CSE 548: Analysis of Algorithms

Lecture 10
(Dijkstra’s SSSP & Fibonacci Heaps)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2019
A *Fibonacci heap* can be viewed as an extension of Binomial heaps which supports **DECREASE-KEY** and **DELETE** operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap (worst-case)</th>
<th>Binomial 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>$O(\log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(\log n)$</td>
<td>–</td>
</tr>
<tr>
<td>DELETE</td>
<td>$O(\log n)$</td>
<td>–</td>
</tr>
</tbody>
</table>
A Fibonacci heap can be viewed as an extension of Binomial heaps which supports `DECREASE-KEY` and `DELETE` operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap (worst-case)</th>
<th>Binomial Heap (amortized)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MAKE-HEAP</code></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>INSERT</code></td>
<td>$O(\log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>MINIMUM</code></td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>EXTRACT-MIN</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><code>UNION</code></td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td><code>DECREASE-KEY</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (worst case)</td>
</tr>
<tr>
<td><code>DELETE</code></td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ (worst case)</td>
</tr>
</tbody>
</table>
**Fibonacci Heaps**
( Fredman & Tarjan, 1984 )

A Fibonacci heap can be viewed as an extension of Binomial heaps which supports **DECREASE-KEY** and **DELETE** operations efficiently.

<table>
<thead>
<tr>
<th>Heap Operation</th>
<th>Binary Heap ( worst-case )</th>
<th>Binomial Heap ( amortized )</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>INSERT</td>
<td>$O(\log n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ ( worst case )</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$ ( amortized )</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

```
Dijkstra-SSSP ( G = (V,E), w, s )
1. for each v \in G[V] do v.d \leftarrow \infty
2. s.d \leftarrow 0
3. H \leftarrow \phi \quad \{ \text{empty min-heap} \}
4. for each v \in G[V] do INSERT( H, v )
5. while H \neq \emptyset do
6. \quad u \leftarrow EXTRACT-MIN( H )
7. \quad for each v \in Adj[u] do
8. \quad \quad if v.d > u.d + w_{u,v} then
9. \quad \quad DECREASE-KEY( H, v, u.d + w_{u,v} )
10. \quad \quad v.d \leftarrow u.d + w_{u,v}
```
Dijkstra’s SSSP Algorithm with a Min-Heap
(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

---

**Dijkstra-SSSP**($G = (V, E)$, $w$, $s$)

1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \emptyset$ \{ empty min-heap \}
4. for each $v \in G[V]$ do INSERT($H$, $v$)
5. while $H \neq \emptyset$ do
6.   $u \leftarrow$ EXTRACT-MIN($H$)
7.   for each $v \in Adj[u]$ do
8.     if $v.d > u.d + w_{u,v}$ then
9.     DECREASE-KEY($H$, $v$, $u.d + w_{u,v}$)
10. $v.d \leftarrow u.d + w_{u,v}$

Let $n = |G[V]|$ and $m = |G[E]|$

# INSERTS = n
# EXTRACT-MINS = n
# DECREASE-KEYS ≤ m

**Total cost**

\[ \leq n(cost_{Insert} + cost_{Extract-Min}) + m(cost_{Decrease-Key}) \]
Dijkstra's SSSP Algorithm with a Min-Heap
(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph \( G = (V, E) \) with vertex set \( V \) and edge set \( E \), a weight function \( w \), and a source vertex \( s \in G[V] \).

**Output:** For all \( v \in G[V] \), \( v.d \) is set to the shortest distance from \( s \) to \( v \).

Let \( n = |G[V]| \) and \( m = |G[E]| \)

For Binary Heap (worst-case costs):

- \( cost_{\text{Insert}} = O(\log n) \)
- \( cost_{\text{Extract-Min}} = O(\log n) \)
- \( cost_{\text{Decrease-Key}} = O(\log n) \)

\[
\therefore \text{Total cost (worst-case)} = O((m + n) \log n)
\]
**Dijkstra’s SSSP Algorithm with a Min-Heap**  
( **SSSP: Single-Source Shortest Paths** )

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

Let $n = |G[V]|$ and $m = |G[E]|$

For Binomial Heap ( amortized costs ):

- $cost_{Insert} = O(1)$
- $cost_{Extract-Min} = O(\log n)$
- $cost_{Decrease-Key} = O(\log n)$  
  \hspace{1cm} ( worst-case )

\[ \therefore \text{Total cost ( worst-case )} \]
\[ = O((m + n) \log n) \]
**Dijkstra’s SSSP Algorithm with a Min-Heap**

(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

---

Dijkstra-SSSP ($G = (V, E)$, $w$, $s$)

1. for each $v \in G[V]$ do $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \phi$  
   { empty min-heap }
4. for each $v \in G[V]$ do INSERT($H$, $v$)
5. while $H \neq \emptyset$ do
6. \hspace{1em} $u \leftarrow \text{EXTRACT-MIN}(H)$
7. \hspace{1em} for each $v \in \text{Adj}[u]$ do
8. \hspace{2em} if $v.d > u.d + w_{u,v}$ then
9. \hspace{3.5em} $\text{DECREASE-KEY}(H, v, u.d + w_{u,v})$
10. \hspace{2em} $v.d \leftarrow u.d + w_{u,v}$

Let $n = |G[V]|$ and $m = |G[E]|$

Total cost

$\leq n (\text{cost}_{\text{Insert}} + \text{cost}_{\text{Extract-Min}}) + m (\text{cost}_{\text{Decrease-Key}})$

**Observation:**

Obtaining a worst-case bound for a sequence of $n$ INSERTS, $n$ EXTRACT-MINS and $m$ DECREASE-KEYS is enough.

∴ Amortized bound per operation is sufficient.
Dijkstra’s SSSP Algorithm with a Min-Heap
( SSSP: Single-Source Shortest Paths )

Input: Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

Output: For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

Let $n = |G[V]|$ and $m = |G[E]|$

Total cost
$$\leq n(\text{cost}_{\text{Insert}} + \text{cost}_{\text{Extract-Min}}) + m(\text{cost}_{\text{Decrease-Key}})$$

Observation:
For $n(\text{cost}_{\text{Insert}} + \text{cost}_{\text{Extract-Min}})$ the best possible bound is $\Theta(n \log n)$.
( else violates sorting lower bound )

Perhaps $m(\text{cost}_{\text{Decrease-Key}})$ can be improved to $o(m \log n)$. 

---

Dijkstra-SSSP ($G = (V,E), w, s$)
1. \textbf{for each} $v \in G[V]$ \textbf{do} $v.d \leftarrow \infty$
2. $s.d \leftarrow 0$
3. $H \leftarrow \phi$ \hspace{1cm} \{ empty min-heap \}
4. \textbf{for each} $v \in G[V]$ \textbf{do} \text{INSERT}( H, v )
5. \textbf{while} $H \neq \emptyset$ \textbf{do}
6. \hspace{1cm} $u \leftarrow \text{EXTRACT-MIN}( H )$
7. \hspace{1cm} \textbf{for each} $v \in \text{Adj}[u]$ \textbf{do}
8. \hspace{2cm} \textbf{if} $v.d > u.d + w_{u,v}$ \textbf{then}
9. \hspace{3cm} $\text{DECREASE-KEY}( H, v, u.d + w_{u,v} )$
10. \hspace{2cm} $v.d \leftarrow u.d + w_{u,v}$
A Fibonacci heap can be viewed as an extension of Binomial heaps which supports DECREASE-KEY and DELETE operations efficiently.

But the trees in a Fibonacci heap are no longer binomial trees as we will be cutting subtrees out of them.

However, all operations (except DECREASE-KEY and DELETE) are still performed in the same way as in binomial heaps.

The rank of a tree is still defined as the number of children of the root, and we still link two trees if they have the same rank.
Implementing **DECREASE-KEY**(*H, x, k*)

**DECREASE-KEY**(*H, x, k*): One possible approach is to cut out the subtree rooted at *x* from *H*, reduce the value of *x* to *k*, and insert that subtree into the root list of *H*.

**Problem**: If we cut out a lot of subtrees from a tree its size will no longer be exponential in its rank. Since our analysis of **EXTRACT-MIN** in binomial heaps was highly dependent on this exponential relationship, that analysis will no longer hold.

**Solution**: Limit #cuts among the children of any node to 2. We will show that the size of each tree will still remain exponential in its rank.

When a 2nd child is cut from a node *x*, we also cut *x* from its parent leading to a possible sequence of cuts moving up towards the root.
Analysis of Fibonacci Heap Operations

Recurrence for Fibonacci numbers: \( f_n = \begin{cases} 
0 & \text{if } n = 0, \\
1 & \text{if } n = 1, \\
f_{n-1} + f_{n-2} & \text{otherwise.} 
\end{cases} \)

We showed in a previous lecture: \( f_n = \frac{1}{\sqrt{5}} \left( \phi^n - \hat{\phi}^n \right), \)

where \( \phi = \frac{1 + \sqrt{5}}{2} \) and \( \hat{\phi} = \frac{1 - \sqrt{5}}{2} \) are the roots \( z^2 - z - 1 = 0. \)
### Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>0</th>
<th></th>
<th>1</th>
<th>$1 + f_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>1</td>
<td></td>
<td>2</td>
<td>$1 + f_0 + f_1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1</td>
<td></td>
<td>3</td>
<td>$1 + f_0 + f_1 + f_2$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>2</td>
<td></td>
<td>5</td>
<td>$1 + f_0 + f_1 + f_2 + f_3$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>3</td>
<td></td>
<td>8</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>5</td>
<td></td>
<td>13</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5$</td>
</tr>
<tr>
<td>$f_6$</td>
<td>8</td>
<td></td>
<td>21</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6$</td>
</tr>
<tr>
<td>$f_7$</td>
<td>13</td>
<td></td>
<td>34</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$</td>
</tr>
<tr>
<td>$f_8$</td>
<td>21</td>
<td></td>
<td>55</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8$</td>
</tr>
<tr>
<td>$f_9$</td>
<td>34</td>
<td></td>
<td>89</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9$</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>55</td>
<td></td>
<td>144</td>
<td>$1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10}$</td>
</tr>
</tbody>
</table>
## Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>0</th>
<th>$f_1$</th>
<th>1</th>
<th>$f_2$</th>
<th>1</th>
<th>$f_3$</th>
<th>2</th>
<th>$f_4$</th>
<th>3</th>
<th>$f_5$</th>
<th>5</th>
<th>$f_6$</th>
<th>8</th>
<th>$f_7$</th>
<th>13</th>
<th>$f_8$</th>
<th>21</th>
<th>$f_9$</th>
<th>34</th>
<th>$f_{10}$</th>
<th>55</th>
<th>$f_{11}$</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0$</td>
<td>$f_1$</td>
<td>$f_2$</td>
<td>$f_3$</td>
<td>$f_4$</td>
<td>$f_5$</td>
<td>$f_6$</td>
<td>$f_7$</td>
<td>$f_8$</td>
<td>$f_9$</td>
<td>$f_{10}$</td>
<td>$f_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 1 | $1 + f_0$ | 1 | $1 + f_0 + f_1$ | 1 | $1 + f_0 + f_1 + f_2$ | 1 | $1 + f_0 + f_1 + f_2 + f_3$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9$ | 1 | $1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10}$ |
# Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>1</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1</td>
</tr>
<tr>
<td>$f_3$</td>
<td>2</td>
</tr>
<tr>
<td>$f_4$</td>
<td>3</td>
</tr>
<tr>
<td>$f_5$</td>
<td>5</td>
</tr>
<tr>
<td>$f_6$</td>
<td>8</td>
</tr>
<tr>
<td>$f_7$</td>
<td>13</td>
</tr>
<tr>
<td>$f_8$</td>
<td>21</td>
</tr>
<tr>
<td>$f_9$</td>
<td>34</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>55</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>89</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>144</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
1 & = 1 \\
1 + f_0 & = 1 + 1 \\
1 + f_0 + f_1 & = 2 \\
1 + f_0 + f_1 + f_2 & = 3 \\
1 + f_0 + f_1 + f_2 + f_3 & = 5 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 & = 8 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 & = 13 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 & = 21 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 & = 34 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 & = 55 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 & = 89 \\
1 + f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 + f_{10} & = 144
\end{align*}
\]

**Lemma 1:** For all integers $n \geq 0$, $f_{n+2} = 1 + \sum_{i=0}^{n} f_i$. 
**Lemma 1:** For all integers \( n \geq 0 \), \( f_{n+2} = 1 + \sum_{i=0}^{n} f_i \).

**Proof:** By induction on \( n \).

Base case: \( f_2 = 1 = 1 + 0 = 1 + f_0 = 1 + \sum_{i=0}^{n} f_i \).

Inductive hypothesis: \( f_{k+2} = 1 + \sum_{i=0}^{k} f_i \) for \( 0 \leq k \leq n - 1 \).

Then \( f_{n+2} = f_{n+1} + f_n = f_n + (1 + \sum_{i=0}^{n-1} f_i) = 1 + \sum_{i=0}^{n} f_i \).

**Analysis of Fibonacci Heap Operations**
## Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>0</th>
<th>$&lt;$</th>
<th>1.00</th>
<th>$\phi^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>1</td>
<td>$&lt;$</td>
<td>1.62</td>
<td>$\phi^1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1</td>
<td>$&lt;$</td>
<td>2.62</td>
<td>$\phi^2$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>2</td>
<td>$&lt;$</td>
<td>4.24</td>
<td>$\phi^3$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>3</td>
<td>$&lt;$</td>
<td>6.85</td>
<td>$\phi^4$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>5</td>
<td>$&lt;$</td>
<td>11.09</td>
<td>$\phi^5$</td>
</tr>
<tr>
<td>$f_6$</td>
<td>8</td>
<td>$&lt;$</td>
<td>17.94</td>
<td>$\phi^6$</td>
</tr>
<tr>
<td>$f_7$</td>
<td>13</td>
<td>$&lt;$</td>
<td>29.03</td>
<td>$\phi^7$</td>
</tr>
<tr>
<td>$f_8$</td>
<td>21</td>
<td>$&lt;$</td>
<td>46.98</td>
<td>$\phi^8$</td>
</tr>
<tr>
<td>$f_9$</td>
<td>34</td>
<td>$&lt;$</td>
<td>76.01</td>
<td>$\phi^9$</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>55</td>
<td>$&lt;$</td>
<td>122.99</td>
<td>$\phi^{10}$</td>
</tr>
</tbody>
</table>
# Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
<td>55</td>
<td>89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.00</th>
<th>$\phi^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.62</td>
<td>$\phi^1$</td>
</tr>
<tr>
<td>2.62</td>
<td>$\phi^2$</td>
</tr>
<tr>
<td>4.24</td>
<td>$\phi^3$</td>
</tr>
<tr>
<td>6.85</td>
<td>$\phi^4$</td>
</tr>
<tr>
<td>11.09</td>
<td>$\phi^5$</td>
</tr>
<tr>
<td>17.94</td>
<td>$\phi^6$</td>
</tr>
<tr>
<td>29.03</td>
<td>$\phi^7$</td>
</tr>
<tr>
<td>46.98</td>
<td>$\phi^8$</td>
</tr>
<tr>
<td>76.01</td>
<td>$\phi^9$</td>
</tr>
<tr>
<td>122.99</td>
<td>$\phi^{10}$</td>
</tr>
</tbody>
</table>

$\phi$ is the golden ratio, \( \phi = \frac{1 + \sqrt{5}}{2} \).
### Analysis of Fibonacci Heap Operations

<table>
<thead>
<tr>
<th>$f_0$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>1</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$\geq$ 1.00</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$\geq$ 2.00</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$\geq$ 3.25</td>
</tr>
<tr>
<td>$f_6$</td>
<td>$\geq$ 5.00</td>
</tr>
<tr>
<td>$f_7$</td>
<td>$\geq$ 7.92</td>
</tr>
<tr>
<td>$f_8$</td>
<td>$\geq$ 12.00</td>
</tr>
<tr>
<td>$f_9$</td>
<td>$\geq$ 19.50</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>$\geq$ 31.50</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>$\geq$ 49.80</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>$\geq$ 81.30</td>
</tr>
</tbody>
</table>

**Lemma 2:** For all integers $n \geq 0$, $f_{n+2} \geq \phi^n$. 
Lemma 2: For all integers \( n \geq 0 \), \( f_{n+2} \geq \phi^n \).

Proof: By induction on \( n \).

Base case: \( f_2 = 1 = \phi^0 \) and \( f_3 = 2 > \phi^1 \).

Inductive hypothesis: \( f_{k+2} \geq \phi^k \) for \( 0 \leq k \leq n - 1 \).

Then \( f_{n+2} = f_{n+1} + f_n \)
\[ \geq \phi^{n-1} + \phi^{n-2} \]
\[ = (\phi + 1)\phi^{n-2} \]
\[ = \phi^2 \phi^{n-2} \]
\[ = \phi^n \]
Lemma 3: Let $x$ be any node in a Fibonacci heap, and suppose that $k = rank(x)$. Let $y_1, y_2, ..., y_k$ be the children of $x$ in the order in which they were linked to $x$, from the earliest to the latest. Then $rank(y_i) \geq \max\{0, i - 2\}$ for $1 \leq i \leq k$.

Proof: Obviously, $rank(y_1) \geq 0$.

For $i > 1$, when $y_i$ was linked to $x$, all of $y_1, y_2, ..., y_{i-1}$ were children of $x$. So, $rank(x) \geq i - 1$.

Because $y_i$ is linked to $x$ only if $rank(y_i) = rank(x)$, we must have had $rank(y_i) \geq i - 1$ at that time.

Since then, $y_i$ has lost at most one child, and hence $rank(y_i) \geq i - 2$. 
Lemma 4: Let $z$ be any node in a Fibonacci heap with $n = \text{size}(z)$ and $r = \text{rank}(z)$. Then $r \leq \log \phi n$.

Proof: Let $s_k$ be the minimum possible size of any node of rank $k$ in any Fibonacci heap.

Trivially, $s_0 = 1$ and $s_1 = 2$.

Since adding children to a node cannot decrease its size, $s_k$ increases monotonically with $k$.

Let $x$ be a node in any Fibonacci heap with $\text{rank}(x) = r$ and $\text{size}(x) = s_r$. 

Analysis of Fibonacci Heap Operations
Lemma 4: Let \( z \) be any node in a Fibonacci heap with \( n = \text{size}(z) \) and \( r = \text{rank}(z) \). Then \( r \leq \log_\phi n \).

Proof (continued): Let \( y_1, y_2, \ldots, y_r \) be the children of \( x \) in the order in which they were linked to \( x \), from the earliest to the latest.

Then \( s_r \geq 1 + \sum_{i=1}^{r} \text{rank}(y_i) \geq 1 + \sum_{i=1}^{r} \max\{0, i-2\} = 2 + \sum_{i=2}^{r} s_{i-2} \)

We now show by induction on \( r \) that \( s_r \geq f_{r+2} \) for all integer \( r \geq 0 \).

Base case: \( s_0 = 1 = f_2 \) and \( s_1 = 2 = f_3 \).

Inductive hypothesis: \( s_k \geq f_{k+2} \) for \( 0 \leq k \leq r - 1 \).

Then \( s_r \geq 2 + \sum_{i=2}^{r} s_{i-2} \geq 2 + \sum_{i=2}^{r} f_i = 1 + \sum_{i=1}^{r} f_i = f_{r+2} \).

Hence \( n \geq s_r \geq f_{r+2} \geq \phi^r \Rightarrow r \leq \log_\phi n \).
Corollary: The maximum degree of any node in an $n$ node Fibonacci heap is $O(\log n)$.

Proof: Let $z$ be any node in the heap.

Then from Lemma 4,

$$\text{degree}(z) = \text{rank}(z) \leq \log_\phi (\text{size}(z)) \leq \log_\phi n = O(\log n).$$
Analysis of Fibonacci Heap Operations

All nodes are initially unmarked.

We mark a node when
- it loses its first child

We unmark a node when
- it loses its second child, or
- becomes the child of another node (e.g., LINKed)

We extend the potential function used for binomial heaps:

\[ \Phi(D_i) = 2t(D_i) + 3m(D_i), \]

where \( D_i \) is the state of the data structure after the \( i^{th} \) operation,
\( t(D_i) \) is the number of trees in the root list, and
\( m(D_i) \) is the number of marked nodes.
Analysis of Fibonacci Heap Operations

We extend the potential function used for binomial heaps:

$$\Phi(D_i) = 2t(D_i) + 3m(D_i),$$

where $D_i$ is the state of the data structure after the $i^{th}$ operation, $t(D_i)$ is the number of trees in the root list, and $m(D_i)$ is the number of marked nodes.

**Decrease-Key**($H, x, k_x$): Let $k = \#\text{cascading cuts performed}$.

Then the actual cost of cutting the tree rooted at $x$ is 1, and the actual cost of each of the cascading cuts is also 1.

$$\therefore \text{overall actual cost, } c_i = 1 + k$$
**Fibonacci Heaps from Binomial Heaps**

Potential function: \( \Phi(D_i) = 2t(D_i) + 3m(D_i) \)

**DECREASE-KEY( \( H, x, k_x \) )**:

New trees: 1 tree rooted at \( x \), and

1 tree produced by each of the \( k \) cascading cuts.

\[ t(D_i) - t(D_{i-1}) = 1 + k \]

Marked nodes: 1 node unmarked by each cascading cut, and

at most 1 node marked by the last cut/cascading cut.

\[ m(D_i) - m(D_{i-1}) \leq -k + 1 \]

Potential drop, \( \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) \)

\[ \begin{align*}
\Delta_i &= 2(t(D_i) - t(D_{i-1})) + 3(m(D_i) - m(D_{i-1})) \\
&\leq 2(1 + k) + 3(-k + 1) \\
&= -k + 5
\end{align*} \]
Fibonacci Heaps from Binomial Heaps

Potential function: $\Phi(D_i) = 2t(D_i) + 3m(D_i)$

**DECREASE-KEY**$(H, x, k_x)$:

Amortized cost, $\hat{c}_i = c_i + \Delta_i$

$\leq (1 + k) + (-k + 5)$

$= 6$

$= O(1)$
**Fibonacci Heaps from Binomial Heaps**

Potential function: \( \Phi(D_i) = 2t(D_i) + 3m(D_i) \)

**EXTRACT-MIN( H ):**

Let \( d_n \) be the max degree of any node in an \( n \)-node Fibonacci heap.

Cost of creating the array of pointers is \( \leq d_n + 1 \).

Suppose we start with \( k \) trees in the doubly linked list, and perform \( l \) link operations during the conversion from linked list to array version. So we perform \( k + l \) work, and end up with \( k - l \) trees.

Cost of converting to the linked list version is \( k - l \).

Actual cost, \( c_i \leq d_n + 1 + (k + l) + (k - l) = 2k + d_n + 1 \)

Since no node is marked, and each link reduces the #trees by 1, potential change, \( \Delta_i = \Phi(D_i) - \Phi(D_{i-1}) \geq -2l \)
Fibonacci Heaps from Binomial Heaps

Potential function: $\Phi(D_i) = 2t(D_i) + 3m(D_i)$

**EXTRACT-MIN($H$):**

actual cost, $c_i \leq d_n + 1 + (k + l) + (k - l) = 2k + d_n + 1$

potential change, $\Delta_i = \Phi(D_i) - \Phi(D_{i-1}) \geq -2l$

amortized cost, $\hat{c}_i = c_i + \Delta_i \leq 2(k - l) + d_n + 1$

But $k - l \leq d_n + 1$ (as we have at most one tree of each rank)

So, $\hat{c}_i \leq 3d_n + 3 = O(\log n)$. 
Fibonacci Heaps from Binomial Heaps

Potential function: $\Phi(D_i) = 2t(D_i) + 3m(D_i)$

**DELETE**($H$, $x$):

**STEP 1:** **DECREASE-KEY**($H$, $x$, $-\infty$)

**STEP 2:** **EXTRACT-MIN**($H$)

amortized cost, $\hat{c}_i = \text{amortized cost of } \text{DECREASE-KEY}$

+ amortized cost of **EXTRACT-MIN**

$= O(1) + O(\log n)$

$= O(\log n)$