5.2 Fibonacci Heaps (Analysis)

Frank Stajano  Thomas Sauerwald

Lent 2015
Outline

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 $\mathcal{O}(\# \text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

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

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

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

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

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

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

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

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

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

Lifecycle of a node

5.2: Fibonacci Heaps (Analysis)
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 $O(1)$ ?

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

Glimpse at the Analysis

Amortized Analysis

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

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

**Change in Potential**

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

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

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

**Φ(H) = trees(H) + 2 \cdot 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

- $\text{trees}(H') = \text{trees}(H) + x$
Amortized Analysis of **DECREASE-KEY**

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

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

**Potential Function**

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

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

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**
  - $\tilde{c}_i = c_i + \Delta \Phi$

**Actual Cost**

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

**Change in Potential**

**Amortized Cost**
Amortized Analysis of \textsc{Decrease-Key}

\begin{itemize}
  \item \textsc{Decrease-Key}: $O(x + 1)$, where $x$ is the number of cuts.
\end{itemize}

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

\begin{itemize}
  \item \text{trees}(H') = \text{trees}(H) + x
  \item \text{marks}(H') \leq \text{marks}(H) - x + 2
  \Rightarrow \Delta \Phi \leq x + 2 \cdot (-x + 2) = 4 - x.
\end{itemize}

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

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

**Φ(H)**

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

**Change in Potential**
- trees($H'$) = trees($H$) + $x$
- marks($H'$) ≤ 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 DECREASE-KEY

Actual Cost

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

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

First Coin \(\sim\) pays cut
Second Coin \(\sim\) increase of \(\text{trees}(H)\)

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

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

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

Actual Cost
Amortized Analysis of \textsc{Extract-Min}

Actual Cost

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

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

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

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

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

- Change in Potential
  - marks($H'$) \? 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**
  - $\text{marks}(H') \neq \text{marks}(H)$

---

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

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

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

**Potential Function**
\[
\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)
\]
Amortized Analysis of \textsc{Extract-Min}

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

\begin{align*}
\Phi(H) &= \text{trees}(H) + 2 \cdot \text{marks}(H)
\end{align*}

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

### Actual Cost

- **EXTRACT-MIN**: $O(trees(H) + d(n))$

### Change in Potential

- $marks(H') \leq marks(H)$
- $trees(H') \leq$

### Potential Function

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$
Amortized Analysis of EXTRACT-MIN

Actual Cost

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

Change in Potential

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

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

Change in Potential

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

$d(n)$

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

5.2: Fibonacci Heaps (Analysis) T.S. 6
Amortized Analysis of \textsc{Extract-Min}

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

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

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

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

5.2: Fibonacci Heaps (Analysis)
Amortized Analysis of EXTRACT-MIN

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

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

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

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

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

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

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

Φ($H$) = trees($H$) + 2 · marks($H$)

Change in Potential
- marks($H'$) ≤ marks($H$)
- trees($H'$) ≤ $d(n)$ + 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 $d(n)$?
Glimpse at the Analysis

Amortized Analysis

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

Binomial Heap
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 \( \Rightarrow d(n) \leq \log_2 n \).

\[ d = 3, \quad n = 2^3 \]
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 $d(n) \leq \log_\phi n$, where $\phi \approx 1.62$. 

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

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.

\[
\forall 1 \leq i \leq k: d_i \geq i - 2
\]
Lower Bounding Degrees of Children

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

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

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

- 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 of a tree with nodes $y_1, y_2, y_3$ and a root $x$.]
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.

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.

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

```
  x
 / \  /
/   / /
/   /  
  y_1 y_2 y_3 y_4
```
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.

![Diagram of a tree structure with nodes labeled $x$, $y_1$, $y_2$, $y_3$, $y_4$, $\ldots$, $y_k$. The node $y_2$ has a cross mark indicating a special node.]

---

5.2: Fibonacci Heaps (Analysis)
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
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 and $d_1, d_2, \ldots, d_k$ be their degrees
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 and $d_1, d_2, \ldots, d_k$ be their degrees.

\[ \forall 1 \leq i \leq k: \quad d_i \geq i - 2 \]
∀ 1 ≤ i ≤ k: \( d_i \geq i - 2 \)
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 $k$. 

\[ \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \]
∀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)
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 \( k \).

\( N(0) \)

- 0
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 $k$.

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

Definition

$N(0) = 0$  
$N(1) = 1$  
$N(2) = 2$  
$N(3) = 3$  
$N(4) = 5$  
$N(5) = 8$  
$N(k) = F(k+2)$

5.2: Fibonacci Heaps (Analysis)
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 \( k \).

\[
\begin{array}{ccc}
N(0) & N(1) \\
\bullet 0 & \bullet 1 \\
\end{array}
\]
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 \( k \).

\[
\begin{align*}
N(0) & = 0 \\
N(1) & = 1 \\
N(2) & = 5
\end{align*}
\]
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 \( k \).

\[
\begin{array}{cccc}
N(0) & N(1) & N(2) \\
\bullet 0 & \bullet 1 & \bullet 2 \\
\end{array}
\]

...
From Degrees to Minimum Subtree Sizes

∀ \hspace{1em} 1 \leq i \leq k : \hspace{1em} 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 \( k \).

\[
\begin{array}{ccc}
N(0) & N(1) & N(2) \\
\bullet \hspace{1em} 0 & \bullet \hspace{1em} 1 & \bullet \hspace{1em} 2 \\
\hspace{1em} \bullet \hspace{1em} 0 & \bullet \hspace{1em} 0 & \bullet \hspace{1em} 0 \\
\end{array}
\]
\[ \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \]

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

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

**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

∀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 \( k \).

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

**Definition**

Let \(N(k)\) be the minimum possible number of nodes of a subtree rooted at a node of degree \(k\).
\begin{align*}
\forall 1 \leq i \leq k: \quad d_i \geq i - 2
\end{align*}

**Definition**

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

![Diagram of tree with nodes and edges]

- $N(0) = 0$
- $N(1) = 1$
- $N(2) = 2$
- $N(3) = 3$
- $N(4) = 5$ 

$N(k) = F(k+2)$
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 \( k \).

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

\[
\begin{array}{c|c|c|c|c}
 k & N(0) & N(1) & N(2) & N(3) & N(4) \\
0 & 0 & 1 & 2 & 3 & 4 \\
1 & 0 & 1 & 2 & 3 & 4 \\
2 & 0 & 0 & 1 & 1 & 2 \\
3 & 0 & 0 & 1 & 1 & 2 \\
4 & 0 & 0 & 1 & 1 & 2 \\
\end{array}
\]
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 \( k \).

<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>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( y_1 )</td>
<td>( y_2 )</td>
<td>( y_3 )</td>
<td>( y_4 )</td>
<td>( y_k )</td>
</tr>
</tbody>
</table>

\( d_i \geq i - 2 \) for all \( 1 \leq i \leq k \).
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 \( k \).

\[
\begin{align*}
N(0) &= 1 & N(1) &= 1 & N(2) &= 2 & N(3) &= 3 & N(4) &= 4 \\
0 & & 1 & & 2 & & 3 & & 4 \\
0 & & 0 & & 0 & & 0 & & 0 \\
0 & & 1 & & 1 & & 0 & & 0 \\
0 & & 0 & & 0 & & 0 & & 0 \\
0 & & 0 & & 0 & & 0 & & 0
\end{align*}
\]

5.2: Fibonacci Heaps (Analysis)
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 \( k \).

\[
N(0) = 1 \quad N(1) = 2 \quad N(2) = 3 \\
N(3) = 5 + 3 \quad N(4) = F(k+2)
\]
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 \( k \).

\[
\begin{align*}
N(0) &= 1 \\
N(1) &= 2 \\
N(2) &= 3 \\
N(3) &= 5 \\
N(4) &= 8
\end{align*}
\]
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 \( k \).

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

∀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 $k$.

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

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

**Definition**

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

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

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

**Definition**

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

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

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

∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

Definition

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

\[
\begin{align*}
N(0) &= 1 & N(1) &= 2 & N(2) &= 3 & N(3) &= 5 \\
N(4) &= 8 = 5 + 3
\end{align*}
\]
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 \( k \).

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

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

\( N(k) = F(k + 2) \)?
\( \forall 1 \leq i \leq k : \quad d_i \geq i - 2 \)

\( N(k) = F(k + 2) ? \)
∀1 ≤ i ≤ k: \( d_i \geq i - 2 \)

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

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

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

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

\[ = F(k + 1) + F(k) = F(k + 2) \]
Lemma 19.3

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

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

$\phi^2 = \phi + 1$

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

$\phi$ is essentially the golden ratio, which is the solution to the equation $x^2 = x + 1$. This equation is derived from the Fibonacci sequence's recurrence relation and represents the ratio of successive Fibonacci numbers as $k$ tends to infinity.
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 \( \varphi = (1 + \sqrt{5})/2 = 1.61803... \).

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

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

Proof by induction on \( k \):
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 \( \varphi^0 = 1 \)
Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$th Fibonacci 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 $\varphi^0 = 1$.
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 $\varphi^0 = 1 \checkmark$
- **Base $k = 1$:** $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in $k$. 
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 $\varphi^0 = 1 \checkmark$
- **Base $k = 1$:** $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2 \checkmark$
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 $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

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 ($k \geq 2$):

  $$F(k + 2) =$$

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

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

\[
F(k+2) = F(k+1) + F(k) \\
\geq \varphi^{k-1} + \varphi^{k-2} \quad \text{(by the inductive hypothesis)} \\
= \varphi^{k-2} \cdot (\varphi + 1)
\]
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 $\varphi = (1 + \sqrt{5})/2 = 1.61803\ldots$.

Proof by induction on $k$:
- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1 \sqrt$
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2 \sqrt$
- 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 \((k+2)\)th 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 (\( k \geq 2 \)):

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

\( \square \)
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 $O(1)$
- **EXTRACT-MIN** amortized cost $O(d(n))$
- **DECREASE-KEY** amortized cost $O(1)$

$$N(k)$$
Amortized Analysis

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

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

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

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

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

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

**Amortized Analysis**

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

\[
 n \geq N(k) = F(k + 2) \geq \phi^k \\
\Rightarrow \quad \log_{\phi} n \geq k
\]
Amortized Analysis

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

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

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

$$\Phi(H) = \text{trees}(H)$$
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 $O(d(n)) \neq O(\log n)$
- **DECREASE-KEY:** actual $O(1)$ amortized $O(1)$

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

![Fibonacci Heap Diagram](image)
## 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>

$\text{DELETE = DECREASE-KEY + EXTRACT-MIN}$

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

Can we perform **EXTRACT-MIN** in $O(\log n)$? If this was possible, then there would be a sorting algorithm with runtime $O(n \log n)$!
### 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>$O(\log n)$</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>

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

- **Can we perform **Extract-Min** in $o(\log n)$?**
### 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>$O(\log n)$</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>

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

If this was possible, 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>

**Summary**

---

5.2: Fibonacci Heaps (Analysis)  
T.S.  
15
## 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>

\[
\text{DELETE} = \text{DECREASE-KEY} + \text{EXTRACT-MIN}
\]

**EXTRACT-MIN = MIN + DELETE**

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

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

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

If this was possible, then there would be a sorting algorithm with runtime $O(n \log n)$!
## 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>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td><strong>INSERT</strong></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><strong>MINIMUM</strong></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><strong>EXTRACT-MIN</strong></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><strong>UNION</strong></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><strong>DECREASE-KEY</strong></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><strong>DELETE</strong></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>

\[ \text{DELETE} = \text{DECREASE-KEY} + \text{EXTRACT-MIN} \]

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**

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

If this was possible, then there would be a sorting algorithm with runtime $O(n \log n)$!
Recent Studies of Fibonacci Heaps

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

- 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 of Fibonacci Heaps

- 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, (Arxive:1407.2569, 2014)

- Queries to marked bits are intercepted and responded with a random bit
Recent Studies of Fibonacci Heaps

- 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, (Arxive:1407.2569, 2014)

- 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 of Fibonacci Heaps

- 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, (Arxive:1407.2569, 2014)

- 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
Recent Studies of Fibonacci Heaps

- 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, (Arxiv:1407.2569, 2014)

- 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>$O(\log \log u)$</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 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>

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