5.2 Fibonacci Heaps (Analysis)

Frank Stajano    Thomas Sauerwald

Lent 2016
Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree
Fibonacci Heap: INSERT

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

**Example:**
- Actual Costs: $O(1)$
Fibonacci Heap: INSERT

- Create a singleton tree
**Fibonacci Heap: INSERT**

- Create a singleton tree
- Add to root list
Fibonacci Heap: INSERT

- Create a singleton tree
- Add to root list

Actual Costs: $O(1)$
Fibonacci Heap: \textsc{INSERT}

- Create a singleton tree
- Add to root list and update min-pointer (if necessary)
**Fibonacci Heap: INSERT**

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

**Actual Costs: $O(1)$**

```
   17  24  23  7  21
  /   /   /   /   /
 30  26  46  3  18
   /   /   /   /   /
 35  39  52  41  44
```

Actual Costs: $O(1)$
Fibonacci Heap: **EXTRACT-MIN**

**EXTRACT-MIN**

- **Delete min**
  - Meld children into root list and unmark them
  - Consolidate so that no roots have the same degree
  - Update minimum

```
min
18
39
degree=2
0
1
2
0
degree=0
0
1
2
3
```

**Actual Costs:** $O(\text{trees}(H) + d(n))$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$.

---

5.2: Fibonacci Heaps

T.S. 14
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**

  - Delete min
**Fibonacci Heap: EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓

```plaintext
Every root becomes child of another root at most once!

d(n) is the maximum degree of a root in any Fibonacci heap of size n.
```

Actual Costs:

\[ O\left(\text{trees}\left(\text{H}\right) + d(n)\right) \]

5.2: Fibonacci Heaps

T.S.
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them

Actual Costs:

\[ O\left(\text{trees}(H) + d(n)\right) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: **EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them

---

**Actual Costs:**

\[ O(\text{trees}(H) + d(n)) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them
**Fibonacci Heap: EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓

Actual Costs:

$\mathcal{O}(\text{trees}(H) + d(n))$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$. 

---

5.2: Fibonacci Heaps

T.S.
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree

![Diagram of Fibonacci Heap]

Actual Costs:

\[ O\left( \text{trees} \left( H \right) \right) + d\left( n \right) \]

Every root becomes child of another root at most once!

\[ d\left( n \right) \] is the maximum degree of a root in any Fibonacci heap of size \( n \)
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

Actual Costs:

\[ O(\text{trees}(H) + d(n)) \]

Every root becomes child of another root at most once!

\( d(n) \) is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children)

Actual Costs: $O(\text{trees}(H) + d(n))$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$.  

degree=2

```
7
  30

24
  26
  35

46

23

17

18

52

41

39

44
```
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children)

---

**Actual Costs:**

\[ O\left(\frac{\log n}{\log \log n}\right) \]

Every root becomes child of another root at most once!

\[ d(n) \]

is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children)

---

**Actual Costs:**

\[O\left(\text{trees}\left(H\right) + d(n)\right)\]

\(d(n)\) is the maximum degree of a root in any Fibonacci heap of size \(n\).
Fibonacci Heap: **EXTRACT-MIN**

- **Delete min ✓**
- **Meld children into root list and unmark them ✓**
- **Consolidate** so that no roots have the same degree (# children)

**Actual Costs:**
\[ O(\text{trees}(H) + d(n)) \]

Every root becomes child of another root at most once!

\( d(n) \) is the maximum degree of a root in any Fibonacci heap of size \( n \)

---

5.2: Fibonacci Heaps  
T.S.  
14
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

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

Actual Costs:

\[ O(\text{trees}(H) + d(n)) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
**Fibonacci Heap: EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children)

**Actual Costs:** $O\left(\sum_{i=0}^{H} d_n\right)$

Every root becomes child of another root at most once!

$d_n$ is the maximum degree of a root in any Fibonacci heap of size $n$.
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

Actual Costs:

\[ O\left(\text{trees (H)} + d(n)\right) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \)
**Fibonacci Heap: EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children)

---

**Diagram:***

- Degree table:
  - Degree 0
  - Degree 1
  - Degree 2
  - Degree 3

- Root nodes:
  - 7
  - 24
  - 23
  - 17
  - 18
  - 52
  - 41

- Children:
  - 30
  - 26
  - 46
  - 35
  - 39
  - 44

---

**Actual Costs:**

\[ O\left(trees(H) + d(n)\right) \]

Every root becomes child of another root at most once!
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children)

**Actual Costs:**

\[ O(\text{trees}(H) + d(n)) \]

Every root becomes child of another root at most once!

d(n) is the maximum degree of a root in any Fibonacci heap of size n.
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

Actual Costs: \( O(\text{trees}(H) + \text{d}(n)) \)

Every root becomes child of another root at most once!

\( d(n) \) is the maximum degree of a root in any Fibonacci heap of size \( n \).
**Fibonacci Heap: EXTRACT-MIN**

- **Delete min ✓**
- **Meld children into root list and unmark them ✓**
- **Consolidate** so that no roots have the same degree (# children)

---

**Actual Costs:**

\[ O\left(\text{trees}(H) + d(n)\right) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

Actual Costs:
$O\left(\text{trees} + d(n)\right)$

Every root becomes child of another root at most once!
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min √
  - Meld children into root list and unmark them √
  - **Consolidate** so that no roots have the same degree (# children)

---

Actual Costs:
\[
O\left(\sum_{H \in \text{trees}} d(H) + d(n)\right)
\]

Every root becomes child of another root at most once!

\(d(n)\) is the maximum degree of a root in any Fibonacci heap of size \(n\).

---

5.2: Fibonacci Heaps

T.S.

14
Fibonacci Heap: \textsc{Extract-Min}

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

**Actual Costs:** $O\left(\text{trees}(H) + d(n)\right)$

Every root becomes child of another root at most once!

\textbf{degree}

\begin{tabular}{cccc}
0 & 1 & 2 & 3 \\
\end{tabular}

7 \rightarrow 24 \rightarrow 26 \rightarrow \ldots \rightarrow 52

17 \rightarrow 23

18 \rightarrow 39

35

30

46

41

44
Fibonacci Heap: **EXTRACT-MIN**

- **Delete min** ✓
- **Meld children into root list and unmark them** ✓
- **Consolidate** so that no roots have the same degree (# children)

---

**Actual Costs:**

\[ O\left( \text{trees}(H) + d(n) \right) \]

Every root becomes child of another root at most once!

\(d(n)\) is the maximum degree of a root in any Fibonacci heap of size \(n\).
Fibonacci Heap: **EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children)

---

**Actual Costs:**

\[ O \left( \text{trees}(H) + d(n) \right) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

Actual Costs: \( O(\text{trees}(H) + d(n)) \)

Every root becomes child of another root at most once!

\( d(n) \) is the maximum degree of a root in any Fibonacci heap of size \( n \).
**Fibonacci Heap: EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children)

**Actual Costs:**

\[ O\left(\text{trees (H)} + d(n)\right) \]

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \).
**Fibonacci Heap: EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - Consolidate so that no roots have the same degree (# children)

**degree**

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

**Actual Costs:**

\[ O\left( H \right) + d\left( n \right) \]

Every root becomes child of another root at most once!

\[ d\left( n \right) \] is the maximum degree of a root in any Fibonacci heap of size \( n \)
Fibonacci Heap: \textsc{Extract-Min}

- Delete min ✓
- Meld children into root list and unmark them ✓
- \textbf{Consolidate} so that no roots have the same degree (\# children)

\begin{center}
\begin{tabular}{c|c|c|c|c}
\hline
degree & 0 & 1 & 2 & 3 \\
\hline
\end{tabular}
\end{center}

\begin{itemize}
\item 7
\item 17
\item 30
\item 24
\item 23
\item 26
\item 46
\item 35
\item 18
\item 52
\item 41
\item 39
\item 44
\end{itemize}
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - Consolidate so that no roots have the same degree (# children)

```
Actual Costs: O(trees(H)) + d(n)
```

Every root becomes child of another root at most once!

d(n) is the maximum degree of a root in any Fibonacci heap of size n.

5.2: Fibonacci Heaps

T.S. 14
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - Consolidate so that no roots have the same degree (# children)

**Actual Costs:** $O(\text{trees} + d(n))$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$.
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

Actual Costs:
$O(H + d(n))$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$. 

degree

0 1 2 3

node degrees

17 30 24
23
26 46
35

18 39
52
41
44

17 30 24
23
26 46
35

5.2: Fibonacci Heaps
Fibonacci Heap: **EXTRACT-MIN**

- **Delete min ✓**
- **Meld children into root list and unmark them ✓**
- **Consolidate** so that no roots have the same degree (# children)

---

Degree of roots:

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

---

5.2: Fibonacci Heaps

T.S.
Fibonacci Heap: \textit{EXTRACT-MIN}

- Delete min $\checkmark$
- Meld children into root list and unmark them $\checkmark$
- \textbf{Consolidate} so that no roots have the same degree (# children)

\begin{itemize}
  \item Actual Costs: $O(\text{trees}(H) + d(n))$
  \item Every root becomes child of another root at most once!
  \item $d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$
\end{itemize}
**Fibonacci Heap: EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

---

**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>

---

**Actual Costs:** $O\left(\text{trees} \left( H \right) + d\left( n \right)\right)$

Every root becomes child of another root at most once!

$d\left( n \right)$ is the maximum degree of a root in any Fibonacci heap of size $n$. 

---

5.2: Fibonacci Heaps
Fibonacci Heap: \textbf{EXTRACT-MIN}

- Delete min ✓
- Meld children into root list and unmark them ✓
- \textbf{Consolidate} so that no roots have the same degree (# children)

\begin{itemize}
  \item Actual Costs: $O\left( H + d(n) \right)$
  \item Every root becomes child of another root at most once!
  \item $d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$
\end{itemize}
**Fibonacci Heap: EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - Consolidate so that no roots have the same degree (# children)

---

**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>

---

Actual Costs: $O\left(\sum_{i=1}^{n} d(i)\right)$

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$. 

5.2: Fibonacci Heaps
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children)

**Actual Costs:**

\[ O(\text{trees} (H) + d(n)) \]

Every root becomes child of another root at most once!

d\(n\) is the maximum degree of a root in any Fibonacci heap of size \(n\)

---

**Diagram:**

- **Degree Table:**
  - Degree 0
  - Degree 1
  - Degree 2
  - Degree 3

- **Trees: 7, 17, 30, 24, 26, 35, 46, 18, 41, 44, 39, 52**

- **Consolidation Process:**
  - Trees are merged and unmarked.
  - Degree consolidation ensures no root has the same degree.

---

*5.2: Fibonacci Heaps*
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children)
Fibonacci Heap: EXTRACT-MIN

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
Fibonacci Heap: **EXTRACT-MIN**

**EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children) ✓
- Update minimum

---

**Actual Costs:** $O\left(\text{trees}\left(\text{H}\right) + d\left(n\right)\right)$

Every root becomes child of another root at most once!

$d\left(n\right)$ is the maximum degree of a root in any Fibonacci heap of size $n$.

---

5.2: Fibonacci Heaps
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children) ✓
  - Update minimum ✓

Actual Costs:

\[ O\left(\text{trees}(H) + d(n)\right) \]

Every root becomes child of another root at most once!

\( d(n) \) is the maximum degree of a root in any Fibonacci heap of size \( n \)

5.2: Fibonacci Heaps
**Fibonacci Heap: EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

**Actual Costs:**

```
min
7
17 30
24
23 30

18
41 44
39
52
```
Fibonacci Heap: **EXTRACT-MIN**

- Delete min ✓
- Meld children into root list and unmark them ✓
- **Consolidate** so that no roots have the same degree (# children) ✓
- Update minimum ✓

Every root becomes child of another root at most once!

*Actual Costs:*

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size $n$
Fibonacci Heap: **EXTRACT-MIN**

- **EXTRACT-MIN**
  - Delete min ✓
  - Meld children into root list and unmark them ✓
  - **Consolidate** so that no roots have the same degree (# children) ✓
  - Update minimum ✓

Every root becomes child of another root at most once!

\[ d(n) \] is the maximum degree of a root in any Fibonacci heap of size \( n \)

**Actual Costs:** \( \mathcal{O}(\text{trees}(H) + d(n)) \)

---

5.2: Fibonacci Heaps  
T.S.  
14
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY** of node $x$

- Decrease the key of $x$ (given by a pointer)
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

![Diagram of Fibonacci Heap]

**Actual Cost:** $O(\# \text{ cuts})$

1. **DECREASE-KEY 46 $$\sim$$ 15**
**Fibonacci Heap: DECREASE-KEY**

- **DECREASE-KEY of node x**
  - Decrease the key of x (given by a pointer)
  - (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at x, unmark x, meld into root list

```
1. DECREASE-KEY 46 \rightarrow 15
```

Actual Cost:

\[ O(\# \text{ cuts}) \]
**Fibonacci Heap: DECREASE-KEY**

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)
- Cut tree rooted at $x$, unmark $x$, meld into root list

**DECREASE-KEY of node $x$**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1. D</td>
</tr>
<tr>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>98</td>
<td>35</td>
</tr>
<tr>
<td>99</td>
<td>38</td>
</tr>
<tr>
<td>18</td>
<td>41</td>
</tr>
<tr>
<td>18</td>
<td>52</td>
</tr>
<tr>
<td>39</td>
<td>46</td>
</tr>
</tbody>
</table>

**Actual Cost:** $O(\# \text{ cuts})$

1. **DECREASE-KEY 46 $\leadsto$ 15**
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node $x$

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at $x$, unmark $x$, meld into root list

Actual Cost: $O(\# \text{ cuts})$

1. DECREASE-KEY 46 $\leadsto$ 15
Fibonacci Heap: **DECREASE-KEY**

- Decrease the key of \( x \) (given by a pointer)
- (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at \( x \), unmark \( x \), meld into root list

---

**DECREASE-KEY of node** \( x \)

- Decrease the key of \( x \) (given by a pointer)
- (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at \( x \), unmark \( x \), meld into root list

---

Actual Cost:

\[
O\left(\# \text{ cuts}\right)
\]

1. **DECREASE-KEY** 46 \(\leadsto\) 15
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

---

**Actual Cost:** $O(\# \text{ cuts})$

1. **DECREASE-KEY 46 \(\sim\) 15

---

[Diagram showing a Fibonacci heap with nodes and arrows indicating the decrease-key operation.]
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
- Check if parent node is marked

```
min

7

24

26

98

17

15

30

35

23

18

21

39

52

18

38

41

99

15

26

15

15

15

99

24

5

5

5

26

30

35

98

Actual Cost: O(\# cuts)
```

1. **DECREASE-KEY 46 → 15**
Fibonacci Heap: DECREASE-KEY

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)

```
min

7

24 17 23

26 15 30

98 35

18 21 39

52

38 41

99

1. DECREASE-KEY 46 15
```
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
  - Check if parent node is marked
    - If unmarked, mark it (unless it is a root)

---

![Diagram of Fibonacci Heap]

**Actual Cost:** $O(\#\text{ cuts})$

1. DECREASE-KEY 46 → 15
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY** of node $x$

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at $x$, unmark $x$, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)

---

**min**

```
7
```

```
24
26
```

```
23
17
```

```
```

```
18
39
```

```
15
38
41
```

```
30
52
```

```
98
35
```

**Actual Cost:** $O(\# \text{ cuts})$

1. DECREASE-KEY 46 ↗ 15 ✓
**Fibonacci Heap: ** **DECREASE-KEY**

---

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
  - Check if parent node is marked
    - If unmarked, mark it (unless it is a root)

---

**Actual Cost:** $O(\# \text{ cuts})$

1. **DECREASE-KEY 46 \(\rightarrow\) 15 ✓
2. **DECREASE-KEY 35 \(\rightarrow\) 5
Fibonacci Heap: **DECREASE-KEY**

- **DECREASE-KEY** of node $x$
  - Decrease the key of $x$ (given by a pointer)
  - (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at $x$, unmark $x$, meld into root list and:
    - Check if parent node is marked
      - If unmarked, mark it (unless it is a root)

**min**

```
7
```

```
24
23
17
18

26
30
21
39
```

```
98
38
41
35
52
15
5
99
```

1. **DECREASE-KEY** 46 ⇒ 15 ✓
2. **DECREASE-KEY** 35 ⇒ 5
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)

---

### Example

1. **DECREASE-KEY** 46 → 15 ✓
2. **DECREASE-KEY** 35 → 5
Fibonacci Heap: DECREASE-KEY

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list **and**:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)

```
min

7

24

26

98

17

30

39

38

41

18

23

21

52

15

5

24

35

15

Actual Cost: O(# cuts)
```

1. **DECREASE-KEY** 46 \(\rightarrow\) 15 ✓
2. **DECREASE-KEY** 35 \(\rightarrow\) 5
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)

---

**Actual Cost:** $O(\#\text{ cuts})$

1. **DECREASE-KEY** 46 $\leadsto$ 15 ✔
2. **DECREASE-KEY** 35 $\leadsto$ 5

---

5.2: Fibonacci Heaps  
T.S.  
15
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node $x$**

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at $x$, unmark $x$, meld into root list **and**:
  - Check if parent node is marked
    - If unmarked, mark it (unless it is a root)

```
min

7

24

30

26

98

24

17

23

18

15

39

46

30

21

52

38

41

99

5

1. DECREASE-KEY 46 $\leadsto$ 15 ✓
2. DECREASE-KEY 35 $\leadsto$ 5
```
**Fibonacci Heap: DECREASE-KEY**

- Decrease the key of \(x\) (given by a pointer)
- (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at \(x\), unmark \(x\), meld into root list and:
    - Check if parent node is marked
      - If unmarked, mark it (unless it is a root)
      - If marked,

---

### Decrease-Key of node \(x\)

1. **DECREASE-KEY** 46 \(\sim\) 15 ✓
2. **DECREASE-KEY** 35 \(\sim\) 5
**Decrease-Key** of node $x$

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at $x$, unmark $x$, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

---

1. Decrease-Key $46 \rightarrow 15 \checkmark$
2. Decrease-Key $35 \rightarrow 5$

---

5.2: Fibonacci Heaps
**Fibonacci Heap: DECREASE-KEY**

- **DECREASE-KEY of node** \( x \)
  - Decrease the key of \( x \) (given by a pointer)
  - (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at \( x \), unmark \( x \), meld into root list and:
    - Check if parent node is marked
      - If unmarked, mark it (unless it is a root)
      - If marked, unmark and meld it into root list and recurse (Cascading Cut)

---

1. **DECREASE-KEY 46 \( \sim \) 15 ✓
2. **DECREASE-KEY 35 \( \sim \) 5
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY** of node \( x \):

- Decrease the key of \( x \) (given by a pointer)
- (Here we consider only cases where heap-order is violated)

\[ \Rightarrow \] Cut tree rooted at \( x \), unmark \( x \), meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (*Cascading Cut*)

---

**Actual Cost:** \( O(\# \text{ cuts}) \)

1. **DECREASE-KEY** 46 \( \rightarrow \) 15 \( \checkmark \)
2. **DECREASE-KEY** 35 \( \rightarrow \) 5
Fibonacci Heap: **DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
  - Check if parent node is marked
    - If unmarked, mark it (unless it is a root)
    - If marked, unmark and meld it into root list and recurse (**Cascading Cut**)

![Diagram of a Fibonacci heap showing Decrease-Key operations]

1. **DECREASE-KEY 46 ↛ 15 ✓**
2. **DECREASE-KEY 35 ↛ 5**

Actual Cost: \(O(\# \text{ cuts})\)

5.2: Fibonacci Heaps
### Fibonacci Heap: \textit{DECREASE-KEY}

\textbf{DECREASE-KEY} of node \( x \):

- Decrease the key of \( x \) (given by a pointer)
- (Here we consider only cases where heap-order is violated)

\Rightarrow \text{Cut tree rooted at } x, \text{ unmark } x, \text{ meld into root list and:}

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (\textit{Cascading Cut})

\[ \begin{array}{c}
\text{min} \\
7 & 18 & 38 & 15 & 5 & 26 & 24 \\
24 & 17 & 23 & 39 & 41 & 99 & 98 \\
26 & 30 & 21 & 52 & & & \\
98 & 5 & & & & & \\
\end{array} \]

1. \textit{DECREASE-KEY} 46 \( \sim \) 15 \( \checkmark \)
2. \textit{DECREASE-KEY} 35 \( \sim \) 5
Fibonacci Heap: DECREASE-KEY

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

---

1. **DECREASE-KEY 46 ~ 15 ✓**
2. **DECREASE-KEY 35 ~ 5**
Fibonacci Heap: **DECREASE-KEY**

- **DECREASE-KEY** of node $x$
  - Decrease the key of $x$ (given by a pointer)
  - (Here we consider only cases where heap-order is violated)
  - Cut tree rooted at $x$, unmark $x$, meld into root list and:
    - Check if parent node is marked
      - If unmarked, mark it (unless it is a root)
      - If marked, unmark and meld it into root list and recurse (**Cascading Cut**)

---

**Actual Cost:** $O(\# \text{ cuts})$

1. **DECREASE-KEY** 46 $\leadsto$ 15 ✓
2. **DECREASE-KEY** 35 $\leadsto$ 5
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list **and:**

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (**Cascading Cut**)

---

**Diagram:**

- **Node 39** is the node where the key is decreased.
- **Min node:** The minimum node in the heap is 5.
- **Decrease-Key Operations:**
  1. DECREASE-KEY 46 ↪ 15 ✓
  2. DECREASE-KEY 35 ↪ 5 ✓
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY of node x**

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

⇒ Cut tree rooted at x, unmark x, meld into root list and:
  - Check if parent node is marked
    - If unmarked, mark it (unless it is a root)
    - If marked, unmark and meld it into root list and recurse (Cascading Cut)

---

**Actual Cost:**

1. \( \text{DECREASE-KEY } 46 \rightarrow 15 \checkmark \)
2. \( \text{DECREASE-KEY } 35 \rightarrow 5 \checkmark \)
**Fibonacci Heap: DECREASE-KEY**

**DECREASE-KEY** of node $x$

- Decrease the key of $x$ (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at $x$, unmark $x$, meld into root list and:
- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

```
7 18 38 15 5 26 24
17 23 21 39 41 99 98
30 52
```

- Actual Cost: $\mathcal{O}(\#\text{ cuts})$

1. **DECREASE-KEY** 46 $\rightarrow$ 15 ✓
2. **DECREASE-KEY** 35 $\rightarrow$ 5 ✓
Outline

Recap of **INSERT**, **EXTRACT-MIN** and **DECREASE-KEY**

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree
Amortized Analysis via Potential Method

- **INSERT:** actual $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$
**Amortized Analysis via Potential Method**

- **INSERT:** actual $\mathcal{O}(1)$
- **EXTRACT-MIN:** actual $\mathcal{O}(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $\mathcal{O}($# cuts$) \leq \mathcal{O}($marks$(H))$

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$
Amortized Analysis via Potential Method

- **INSERT:** actual $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$
Amortized Analysis via Potential Method

- **INSERT**: actual $O(1)$
- **EXTRACT-MIN**: actual $O(\text{trees}(H) + d(n))$
- **DECREASE-KEY**: actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$

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

Lifecycle of a node:
- Loses first child
- Loses second child

5.2: Fibonacci Heaps (Analysis)
**Amortized Analysis via Potential Method**

- **INSERT:** actual $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]
Amortized Analysis via Potential Method

- **INSERT**: actual $O(1)$  
  amortized $O(1)$
- **EXTRACT-MIN**: actual $O(\text{trees}(H) + d(n))$  
  amortized $O(d(n))$
- **DECREASE-KEY**: actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$  
  amortized $O(1)$

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]
Amortized Analysis via Potential Method

- **INSERT**: actual $O(1)$  
  amortized $O(1)$ ✓

- **EXTRACT-MIN**: actual $O(\text{trees}(H) + d(n))$  
  amortized $O(d(n))$ ?

- **DECREASE-KEY**: actual $O(\# \text{ cuts}) \leq O(\text{marks}(H))$  
  amortized $O(1)$ ?

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

Lifecycle of a node
Outline

Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree
**Amortized Analysis of DECREASE-KEY**

<table>
<thead>
<tr>
<th>Actual Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DECREASE-KEY</strong>: $O(x + 1)$, where $x$ is the number of cuts.</td>
</tr>
</tbody>
</table>
Amortized Analysis of **DECREASE-KEY**

Actual Cost

- **DECREASE-KEY**: $O(x + 1)$, where $x$ is the number of cuts.

$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$
Amortized Analysis of DECREASE-KEY

- **Actual Cost**
  - **DECREASE-KEY**: \( O(x + 1) \), where \( x \) is the number of cuts.

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

Change in Potential

\[
\tilde{c}_i = c_i + \Delta \Phi \leq O(x + 1) + 4 - x = O(1)
\]
Amortized Analysis of **DECREASE-KEY**

Actual Cost

- **DECREASE-KEY**: $O(x + 1)$, where $x$ is the number of cuts.

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

Change in Potential

- $\text{trees}(H') =$

Change in Potential Diagram:

- $\tilde{c}_i = c_i + \Delta \Phi \\ \Rightarrow \tilde{c}_i = O(1)$
Amortized Analysis of **DECREASE-KEY**

- **Actual Cost**
  - **DECREASE-KEY**: $O(x + 1)$, where $x$ is the number of cuts.

- **Actual Cost**
  - $\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$

- **Change in Potential**
  - $\text{trees}(H') = \text{trees}(H) + x$

---

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of DECREASE-KEY

Actual Cost

- **DECREASE-KEY**: \( \mathcal{O}(x + 1) \), where \( x \) is the number of cuts.

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

Change in Potential

- \( \text{trees}(H') = \text{trees}(H) + x \)
- \( \text{marks}(H') \leq \)

Change in Potential Diagram:

- \( \text{trees}(H') = \text{trees}(H) + x \)
- \( \text{marks}(H') \leq \)

Amortized Cost

Scale up potential units

First Coin: pays cut

Second Coin: increase of trees (\( H' \))
Amortized Analysis of \textsc{Decrease-Key}

- **Actual Cost**
  - \textbf{\textsc{Decrease-Key}}: $O(x + 1)$, where $x$ is the number of cuts.

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

- **Change in Potential**
  - $\text{trees}(H') = \text{trees}(H) + x$
  - $\text{marks}(H') \leq \text{marks}(H) - x + 2$

Change in Potential

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of \textsc{Decrease-Key}

- **Actual Cost**
  - \textsc{Decrease-Key}: $O(x + 1)$, where $x$ is the number of cuts.

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

- **Change in Potential**
  - $\text{trees}(H') = \text{trees}(H) + x$
  - $\text{marks}(H') \leq \text{marks}(H) - x + 2$
  - $\Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x$. 

5.2: Fibonacci Heaps (Analysis)  
T.S. 5
Amortized Analysis of **DECREASE-KEY**

**Actual Cost**
- **DECREASE-KEY**: $\mathcal{O}(x + 1)$, where $x$ is the number of cuts.

**Φ(H) = trees(H) + 2 \cdot marks(H)**

**Change in Potential**
- $trees(H') = trees(H) + x$
- $marks(H') \leq marks(H) - x + 2$
  \[ \Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x. \]

**Amortized Cost**
- $\tilde{c_i} = c_i + \Delta \Phi$

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of DECREASE-KEY

Actual Cost

- **DECREASE-KEY**: $O(x + 1)$, where $x$ is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$
  \[\Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x.\]

Amortized Cost

\[
\tilde{c}_i = c_i + \Delta \Phi \leq O(x + 1) + 4 - x
\]
Amortized Analysis of **DECREASE-KEY**

**Actual Cost**

- **DECREASE-KEY**: $\mathcal{O}(x + 1)$, where $x$ is the number of cuts.

**Actual Cost**

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

**Change in Potential**

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$$\Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x.$$  

**Amortized Cost**

$$\tilde{c}_i = c_i + \Delta \Phi \leq \mathcal{O}(x + 1) + 4 - x = \mathcal{O}(1)$$

5.2: Fibonacci Heaps (Analysis)

T.S. 5
Amortized Analysis of **DECREASE-KEY**

**Actual Cost**
- **DECREASE-KEY**: $O(x + 1)$, where $x$ is the number of cuts.

**Change in Potential**
- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$
  \[\Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x.\]

**Amortized Cost**
\[
\tilde{c}_i = c_i + \Delta \Phi \leq O(x + 1) + 4 - x = O(1)
\]
Amortized Analysis of \texttt{EXTRACT-MIN}

- Actual Cost
  - \texttt{EXTRACT-MIN}: $O(\text{trees}(H) + d(n))$
Amortized Analysis of $\text{EXTRACT-MIN}$

- Actual Cost
  - $\text{EXTRACT-MIN}: \mathcal{O}(\text{trees}(H) + d(n))$

\[
\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)
\]
Amortized Analysis of \texttt{EXTRACT-Min}

- **Actual Cost**
  - \texttt{EXTRACT-Min}: $\mathcal{O}(\text{trees}(H) + d(n))$

- **Change in Potential**

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

---

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of \textsc{Extract-Min}

- **Actual Cost**
  - \textsc{Extract-Min}: $O(\text{trees}(H) + d(n))$

- **Potential Function**
  - $\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$

- **Change in Potential**
  - \text{marks}(H') \neq \text{marks}(H)$
Amortized Analysis of **EXTRACT-MIN**

**Actual Cost**
- **EXTRACT-MIN**: $O(\text{trees}(H) + d(n))$

**Change in Potential**
- $\text{marks}(H')$ vs. $\text{marks}(H)$

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$
Amortized Analysis of \textsc{Extract-Min}

Actual Cost
- \textsc{Extract-Min}: $O(\text{trees}(H) + d(n))$

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

Change in Potential
- $\text{marks}(H') \leq \text{marks}(H)$
Amortized Analysis of $\text{EXTRACT-MIN}$

**Actual Cost**
- $\text{EXTRACT-MIN}: \mathcal{O}(\text{trees}(H) + d(n))$

**Φ(H) = trees(H) + 2 \cdot \text{marks}(H)**

**Change in Potential**
- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq$

**Degrees**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>d(n)</th>
</tr>
</thead>
</table>

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of \texttt{EXTRACT-MIN}

### Actual Cost
- \texttt{EXTRACT-MIN}: $\mathcal{O}(\text{trees}(H) + d(n))$

### Change in Potential
- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq \cdot$

### Potential Function
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

### Change in Potential
- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq$

### Degrees
```
0 1 2 3    
\downarrow \uparrow \uparrow \uparrow 
\downarrow \downarrow \downarrow \downarrow 
```

- $d(n)$
Amortized Analysis of **EXTRACT-MIN**

- **Actual Cost**
  - **EXTRACT-MIN**: $O(\text{trees}(H) + d(n))$

- **ϕ(H)**:
  - $\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$

- **Change in Potential**
  - $\text{marks}(H') \leq \text{marks}(H)$
  - $\text{trees}(H') \leq d(n) + 1$
Amortized Analysis of \textsc{Extract-Min}

- Actual Cost
  - \textsc{Extract-Min}: $\mathcal{O}(\text{trees}(H) + d(n))$

- Change in Potential
  - $\text{marks}(H') \leq \text{marks}(H)$
  - $\text{trees}(H') \leq d(n) + 1$
  - $\Rightarrow \Delta \Phi \leq d(n) + 1 - \text{trees}(H)$

Φ(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)
Amortized Analysis of EXTRACT-MIN

Actual Cost

- **EXTRACT-MIN**: $O(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$
  \[\Delta \Phi \leq d(n) + 1 - \text{trees}(H)\]

Amortized Cost

\[\tilde{c}_i = c_i + \Delta \Phi\]
Amortized Analysis of EXTRACT-MIN

- **Actual Cost**
  - EXTRACT-MIN: $O(\text{trees}(H) + d(n))$

- **Φ(H)**
  - $\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$

- **Change in Potential**
  - $\text{marks}(H') \leq \text{marks}(H)$
  - $\text{trees}(H') \leq d(n) + 1$
  - $\Rightarrow \Delta \Phi \leq d(n) + 1 - \text{trees}(H)$

- **Amortized Cost**
  - $\tilde{c}_i = c_i + \Delta \Phi \leq O(\text{trees}(H) + d(n)) + d(n) + 1 - \text{trees}(H)$

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of \textsc{Extract-Min}

**Actual Cost**
- \textbf{\textsc{Extract-Min}}: $\mathcal{O}(\text{trees}(H) + d(n))$

**Change in Potential**
- \text{marks}(H') \leq \text{marks}(H)
- \text{trees}(H') \leq d(n) + 1
  \Rightarrow \Delta \Phi \leq d(n) + 1 - \text{trees}(H)

**Amortized Cost**
\[
\tilde{c}_i = c_i + \Delta \Phi \leq \mathcal{O}(\text{trees}(H) + d(n)) + d(n) + 1 - \text{trees}(H) = \mathcal{O}(d(n))
\]
Amortized Analysis of EXTRACT-MIN

Actual Cost
- **EXTRACT-MIN**: $O(\text{trees}(H) + d(n))$

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

Change in Potential
- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$
  \[ \Rightarrow \Delta \Phi \leq d(n) + 1 - \text{trees}(H) \]

Amortized Cost
\[
\tilde{c}_i = c_i + \Delta \Phi \leq O(\text{trees}(H) + d(n)) + d(n) + 1 - \text{trees}(H) = O(d(n))
\]

How to bound $d(n)$?
Outline

Recap of **INSERT**, **EXTRACT-MIN** and **DECREASE-KEY**

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree
Bounding the Maximum Degree

Binomial Heap

Every tree is a binomial tree \( \Rightarrow d(n) \leq \log_2 n \).
Bounding the Maximum Degree

Binomial Heap

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$. 

5.2: Fibonacci Heaps (Analysis)
Binomial Heap

Every tree is a binomial tree ⇒ $d(n) \leq \log_2 n$.

$d = 3$, $n = 2^3$
Bounding the Maximum Degree

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$. 

Binomial Heap

Not all trees are binomial trees, but still $d(n) \leq \log_\varphi n$, where $\varphi \approx 1.62$. 

5.2: Fibonacci Heaps (Analysis)
Bounding the Maximum Degree

**Binomial Heap**

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.

**Fibonacci Heap**

Not all trees are binomial trees, but still $d(n) \leq \log_\varphi n$, where $\varphi \approx 1.62$. 
Bounding the Maximum Degree

Every tree is a binomial tree \( \Rightarrow d(n) \leq \log_2 n \).

Binomial Heap

Not all trees are binomial trees, but still \( d(n) \leq \log_\varphi n \), where \( \varphi \approx 1.62 \).

Fibonacci Heap

Skip Analysis
Lower Bounding Degrees of Children

\[ d(n) \leq \log_\phi n \]
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[
d(n) \leq \log_\varphi n
\]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$\displaystyle d(n) \leq \log_\varphi n$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree \( k \) contains at least \( \varphi^{k} \) nodes.

\[
d(n) \leq \log_\varphi n
\]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment
We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state.
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment.
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

We have:
\[
d(n) \leq \log_\varphi n
\]

Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[
d(n) \leq \log_\varphi n
\]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state.
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment.
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_{\varphi} n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$d(n) \leq \log_\varphi n$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement:
Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[ d(n) \leq \log \varphi n \]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$d(n) \leq \log_\varphi n$

Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
Lower Bounding Degrees of Children

We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \phi^k \) nodes.

\[
d(n) \leq \log_\phi n
\]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state.
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment.
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_{\varphi} n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$d(n) \leq \log_\varphi n$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

![Diagram showing a tree with node $x$ and its children $y_1, y_2, y_3, y_4, \ldots, y_k$.]
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
Lower Bounding Degrees of Children

We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$d(n) \leq \log\varphi n$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment
We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[
d(n) \leq \log_\varphi n
\]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment
Lower Bounding Degrees of Children

We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

\[
\begin{align*}
\Rightarrow \forall 1 \leq i \leq k: d_i \geq i - 2
\end{align*}
\]

\[ x \]

\[
\begin{array}{cccccc}
& y_1 & \rightarrow & y_2 & \rightarrow & y_3 & \rightarrow & y_4 & \rightarrow & \ldots & \rightarrow & y_k \\
\end{array}
\]
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state.
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment.
We will prove a stronger statement: Any tree with degree \( k \) contains at least \( \varphi^k \) nodes.

\[ d(n) \leq \log_{\varphi} n \]

- Consider any node \( x \) of degree \( k \) (not necessarily a root) at the final state.
- Let \( y_1, y_2, \ldots, y_k \) be the children in the order of attachment and \( d_1, d_2, \ldots, d_k \) be their degrees.
We will prove a stronger statement: Any tree with degree $k$ contains at least $\varphi^k$ nodes.

\[ d(n) \leq \log_\varphi n \]

- Consider any node $x$ of degree $k$ (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment and $d_1, d_2, \ldots, d_k$ be their degrees

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)
Theorem

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
∀ 1 ≤ i ≤ k: \( d_i \geq i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
N(0) \\
\bullet 0
\]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{align*}
N(0) &= 0 \\
N(1) &= 1
\end{align*}
\]
Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree $k$.
From Degrees to Minimum Subtree Sizes

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

\[ \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \]

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{array}{c|c|c}
N(0) & N(1) & N(2) \\
0 & 1 & 0
\end{array}
\]
Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree $k$.

\[ N(0) = 0, \quad N(1) = 1, \quad N(2) = 2 \]
**From Degrees to Minimum Subtree Sizes**

\[ \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

<table>
<thead>
<tr>
<th></th>
<th>( N(0) )</th>
<th>( N(1) )</th>
<th>( N(2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From Degrees to Minimum Subtree Sizes

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{array}{cccc}
N(0) & N(1) & N(2) & N(3) \\
0 & 1 & 2 & 5 \\
0 & 0 & 0 & 0
\end{array}
\]
\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the **minimum possible number of nodes** of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree $k$.

$N(0) = 0$

$N(1) = 1$

$N(2) = 2$

$N(3) = 3$
∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

<table>
<thead>
<tr>
<th>( N(0) )</th>
<th>( N(1) )</th>
<th>( N(2) )</th>
<th>( N(3) )</th>
<th>( N(4) )</th>
</tr>
</thead>
</table>
| 0 | 1 | 2 | 3 | \( 1 \)  
  0  
  0  
  0  
  1  
  0 |
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i ≥ i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

\[ \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \]

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{align*}
N(0) &= 0 \\
N(1) &= 1 \\
N(2) &= 2 \\
N(3) &= 3 \\
N(4) &= 4
\end{align*}
\]
$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$

**Definition**

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree $k$.
∀1 ≤ i ≤ k: \( d_i ≥ i − 2 \)

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i ≥ i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{align*}
N(0) &= 1 & N(1) &= 2 & N(2) = 3 & N(3) = 5 & N(4) = 8
\end{align*}
\]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

- \( N(0) = 1 \)
- \( N(1) = 2 \)
- \( N(2) = 3 \)
- \( N(3) \)
- \( N(4) \)
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i ≥ i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
N(0) = 1 \quad N(1) = 2 \quad N(2) = 3 \quad N(3) = 5 \quad N(4)
\]

5.2: Fibonacci Heaps (Analysis)
From Degrees to Minimum Subtree Sizes

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\forall 1 \leq i \leq k: \quad d_i \geq i - 2
\]

\[
N(0) = 1 \quad N(1) = 2 \quad N(2) = 3 \quad N(3) = 5 \quad N(4) = 8
\]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i ≥ i − 2 \)

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{align*}
N(0) &= 1 & N(1) &= 2 & N(2) &= 3 & N(3) &= 5 & N(4) &= 8 \\
0 & & 1 & & 2 & & 3 & & 4 \\
0 & & 0 & & 0 & & 0 & & 0
\end{align*}
\]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i ≥ i - 2 \)

**Definition**

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
\begin{align*}
N(0) &= 1 \\
N(1) &= 2 \\
N(2) &= 3 \\
N(3) &= 5 \\
N(4) &= 8
\end{align*}
\]
Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree $k$. 

\[
\begin{align*}
N(0) &= 1 \\
N(1) &= 2 \\
N(2) &= 3 \\
N(3) &= 5 \\
N(4) &= 8
\end{align*}
\]
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: d_i ≥ i − 2

Definition
Let N(k) be the minimum possible number of nodes of a subtree rooted at a node of degree k.

N(0) = 1  N(1) = 2  N(2) = 3  N(3) = 5  N(4) = 8 = 5 + 3
From Degrees to Minimum Subtree Sizes

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

Definition

Let \( N(k) \) be the minimum possible number of nodes of a subtree rooted at a node of degree \( k \).

\[
N(k) = F(k + 2)\
\]

\[
\begin{align*}
N(0) &= 1 \\
N(1) &= 2 \\
N(2) &= 3 \\
N(3) &= 5 \\
N(4) &= 8 = 5 + 3
\end{align*}
\]
∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

\[ N(k) = F(k + 2) \]
From Minimum Subtree Sizes to Fibonacci Numbers

\[ \forall 1 \leq i \leq k: \; d_i \geq i - 2 \]

\[ N(k) = F(k + 2)? \]
From Minimum Subtree Sizes to Fibonacci Numbers

\( \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \)

\[ N(k) = F(k + 2) ? \]

\[ N(k) = \]

\[ \begin{array}{c}
1 \\
N(2 - 2) \\
N(3 - 2) \\
\ldots \\
N(k - 2)
\end{array} \]

\[ N(k) = 1 + 1 + N(2 - 2) + N(3 - 2) + \cdots + N(k - 2) \]

\[ = 1 + 1 + \sum_{\ell=0}^{k-2} N(\ell) \]

\[ = 1 + 1 + \sum_{\ell=0}^{k-3} N(\ell) + N(k - 2) \]

\[ = N(k - 1) + N(k - 2) \]

\[ = F(k + 1) + F(k) = F(k + 2) \]
Exponential Growth of Fibonacci Numbers

Lemma 19.3
For all integers $k \geq 0$, the $(k + 2)$nd Fib. number satisfies $F(k + 2) \geq \varphi^k$, where $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$. 
Lemma 19.3

For all integers $k \geq 0$, the $(k + 2)$nd Fib. number satisfies $F(k + 2) \geq \varphi^k$, where $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in $k$. 
Exponential Growth of Fibonacci Numbers

Lemma 19.3
For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1 + \sqrt{5})/2 = 1.61803 \ldots$.

Proof by induction on $k$:

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in $k$. 
Exponential Growth of Fibonacci Numbers

Lemma 19.3
For all integers \( k \geq 0 \), the \((k + 2)\)nd Fib. number satisfies \( F(k + 2) \geq \varphi^k \), where \( \varphi = \left(1 + \sqrt{5}\right)/2 = 1.61803\ldots \).

\[ \varphi^2 = \varphi + 1 \]

Fibonacci Numbers grow at least exponentially fast in \( k \).

Proof by induction on \( k \):
- Base \( k = 0 \): \( F(2) = 1 \) and \( \varphi^0 = 1 \)
Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = \frac{1 + \sqrt{5}}{2} = 1.61803\ldots$.

Proof by induction on $k$:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1 \checkmark$
Exponential Growth of Fibonacci Numbers

For all integers $k \geq 0$, the $(k + 2)$nd Fib. number satisfies $F(k + 2) \geq \varphi^k$, where $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$

Proof by induction on $k$:

- **Base $k = 0$**: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- **Base $k = 1$**: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$
Lemma 19.3

For all integers \( k \geq 0 \), the \((k + 2)\)nd Fib. number satisfies \( F(k + 2) \geq \varphi^k \), where \( \varphi = (1 + \sqrt{5})/2 = 1.61803 \ldots \).

\( \varphi^2 = \varphi + 1 \)

Proof by induction on \( k \):

- Base \( k = 0 \): \( F(2) = 1 \) and \( \varphi^0 = 1 \)
- Base \( k = 1 \): \( F(3) = 2 \) and \( \varphi^1 \approx 1.619 < 2 \)
Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers \( k \geq 0 \), the \((k+2)\)nd Fib. number satisfies \( F(k+2) \geq \varphi^k \), where \( \varphi = \frac{(1 + \sqrt{5})}{2} = 1.61803\ldots \).

\[ \varphi^2 = \varphi + 1 \]

Fibonacci Numbers grow at least exponentially fast in \( k \).

Proof by induction on \( k \):

- **Base \( k = 0 \):** \( F(2) = 1 \) and \( \varphi^0 = 1 \) \( \checkmark \)
- **Base \( k = 1 \):** \( F(3) = 2 \) and \( \varphi^1 \approx 1.619 < 2 \) \( \checkmark \)
- **Inductive Step \( k \geq 2 \):**

\[ F(k+2) = \]
Lemma 19.3

For all integers \( k \geq 0 \), the \((k+2)\)nd Fib. number satisfies \( F(k+2) \geq \varphi^k \), where \( \varphi = (1 + \sqrt{5})/2 = 1.61803 \ldots \).

\[ \varphi^2 = \varphi + 1 \]

Fibonacci Numbers grow at least exponentially fast in \( k \).

Proof by induction on \( k \):
- Base \( k = 0 \): \( F(2) = 1 \) and \( \varphi^0 = 1 \) \( \checkmark \)
- Base \( k = 1 \): \( F(3) = 2 \) and \( \varphi^1 \approx 1.619 < 2 \) \( \checkmark \)
- Inductive Step \( (k \geq 2) \):
  \[
  F(k + 2) = F(k + 1) + F(k)
  \]
Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers \( k \geq 0 \), the \((k+2)\)nd Fib. number satisfies \( F(k+2) \geq \varphi^k \), where \( \varphi = (1 + \sqrt{5})/2 = 1.61803 \ldots \).

\[ \varphi^2 = \varphi + 1 \]

Fibonacci Numbers grow at least exponentially fast in \( k \).

Proof by induction on \( k \):

- **Base** \( k = 0 \): \( F(2) = 1 \) and \( \varphi^0 = 1 \) \( \checkmark \)
- **Base** \( k = 1 \): \( F(3) = 2 \) and \( \varphi^1 \approx 1.619 < 2 \) \( \checkmark \)
- **Inductive Step** \( (k \geq 2) \):

\[
F(k+2) = F(k+1) + F(k) \\
\geq \varphi^{k-1} + \varphi^{k-2} \quad \text{(by the inductive hypothesis)}
\]
Exponential Growth of Fibonacci Numbers

Lemma 19.3
For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$

Proof by induction on $k$:
- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step ($k \geq 2$):
  \[
  F(k+2) = F(k+1) + F(k) \\
  \geq \varphi^{k-1} + \varphi^{k-2} \\
  = \varphi^{k-2} \cdot (\varphi + 1)
  \] (by the inductive hypothesis)
Lemma 19.3

For all integers \( k \geq 0 \), the \((k + 2)\)th Fib. number satisfies 
\[
F(k + 2) \geq \varphi^k,
\]
where \( \varphi = \frac{(1 + \sqrt{5})}{2} = 1.61803\ldots \).

\[ \varphi^2 = \varphi + 1 \]

Fibonacci Numbers grow at least exponentially fast in \( k \).

Proof by induction on \( k \):

- **Base** \( k = 0 \): \( F(2) = 1 \) and \( \varphi^0 = 1 \) \( \checkmark \)
- **Base** \( k = 1 \): \( F(3) = 2 \) and \( \varphi^1 \approx 1.619 < 2 \) \( \checkmark \)
- **Inductive Step** \( (k \geq 2) \):

\[
F(k + 2) = F(k + 1) + F(k) \\
\geq \varphi^{k-1} + \varphi^{k-2} \quad \text{(by the inductive hypothesis)} \\
= \varphi^{k-2} \cdot (\varphi + 1) \\
= \varphi^{k-2} \cdot \varphi^2 \\
= \varphi^{k} \quad (\varphi^2 = \varphi + 1)
\]
Lemma 19.3

For all integers $k \geq 0$, the $(k + 2)$nd Fib. number satisfies $F(k + 2) \geq \varphi^k$, where $\varphi = \frac{1 + \sqrt{5}}{2} = 1.61803 \ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in $k$.

Proof by induction on $k$:

- **Base $k = 0$**: $F(2) = 1$ and $\varphi^0 = 1 \checkmark$
- **Base $k = 1$**: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2 \checkmark$
- **Inductive Step ($k \geq 2$):**

$$F(k + 2) = F(k + 1) + F(k) \geq \varphi^{k-1} + \varphi^{k-2} \quad \text{(by the inductive hypothesis)}$$

$$= \varphi^{k-2} \cdot (\varphi + 1)$$

$$= \varphi^{k-2} \cdot \varphi^2$$

$$= \varphi^k \quad \text{(}\varphi^2 = \varphi + 1\text{)}$$

$\square$
Putting the Pieces Together

Amortized Analysis

- **INSERT**: amortized cost $O(1)$
- **EXTRACT-MIN**: amortized cost $O(d(n))$
- **DECREASE-KEY**: amortized cost $O(1)$
Amortized Analysis

- **INSERT**: amortized cost $\mathcal{O}(1)$
- **EXTRACT-MIN**: amortized cost $\mathcal{O}(d(n))$
- **DECREASE-KEY**: amortized cost $\mathcal{O}(1)$

$N(k)$
Amortized Analysis

- **INSERT**: amortized cost $O(1)$
- **EXTRACT-MIN**: amortized cost $O(d(n))$
- **DECREASE-KEY**: amortized cost $O(1)$

\[ N(k) = F(k + 2) \]
Amortized Analysis

- **INSERT**: amortized cost $\mathcal{O}(1)$
- **EXTRACT-MIN**: amortized cost $\mathcal{O}(d(n))$
- **DECREASE-KEY**: amortized cost $\mathcal{O}(1)$

\[ N(k) = F(k + 2) \geq \varphi^k \]
Amortized Analysis

- **INSERT**: amortized cost $O(1)$
- **EXTRACT-MIN**: amortized cost $O(d(n))$
- **DECREASE-KEY**: amortized cost $O(1)$

\[ n \geq N(k) = F(k + 2) \geq \varphi^k \]
Putting the Pieces Together

Amortized Analysis

- **INSERT**: amortized cost $O(1)$
- **EXTRACT-MIN**: amortized cost $O(d(n))$
- **DECREASE-KEY**: amortized cost $O(1)$

\[
\begin{align*}
 n & \geq N(k) = F(k + 2) \geq \varphi^k \\
 \Rightarrow \quad \log_\varphi n & \geq k
\end{align*}
\]
Putting the Pieces Together

**Amortized Analysis**

- **INSERT**: amortized cost \( \mathcal{O}(1) \)
- **EXTRACT-MIN**: amortized cost \( \mathcal{O}(d(n)) \) \( \mathcal{O}(\log n) \)
- **DECREASE-KEY**: amortized cost \( \mathcal{O}(1) \)

\[
\begin{align*}
n \geq N(k) &= F(k + 2) \geq \varphi^k \\
\Rightarrow \quad \log_\varphi n \geq k
\end{align*}
\]
What if we don’t have marked nodes?

- **INSERT:** actual $\mathcal{O}(1)$
- **EXTRACT-MIN:** actual $\mathcal{O}(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $\mathcal{O}(1)$
What if we don’t have marked nodes?

- **INSERT:** actual $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $O(1)$

$$\Phi(H) = \text{trees}(H)$$
What if we don’t have marked nodes?

- **INSERT**: actual $O(1)$
- **EXTRACT-MIN**: actual $O(trees(H) + d(n))$
- **DECREASE-KEY**: actual $O(1)$

\[ \Phi(H) = trees(H) \]
What if we don’t have marked nodes?

- **INSERT:** actual $\mathcal{O}(1)$
- **EXTRACT-MIN:** actual $\mathcal{O}(\text{trees}(H) + d(n))$
- **DECREASE-KEY:** actual $\mathcal{O}(1)$

\[
\Phi(H) = \text{trees}(H)
\]
What if we don’t have marked nodes?

- **INSERT:** actual $O(1)$  
  amortized $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$  
  amortized $O(d(n))$
- **DECREASE-KEY:** actual $O(1)$  
  amortized $O(1)$

$$\Phi(H) = \text{trees}(H)$$
What if we don’t have marked nodes?

- **INSERT:** actual $O(1)$ amortized $O(1)$
- **EXTRACT-MIN:** actual $O(\text{trees}(H) + d(n))$ amortized $O(d(n)) \neq O(\log n)$
- **DECREASE-KEY:** actual $O(1)$ amortized $O(1)$

$$\Phi(H) = \text{trees}(H)$$
## Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. 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>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$O(n)$</td>
<td>$O(1)$</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>UNION</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</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>
</tbody>
</table>

5.2: Fibonacci Heaps (Analysis)
### Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. 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>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</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>
</tbody>
</table>

- **Can we perform **EXTRACT-MIN** in $o(\log n)$?**

- **DELETE = DECREASE-KEY + EXTRACT-MIN**

- Crucial for many applications including shortest paths and minimum spanning trees!
# Summary

If this was possible, then there would be a sorting algorithm with runtime $o(n \log n)$!

Can we perform `EXTRACT-MIN` in $o(\log n)$?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibonacci heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

5.2: Fibonacci Heaps (Analysis)

T.S.
### Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-HEAP</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>INSERT</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>EXTRACT-MIN</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
</tr>
<tr>
<td>UNION</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>DELETE</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
<td>$\mathcal{O}(\log n)$</td>
</tr>
</tbody>
</table>

5.2: Fibonacci Heaps (Analysis)
**Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAKE-HEAP</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>UNION</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

**DELETE = DECREASE-KEY + EXTRACT-MIN**

**5.2: Fibonacci Heaps (Analysis)**

Crucial for many applications including shortest paths and minimum spanning trees!
## Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. 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>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$O(n)$</td>
<td>$O(1)$</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>UNION</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</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>
</tbody>
</table>

**DELETE** = **DECREASE-KEY** + **EXTRACT-MIN**

**EXTRACT-MIN** = **MIN** + **DELETE**

---

**5.2: Fibonacci Heaps (Analysis)**

---

**Crucial for many applications including shortest paths and minimum spanning trees!**

If we could perform **EXTRACT-MIN** in $O(\log n)$, then there would be a sorting algorithm with runtime $O(n \log n)$!
<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked list</th>
<th>Binary heap</th>
<th>Binomial heap</th>
<th>Fibon. 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>INSERT</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$O(n)$</td>
<td>$O(1)$</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>UNION</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DECREASE-KEY</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</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>
</tbody>
</table>

Crucial for many applications including shortest paths and minimum spanning trees!
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

**Strict Fibonacci Heap:**
- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC’12)

**Strict Fibonacci Heap:**
- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but **actual costs**!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP’15)

- Queries to **marked bits** are intercepted and responded with a random bit
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC’12)

Strict Fibonacci Heap:
- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP’15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap and the number of operations
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC’12)

**Strict Fibonacci Heap:**
- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP’15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap and the number of operations
  \[\Rightarrow\] less efficient than the original Fibonacci heap
Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC’12)

Strict Fibonacci Heap:
- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP’15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap and the number of operations
  ⇒ less efficient than the original Fibonacci heap
  ⇒ marked bit is not redundant!
Outlook: A More Efficient Priority Queue for fixed Universe

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fibonacci heap amortized cost</th>
<th>Van Emde Boas Tree actual cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MERGE/UNION</strong></td>
<td>$O(1)$</td>
<td>-</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>SUCC</strong></td>
<td>-</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>PRED</strong></td>
<td>-</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MAXIMUM</strong></td>
<td>-</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
**Outlook: A More Efficient Priority Queue for fixed Universe**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fibonacci heap</th>
<th>Van Emde Boas Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>amortized cost</td>
<td>actual cost</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MINIMUM</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>EXTRACT-MIN</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MERGE/UNION</strong></td>
<td>$O(1)$</td>
<td>-</td>
</tr>
<tr>
<td><strong>DECREASE-KEY</strong></td>
<td>$O(1)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>$O(\log n)$</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>SUCC</strong></td>
<td>-</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>PRED</strong></td>
<td>-</td>
<td>$O(\log \log u)$</td>
</tr>
<tr>
<td><strong>MAXIMUM</strong></td>
<td>-</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

all this requires key values to be in a universe of size $u$!