Merging heaps

- Binary Heap has limited (fast) functionality
  - FindMin, DeleteMin and Insert only
  - does not support fast merges of two heaps
- For some applications, the items arrive in prioritized clumps, rather than individually
- Is there somewhere in the heap design that we can give up a little performance so that we can gain faster merge capability?

Worst Case Run Times

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary Heap</th>
<th>Binomial Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>Θ(log N)</td>
<td>Θ(log N)</td>
</tr>
<tr>
<td>FindMin</td>
<td>Θ(1)</td>
<td>Θ(log N)</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>Θ(log N)</td>
<td>Θ(log N)</td>
</tr>
<tr>
<td>Merge</td>
<td>Θ(N)</td>
<td>O(log N)</td>
</tr>
</tbody>
</table>
Binomial Queues

- Binomial queues give up $O(1)$ FindMin performance in order to provide $O(\log N)$ merge performance.
- A **binomial queue** is a collection (or forest) of heap-ordered trees:
  - *Not just one tree, but a collection of trees!*
  - Each tree has a defined structure and capacity
  - Each tree has the familiar heap-order property

---

Structure Property

- Each tree contains two copies of the previous tree:
  - the second copy is attached at the root of the first copy
- The number of nodes in a tree of depth $d$ is exactly $2^d$

<table>
<thead>
<tr>
<th>depth</th>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>depth</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

- Any number $N$ can be represented in base 2: $\sum_{i=0}^{i=n-1} a_i 2^i$
  - A base 2 value identifies the powers of 2 that are to be included

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>Decimal$_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
</tbody>
</table>
Numbers of nodes

• Any number of entries in the binomial queue can be stored in a forest of binomial trees
• Each tree holds the number of nodes appropriate to its depth, i.e., $2^d$ nodes
• So the structure of a forest of binomial trees can be characterized with a single binary number
  $101_2 \rightarrow 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$ nodes

What is a merge?

• There is a direct correlation between
  › the number of nodes in the tree
  › the representation of that number in base 2
  › and the actual structure of the tree
• When we merge two queues of sizes $N_1$ and $N_2$, the number of nodes in the new queue is the sum of $N_1 + N_2$
• We can use that fact to help see how fast merges can be accomplished

Structure Examples

<table>
<thead>
<tr>
<th>$N = 2_{10} = 10_2$</th>
<th>$2^2 = 4$</th>
<th>$2^1 = 2$</th>
<th>$2^0 = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N = 4_{10} = 100_2$</th>
<th>$2^2 = 4$</th>
<th>$2^1 = 2$</th>
<th>$2^0 = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N = 3_{10} = 11_2$</th>
<th>$2^2 = 4$</th>
<th>$2^1 = 2$</th>
<th>$2^0 = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N = 5_{10} = 101_2$</th>
<th>$2^2 = 4$</th>
<th>$2^1 = 2$</th>
<th>$2^0 = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BQ.1

Example 1.

Merge BQ.1 and BQ.2

Easy Case.

There are no comparisons and there is no restructuring.
Example 2.

Merge BQ.1 and BQ.2

This is an add with a carry out.

It is accomplished with one comparison and one pointer change: O(1)

Example 3.

Merge BQ.1 and BQ.2

Part 1 - Form the carry.

Part 2 - Add the existing values and the carry.

---

**Merge Algorithm**

- Just like binary addition algorithm
- Assume trees $X_0, \ldots, X_n$ and $Y_0, \ldots, Y_n$ are binomial queues
  - $X_i$ and $Y_i$ are of type $B_i$ or null

\[
C_0 := \text{null}; \\
\text{for } i = 0 \text{ to } n \text{ do} \\
\quad \text{combine } X_i, Y_i, \text{ and } C_i \text{ to form } Z_i \text{ and new } C_{i+1} \\
\quad Z_{n+1} := C_{n+1}
\]
**Exercise**

- Create a single node queue $B_0$ with the new item and merge with existing queue
- $O(\log N)$ time

**O(\log N) time to Merge**

- For $N$ keys there are at most $\lceil \log_2 N \rceil$ trees in a binomial forest.
- Each merge operation only looks at the root of each tree.
- Total time to merge is $O(\log N)$.

**Insert**

**DeleteMin**

1. Assume we have a binomial forest $X_0, \ldots, X_m$
2. Find tree $X_k$ with the smallest root
3. Remove $X_k$ from the queue
4. Remove root of $X_k$ (return this value)
   - This yields a binomial forest $Y_0, Y_1, \ldots, Y_{k-1}$.
5. Merge this new queue with remainder of the original (from step 3)
- Total time = $O(\log N)$
Implementation

- Binomial forest as an array of multiway trees
  ‣ FirstChild, Sibling pointers

DeleteMin Example

FindMin

Remove min

Return this

New forest

Old forest

Merge
Why Binomial?

\[ \binom{d}{k} = \frac{d!}{(d-k)k!} \]

<table>
<thead>
<tr>
<th>tree depth (d)</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes at depth (k)</td>
<td>1, 4, 6, 4, 1</td>
<td>1, 3, 3, 1</td>
<td>1, 2, 1</td>
<td>1, 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Other Priority Queues

- **Leftist Heaps**
  - O(log N) time for insert, deletemin, merge
  - The idea is to have the left part of the heap be long and the right part short, and to perform most operations on the left part.

- **Skew Heaps** ("splaying leftist heaps")
  - O(log N) amortized time for insert, deletemin, merge

Exercise Solution