Comparing Heaps

<table>
<thead>
<tr>
<th>Worst case [avg/amort. case]</th>
<th>Binary Heap</th>
<th>Leftist Heap</th>
<th>Skew Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(n) [O(\log n)]</td>
</tr>
<tr>
<td>findMin</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>deleteMin</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(n) [O(\log n)]</td>
</tr>
<tr>
<td>merge</td>
<td>O(n)</td>
<td>O(\log n)</td>
<td>O(n) [O(\log n)]</td>
</tr>
<tr>
<td>build</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

Our Last Priority Queue

In exchange for getting fast merges with leftist and skew heaps, we have lost the fast (average case) inserts of binary heaps.

We’ll look at one more priority queue data structure that doesn’t make this trade-off: binomial queues.

But first, a brief diversion into binomial trees...
Binomial Trees

- A binomial tree $B_h$ has height $h$ and exactly $2^h$ nodes.
- $B_h$ is formed by making $B_{h-1}$ a child of another $B_{h-1}$.

<table>
<thead>
<tr>
<th>Height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Binomial Trees

- The root has exactly $h$ children.
- Every subtree of a binomial tree is also a binomial tree.

<table>
<thead>
<tr>
<th>Height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Binomial Trees

Number of nodes at depth $d$ is a binomial coeff.:  
\[
\binom{h}{d} = \frac{h!}{(h-d)!d!}
\]

Pascal’s triangle:

<table>
<thead>
<tr>
<th>Height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Binomial Queues

- Structural property
  - Forest of binomial trees with at most one tree of any height

  What’s a forest?

- Order property
  - Each binomial tree has the heap-order property
Binomial Queue with $n$ elements

Binomial Q with $n$ elements has a unique structural representation in terms of binomial trees!

Write $n$ in binary: $n = 1101_{(base\ 2)} = 13_{(base\ 10)}$

<table>
<thead>
<tr>
<th>height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Properties of Binomial Queue

- At most one binomial tree of any height.
- $n$ nodes
  $\Rightarrow$ # of bits in binary representation: $O(\log n)$
  $\Rightarrow$ number of trees: $\leq \lceil \log n \rceil + 1 \in O(\log n)$
  $\Rightarrow$ deepest tree has height: $O(\log n)$
- Is each subtree a binomial queue? Yes.

A Few More Examples
How to merge queues?

- 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, the number of nodes in the new queue is the sum of \( N_1 + N_2 \)
- We can use these facts to help see how fast merges can be accomplished
- E.g., add 3+3 in base 2 arithmetic:

\[
\begin{array}{c}
\text{3} \\
\text{3} \\
\end{array}
\begin{array}{c}
\text{+} \\
\text{+} \\
\end{array}
\begin{array}{c}
\text{1} \\
\text{1} \\
\end{array}
\begin{array}{c}
\text{0} \\
\end{array}
\]

**Example 1.**

Merge BQ.1 and BQ.2

**Easy Case.**

There are no comparisons and there is no restructuring.

<table>
<thead>
<tr>
<th>BQ.1</th>
<th>BQ.2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>BQ.1</th>
<th>BQ.2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>
Example 3.

Merge BQ.1 and BQ.2

Part 1 – Add $B_0$ trees, form the carry.

\[
\begin{array}{c|c|c|c}
N-10_{10} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
BQ.1 & 1 & 3 & 7 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
N-10_{10} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
+ BQ.2 & 4 & 6 & 8 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
= carry & 7 & 3 & 8 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
N-10_{10} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

Example 3.

Part 2 - Add the existing $B_1$ trees and the carry.

\[
\begin{array}{c|c|c|c}
N-10_{10} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
+ BQ.2 & 4 & 6 & 8 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
= BQ.3 & 7 & 3 & 8 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
N-10_{10} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

Merge Algorithm

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

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

Complexity of Merge

Constant time for each tree.

Max number of trees is: $O(\log n)$

$\Rightarrow$ worst case running time is $O(\log n)$
**Insert**

- Create a single node queue $B_0$ with the new item and merge with existing queue.

- Total time (worst case) = $O(\log n)$

- Total time (average case) = $O(1)$
  - Hint: Think of adding 1 to 1101

**DeleteMin**

1. Assume we have a binomial queue $X_0, ..., X_m$
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 queue $Y_0, Y_1, ..., Y_{k-1}$
5. Merge this new queue with remainder of the original (from step 3)

- Total time (worst case) = $O(\log n)$
More Operations on Binomial Queue

- `buildBinomialQ` can be done with repeated inserts in $O(n)$ time.

- Can we do `decreaseKey` efficiently? `increaseKey`?

- What about `findMin`?

---

Comparing Heaps

<table>
<thead>
<tr>
<th>Worst case [avg/amort. case]</th>
<th>Binary Heap</th>
<th>Leftist Heap</th>
<th>Skew Heap</th>
<th>Binomial Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>$O(\log n)$ [ $O(1)$ ]</td>
<td>$O(\log n)$ [ $O(\log n)$ ]</td>
<td>$O(n)$ [ $O(\log n)$ ]</td>
<td>$O(\log n)$ [ $O(1)$ ]</td>
</tr>
<tr>
<td>findMin</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>deleteMin</td>
<td>$O(\log n)$ [ $O(\log n)$ ]</td>
<td>$O(\log n)$ [ $O(\log n)$ ]</td>
<td>$O(n)$ [ $O(\log n)$ ]</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>merge</td>
<td>$O(n)$</td>
<td>$O(\log n)$ [ $O(\log n)$ ]</td>
<td>$O(n)$ [ $O(\log n)$ ]</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>build</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>