O(log n)

often called heapify

Fix Downward(i)

Fig 25.4

Requirements:
- Only violation of Heap Ordered between node i and its children
- Continue here

possible violations

select larger child and swap i with that child
Fix Upward (i)

Fig 25.3

Requires: Only possible violation of Heap Ordered

i is between i + its parent
Overview of binary heap

Advantages - very space efficient, very simple, low constants, hidden in asymptotic notation

Drawbacks

Priority queue - doesn't maintain full sorted order, linear time search

Binary heap - merging takes linear time, increasing priority (through tracker) take log time
Merging (Creating) Priority Queues

Given arbitrary array $a[0], \ldots, a[n-1]$ that I want to turn into heap

successive insertions $O(n \log n)$

For \( \text{int } i = n-1; i > = 0; i-- \) \n\n\text{Fix Downward}(i)

$O(n)$ time
## Overview of Priority Queue Data Structures

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary Heap</th>
<th>Leftist Heap</th>
<th>Pairing Heap</th>
<th>Fibonacci Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>extractMax</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>max</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>add</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>merge</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>increase priority</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>decrease priority</td>
<td></td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

- **Key**: Amortized
- **Time Complexity**:
  - Constant: O(1)
  - Logarithmic: O(log n)
  - Linear: O(n)
Quiz

1. Describe 2 applications for graphs — give problems what are vertices what are edges 4 pts each

2. Do you have any question about the video 2 pts
Types of Graphs

- weighted & unweighted
- directed & undirected
- graph & multigraph
  - allows multiple edges between same two vertices
Finding Shortest Travel Routes

Each airport is a vertex.

Sample edges (flight) - multiple edges between vertices.

Multigraph - 

#edges = #flights
How do we represent a graph?

- **Vertices**: \( V \)
- **Edges**: \( E \)
- **List**: \( \text{List} \)
- **Set**: \( \text{Set} \)
- **Tagged Bucket Collection**: \( \langle V, \text{collection} \langle E \rangle \rangle \)

**Adjacency List**

- **Tag**
- **Vertex**
- **Edges**
- **Set of Edges**
Adj List

a: b, c
b: a, c, d
c: a, b, d
d: b, c

Instead

a: e₁, e₂
b: e₁, e₃, e₅
c: e₂, e₃, e₄
d: e₄, e₅
Set of Vertices

Each vertex mapping to bucket of edges

Adjacency Set \) allow expected constant time search for an edge

Vertex \rightarrow Set of outgoing Edges
What do we do for an undirected graph with adjacency list?

\textbf{a}: e_1, e_3, e_2
\textbf{b}: e_1
\textbf{c}: e_3, e_4
\textbf{d}: e_4, e_2
Alternate Representation

Adjacency Matrix

\[
\begin{array}{cccc}
\text{a} & \leftarrow & \text{b} & \leftarrow \\
\downarrow^{e_2} & \downarrow^{e_3} & \downarrow^{e_4} \\
\text{c} & \rightarrow & \text{d} & \rightarrow \\
\end{array}
\]

\[
\begin{pmatrix}
\text{a} & e_1 & \text{null} \\
\text{b} & e_2 & e_3 \\
\text{c} & \text{null} & \text{null} \\
\text{d} & \text{null} & \text{null} \\
\end{pmatrix}
\]
adjacency matrix (only really useful here)

dense graph - most edges exist

exchange rates

sparse graph - lots of edges don't exist

precedence graph (often very sparse)

metro

WWW (web)