \( \text{key}(x) < \text{key}(p) \)
then Cut\((H, x, p)\)

Cascading-Cut\((H, p)\)

if \( \text{key}(x) < \text{key}(\text{min}(H)) \)
then \( \text{min}(H) = x \)

Cut\((H, x, p)\)

remove \( x \) from children of \( p \)
add \( x \) to root list of \( H \)
\[ \text{mark}(x) = \text{False} \]

Cascading-Cut\((H, p)\)

\[ \text{next-p} = \text{parent}(p) \]

if \( \text{next-p} \neq \text{NIL} \)
then if \( \text{mark}(p) = \text{False} \)
then \( \text{mark}(p) = \text{True} \)
else Cut(H, p, next-p)
    Cascading-Cut(H, next-p)

---

**Analysis**

Let $c_i = O(c)$ be the number of cascading cuts

$\Phi(H') = (t(H) + c) + 2(m(H) - c + 2)$, c-1 unmarked, 1 marked

$\hat{c}_i = O(c) + (t(H) + c) + 2(m(H) - c + 2) - (t(H) + 2m(H))$

$= O(c) + 4 - c$

$= O(1)$

---

**Example**

Click mouse to advance to next frame.

---

**Operations**

Delete

Delete(H, x)

- Decrease-Key(H, x, -\infty) ; O(1) amortized
- Extract-Min(H) ; O(D(n)) amortized

---

Analysis:

Running time = O(D(n)) amortized
Bounding Maximum Degree D(n)

**Lemma 21.1**

\[ F_{k+2} = 1 + \sum_{i=0}^{k} F_i, \text{ where } F_k \text{ is a Fibonacci number.} \]

\[ F_k = \begin{cases} 
  k & \text{if } k < 2 \\
  F_{k-1} + F_{k-2} & \text{if } k \geq 2 
\end{cases} \]

**Lemma 21.3**

For a node x in a Fibonacci heap, where k = degree(x),

\[ \text{size}(x) \geq F_{k+2} \geq \phi^k, \text{ where } \phi = \frac{1+\sqrt{5}}{2} \]

\[ \text{size}(x) = \# \text{nodes in subtree rooted at } x \]

**Corollary 21.4**

\[ D(n) = O(\log n) \]

By Lemma 21.3, \( n \geq \text{size}(x) \geq \phi^k \), where n = nodes in Fibonacci Heap and k = degree of any node x.

Then \( \log_\phi n \geq k \), and \( k = O(\log_\phi n) = O(\log n) \).

---

**Applications**