## Pairing Heaps

<table>
<thead>
<tr>
<th></th>
<th>Fibonacci</th>
<th>Pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove min (or max)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Meld</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Decrease key (or increase)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
# Pairing Heaps

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fibonacci Complexity</th>
<th>Pairing Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Remove min (or max)</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Meld</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Remove</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Decrease key (or increase)</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Pairing Heaps

- Experimental results suggest that pairing heaps are actually faster than Fibonacci heaps.
  - Simpler to implement.
  - Smaller runtime overheads.
  - Less space per node.
Definition

- A min (max) pairing heap is a min (max) tree in which operations are done in a specified manner.
Node Structure

• Child
  ▪ Pointer to first node of children list.

• Left and Right Sibling
  ▪ Used for doubly linked linked list (not circular) of siblings.
  ▪ Left pointer of first node is to parent.
  ▪ $x$ is first node in list iff $x.left.child = x$.

• Data

• Note: No Parent, Degree, or ChildCut fields.
Meld – Max Pairing Heap

- Compare-Link Operation
  - Compare roots.
  - Tree with smaller root becomes leftmost subtree.

- Actual cost = $O(1)$.
Insert

- Create 1-element max tree with new item and meld with existing max pairing heap.

\[
\begin{align*}
\begin{array}{c}
9 \\
7 \\
6 \\
7 \\
\end{array} & + \text{ insert}(2) & = & \begin{array}{c}
9 \\
2 \\
7 \\
6 \\
\end{array} \\
\begin{array}{c}
6 \\
3 \\
\end{array} & & \begin{array}{c}
6 \\
3 \\
\end{array}
\end{align*}
\]
Insert

• Create 1-element max tree with new item and meld with existing max pairing heap.

\[ + \text{ insert}(14) = \]

• Actual cost = \(O(1)\).
Worst-Case Degree

• Insert 9, 8, 7, …, 1, in this order.

• Worst-case degree = n – 1.
Worst-Case Height

• Insert 1, 2, 3, …, n, in this order.

• Worst-case height = n.
IncreaseKey(theNode, theAmount)

- Since nodes do not have parent fields, we cannot easily check whether the key in theNode becomes larger than that in its parent.
- So, detach theNode from sibling doubly-linked list and meld.
IncreaseKey(theNode, theAmount)

If theNode is not the root, remove subtree rooted at theNode from its sibling list.
IncreaseKey(theNode, theAmount)

Meld subtree with remaining tree.
IncreaseKey(theNode, theAmount)

• Actual cost = O(1).
Remove Max

- If empty => fail.
- Otherwise, remove tree root and meld subtrees into a single max tree.
- How to meld subtrees?
  - Good way => $O(\log n)$ amortized complexity for remove max.
  - Bad way => $O(n)$ amortized complexity.
Bad Way To Meld Subtrees

• currentTree = first subtree.
• for (each of the remaining trees)
  currentTree = compareLink(currentTree, nextTree);
Example

• Remove max.

  8 6 4 2 1 3 5 7

• Meld into one tree.

  7 5 3 1 2 4 6
Example

- Actual cost of insert is 1.
- Actual cost of remove max is degree of root.
- n/2 inserts (9, 7, 5, 3, 1, 2, 4, 6, 8) followed by n/2 remove maxs.
  - Cost of inserts is n/2.
  - Cost of remove maxs is $1 + 2 + \ldots + \frac{n}{2} - 1 = \Theta(n^2)$.
  - If amortized cost of an insert is $O(1)$, amortized cost of a remove max must be $\Theta(n)$. 
Good Ways To Meld Subtrees

- Two-pass scheme.
- Multipass scheme.
- Both have same asymptotic complexity.
- Two-pass scheme gives better observed performance.
Two-Pass Scheme

• Pass 1.
  ▪ Examine subtrees from left to right.
  ▪ Meld pairs of subtrees, reducing the number of subtrees to half the original number.
  ▪ If # subtrees was odd, meld remaining original subtree with last newly generated subtree.

• Pass 2.
  ▪ Start with rightmost subtree of Pass 1. Call this the working tree.
  ▪ Meld remaining subtrees, one at a time, from right to left, into the working tree.
Two-Pass Scheme – Example

Pass 1

T1 → S1 → T2
T3 → S2 → T4
T5 → S3 → T6
T7 → S4 → T8
Two-Pass Scheme – Example

Pass 2

S1 → S2 → S3 → S4

R1 → R2 → R3
Multipass Scheme

• Place the subtrees into a FIFO queue.
• Repeat until 1 tree remains.
  ▪ Remove 2 subtrees from the queue.
  ▪ Meld them.
  ▪ Put the resulting tree onto the queue.
Multipass Scheme--Example
Multipass Scheme--Example

- Actual cost = $O(n)$.
Remove Nonroot Element

- Remove theNode from its sibling list.
- Meld children of theNode using either 2-pass or multipass scheme.
- Meld resulting tree with what’s left of original tree.
Remove(node)

Remove theNode from its doubly-linked sibling list.
Remove(theNode)

Meld children of theNode.
Remove(theNode)

Meld with what’s left of original tree.
Remove(theNode)

Actual cost = $O(n)$.