Priority Queue: data structure for set of items from an ordered universe (more precisely their keys are ordered)

item = (name, pointer, key)
supports operations

basic
insert (item)
extractmin - returns and deletes item having smallest key

odd'n operations
merge (Q₁, Q₂)
forms union & Q₁, U Q₂
(destroy Q₁, Q₂)
delete (x) x = name of item
findmin - returns item with least key
decreasekey (x, Δ)
\[ x_{\text{key}} \leftarrow x_{\text{key}} - \Delta \]
more generally, \( x_{\text{key}} \) gets smaller value
Leftist Trees first implementation of meldable priority queues - i.e. supports merging.

Implemented as heap-ordered binary trees

Each node stores one item, and heap order condition is maintained:

- Key of item in child = key of item in parent
- Minimum found in root node

Tree structure satisfies leftist condition:

- Shortest path from any node to a null pointer proceeds rightward. Above tree diagram not leftist

- is leftist
Merging two leftist trees:

\[
\begin{array}{c}
\text{Step 1: Merge the ordered lists of } a_i \text{'s and } b_i \text{'s into single ordered list and form tree with rightward path respecting combined order.}
\\
e.g., a_1 < a_2 < b_1 < a_3 < b_2 \ldots
\end{array}
\]
Possible violations are confined to nodes on right path from root.

Suppose subtrees $U$ and $V$ satisfy leftist property.

Leftist property is destroyed if not satisfied. Can be restored by swapping $U$ and $V$.

**Step 2** We maintain in each node an additional field (right height) = shortest distance to null link.

Start at node on right path furthest from root and do fix-up steps at each node:

- $r_i < \frac{1}{2}$
- $s_k = \min(r_1, r_2) + 1$
work involved in merging = $O(mn)$

also = $\Theta(mn)$

within constant factors of $mn$

**Extractmin**

1. remove $x$

**Insert(x)**

2. merge $(A, B)$

**merge**($x$, $\triangle$)

**Analysis**

Claim: For a tree with $n$ nodes, right height $\leq \log_2(n+1)$
\( N = \# \text{ nodes in tree} \)

\( \text{r.h.} = \text{right height} \)

Number of nodes \( \geq 15 \)
\[ = 2^{(r.h+1)} - 1 \]

\[
\text{r.h.} \leq \log_2 (n+1) - 1
\]

to merge trees of sizes \( m, n \)

involves work = \( O(\text{r.h.} + \text{r.h.}^2) \)

\[ = O(\log_2 m + \log_2 n) \]

\[ = O(\log_2 (m+n)) \]

\[ = O(\log(\max(m,n))) \]

\[ \log_2 (m+n) \leq \log (2 \cdot \max(m,n)) \]
Generally operation cost =

\[ O \left( \log \left( \text{resulting tree size} \right) \right) \]

Running time - overall

sequence of operations \( q_1, q_2, \ldots \)

\( s_i = \text{size of R.E. upon completion of } q_i \).

\( (x) \) cost of sequence = \( O \left( \sum \log s_i \right) \)

Can we dispense with right height fields and still preserve \( (x) \)

\[ \text{Steep} \]

\[ \begin{array}{c}
A \\
B \\
C \\
\vdots \\
D \\
\end{array} \]
A. Do no swaps
B. Swap at every node along right path

Strategy B
approach is referred to as a skew heap

To establish that skew heaps provide
same performance as leftist trees, we develop
amortized analysis.

What does it accomplish?

Given sequence of operations

$O_1, O_2, O_3, \ldots$

with costs $c_i = \text{cost}(O_i)$, let

$a_1, a_2, \ldots$ be a sequence that satisfies
Amortized Analysis for Skew Heap

$m = \text{number of nodes in subtree}$

$n = \text{number of nodes in parent subtree}$ if $m/n \geq 1/2$

the link from parent to child is heavy

light if not heavy

1. A heavy link has a light sibling link
2. For any path starting at root, the number of light links $\leq \log_2(\text{tree size})$

$n' + n'' < n/2$ when link is light

Potential defined to be number of right heavy links
**Intuition**

Expensive operations involve long rightward paths contribute significantly to the potential.

Suppose trees being merged have right paths of length $k_1$ and $k_2$.

$$T_i \quad p \quad T_2$$

$$P_1 = k_1 - \log |T_1|$$

$$P_2 = k_2 - \log |T_2|$$

$$P = k_1 + k_2 - \log |T_1| - \log |T_2|$$

Since each off-path right link that's heavy accounts for an on-path
\[ \Delta P = P_{\text{After}} - P_{\text{Before}} \leq 3 \log n - k_1 - k_2 \]
\[ t + \Delta P = (k_1 + k_2) + \Delta P \leq 3 \log n + t \]

**Fibonacci Heaps**

All standard B.E. operations supported

- *findmin*: returns item with least key
- *insert*
- *merge*
- *deletion*
- *decrease-key* (more efficient than delete + re-insert)

Amortized costs:
- \( \log n \) for deletion
- Constant for the other operations.