Priority Queues
Examples

- Casualty
- Discrete Event Simulation
- Shortest Paths
- Coding Algorithms
- Compression Algorithms
- Optimisation Algorithms
- Games
Priority Queue ADT

- `first()` - get the smallest key-value (but leave it there)
- `insert()` - add a new key-value
- `extractMin()` - remove the smallest key-value
- `decreaseKey()` - reduce the key of a node
- `merge()` - merge two queues together
Priority Queue ADT

Key: insert, extractMin.

Best performance expected?

Imagine doing sorting using a P.Q.

⇒ n inserts, n extractMin()

⇒ O(n) ops of P.Q. operations

Best sorting O(n lg n)

∴ insert, extractMin must be at least O(lg n)
Sorted Array Implementation

- Put everything into an array
- Keep the array sorted by sorting after every operation

- `first()` → $O(1)$ lookup
- `insert()` → $O(n)$
- `extractMin()` → $O(n)$
- `decreaseKey()` → $O(n)$
- `merge()` → $O(n)$
RB Tree Implementation

- first() $\in O(\log n)$
- insert() $\in O(\log n)$
- extractMin() $\in O(\log n)$
- decreaseKey() $\in O(\log n)$
- merge() — insert all of one tree into the other $\in O(n \log n)$
Binary Heap Implementation

- Could use a **min-heap** (like the max-heap we saw for heapsort)

```
   a
  / \
 q   q > a
```

- `insert()`
  - Add to bottom
  - "Bubble" up
  - Worst case $\Rightarrow O(\text{levels})$  
    $= O(\log n)$

- `first()`  
  $O(1) \Rightarrow \text{Lookup}$
Binary Heap Implementation

- `extractMin()`: Analogous to heapsort.
  - Extract
  - Fix
  - \( O(\lg n) \)

- `decreaseKey()`
  - Make change
  - Bubble
  - \( O(\lg n) \)

- `merge()`: \( O(n \lg n) \)
Limitations of the Binary Heap

- It's common to want to merge two priority queues together
- With a binary heap this is costly...
Binomial Heap Implementation

- First define a binomial **tree**
  - Order 0 is a single node
  - Order k is made by merging two binomial trees of order \(k-1\) such that the smaller root becomes the new root

Image courtesy of wikipedia
Merging Trees

- Note that the definition means that two trees of order $X$ are trivially made into one tree of order $X+1$.

![Diagram of merging trees with order 2 and 2, resulting in order 3, with an $O(1)$ process notation.](image)
How Many Nodes in a Binomial Tree?

Hence:

Because we combine two trees of the same order to make the next order tree, we double the number of nodes when we increase the order.
Binomial Heap Implementation

- Binomial **heap**
- A set of binomial trees where every node is **smaller** than its children
- And there is at **most** one tree of each order attached to the root

Image courtesy of wikipedia
Binomial Heap Implementation
Binomial Heaps as Priority Queues

- `first()`
  - The minimum node in each tree is the tree root so the heap minimum is the smallest root

\[ O(\text{no of roots}) \]
How many roots in a binomial heap?

- For a heap with \( n \) nodes, how many root (or trees) do we expect?
- Because there are \( 2^k \) nodes in a tree of order \( k \), the binary representation of \( n \) tells us which trees are present in a heap. *E.g. 10101*

![Diagram of a binary tree structure]

- The biggest tree present will be of order \( \log n \), which corresponds to the \( \lfloor \log n \rfloor + 1 \)-th bit
  - So there can be no more than \( \lfloor \log n \rfloor + 1 \) roots
- \( \text{first()} \) is \( O(\text{no. of roots}) = O(\log n) \)
Merging two heaps is useful for the other priority queue operations.

First, link together the tree heads in increasing tree order.
Merging Heaps

Now check for duplicated tree orders and merge if necessary.
Merging Heaps: Analogy

- This process is actually analogous to binary addition!

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Merging Heaps: Costs

- Let $H_1$ be a heap with $n$ nodes and $H_2$ a heap with $m$ nodes

\[
\{ H_1 + H_2 \} \rightarrow (n+m) \text{ nodes}
\]

\[
H_1 + H_2 \leq \frac{\log_2 n + \log_2 m + 2}{\text{No of roots to merge}}
\]

merge $\Rightarrow O(1)$

$O(\log n)$
Priority Queue Operations

- **insert()**
  - Just create a zero-order tree and merge! \(O(\lg n)\)

- **extractMin()**
  - Splice out the tree with the minimum
  - Form a new heap from the 2\(^{nd}\) level of that tree
  - merge the resulting heap with the original

![Diagram of tree operations]
Priority Queue Operations

- `decreaseKey()`
  - Change the key value
  - Let it 'bubble' up to its new place
  - $O(\text{height of tree})$

```
+-------------------------------+      +-------------------------------+      +-------------------------------+
| O                             | +      | O                             | +      | O                             |
|                               | 2      |                               | 2      |                               |
+-------------------------------+      +-------------------------------+      +-------------------------------+
```

```
adding to height
```

$O(\log n)$
Priority Queue Operations

- `deleteKey()`
  - Decrease node value to be the minimum
  - Call `extractMin()` (!)

- $\mathcal{O}(\log n)$
- $\mathcal{O}(\log n)$

- $\mathcal{O}(\log n)$
Recap...

- **Sorting**
  - Bubble, (Binary) Insertion, Selection, Merge, Quick, Heap.

- **Algorithm Design**
  - Greedy, Brute force, Backtracking, Divide and Conquer, Dynamic Programming

- **Data Structures**
  - Stack, Queue, Deque
  - Tables: R-B Trees, AVL Trees, B-Trees, Hash Tables
  - Priority Queue: Binary heap, Binomial Heap