Priority Queue ADT

important methods

add
max - returns max element
extractMax - removes max + returns

break ties arbitrarily

Contains will take linear time through

remove
update (change priority)

increase priority
decrease priority
Data Structure
Binary Heap

Structure
\[ 0 \]
\[ n=1 \]

Rep.

Property

HEAP ORDERED
the priority of each node is as great as that of its descendants
Array Representation

If they exist:

\[
\begin{align*}
\text{parent}(i) &= \left\lfloor \frac{i-1}{2} \right\rfloor \\
\text{left}(i) &= 2i + 1 \\
\text{right}(i) &= 2i + 2
\end{align*}
\]

heap

size

Space between n references

n + 2n references
Inserting an element

1. add new element to next open slot in the array

2. Swap new element with parent until the parent is at least as large or reach root

$O(\log n)$ time
Finding and Extracting (Removing) Max $O(\log n)$

1. Remember root (store in a variable)
2. Replace root by last element
   \[ \text{heap}[0] = \text{heap}[n-1] \]
   \[ n--; \]
3. Look at two children
   * If \text{HEAPORDERED} is violated, swap with larger child

Diagram:
- Tree structure with nodes labeled a, i, o, r, c, b, and n.
- Arrows indicating parent-child relationships and subtree relationships.
- Annotations indicating steps for finding and extracting the max.

- Max in root (element 0)
- Continue for other subtrees.
Increase Priority

```
tracker t = p.q.addTracked (r);
    t.IncreasePriority (z);
```

change element

swap with parent until its HEOGRAPHED is restored

$O(\log n)$
Fix Upward (i)

Fig 25.3

Requires:
- Only possible violation of Heap Ordered
- i is between i and its parent

swap

continue
Decrease Priority

tracker t

tracker t.decreasePriority(b)

Like extract Max on the subtree root at the node being changed
Fix Downward(i)  

\(O(\log n)\)  

often called heapify  

Fig 25.4  

Possible violations  

select larger child  

swap i with that child  

Requires:  
Only Violation of HeapOrdered between node i and its children  
}

continue here
Deletion

heap 100, 99, 10, 28, 97, 9, 8, 26, 95, 94, 93, 76...

Ex where $\text{heap}[n-1]$ is larger than what it replaces

100

99
98 97
96 95 10
96 95 94 93 76 54

95

If replacement is larger
fix upward
otherwise fix downward
Overview of binary heap

Advantage - very space efficient
very simple with low constants hidden in asymptotic notation

Drawbacks
merging two binary heaps takes linear time
increasing priority (through tracker) takes logarithmic time
Merging two priority queues.

Give an arbitrary array $a[0]...a[n-1]$. Convert to a binary heap in linear time:

For (int $i=n-1$, $i\geq 0$; $i--$)

FixDownward($i$)

$O(n)$

Successive inserts $O(n \log n)$
### Overview of Priority Queue Data Structures

<table>
<thead>
<tr>
<th></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>Extract Max</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
- **Complexity**:
  - **Constant**: ❋
  - **Logarithmic**: ❋
  - **Linear**: ❋
Tracked Binary Heap

heaps

Swap $a+x$