Self-Adjusting Heaps

- No explicit structure. Adjust the structure in a simple, uniform way, so that the efficiency of future operations is improved.

Amortized Time Complexity

- Total time for operations / number of operations.
Example: Amortized Complexity

Let S be an array (with \(n + 1\) elements) and \(\text{top}\) be an nonnegative integer. We will use S and \(\text{top}\) to represent a stack. Initially \(\text{top} = 0\). There are three operations on the stack: Push, Pop and Multipop. These operations are defined as follows:

**Push (x)**
- \(\text{top}++\);
- \(S[\text{top}] = x\);
- end Push;

**Pop (x)**
- if (\(\text{top} == 0\)) then return;
- print \(S[\text{top}]\);
- \(\text{top}--\);
- return;
- end Pop;
Multipop (k)
    for i = 1 to k do
        {if (top == 0) then return;
         print S[top];
         top--;}
    return
end Multipop;

What is the worst case time complexity for Push(x), Pop(x), and Multipop(k)?

Executing any sequence of $n$ operations of the form Push(x), Pop(x), and Multipop(k) takes time equal to $n$ times the worst time complexity of executing any of the above three operations. Is the bound best possible (i.e., is it tight)?
Comparison

- **Worst Case TC**: Insert $O(x)$ and Delete $O(y)$: Every time the algorithm is run each Insert operation takes $O(x)$ and each Delete operation takes $O(y)$.

- **Average Case TC**: Insert $O(x)$ and Delete $O(y)$: When the algorithm is run over a set of inputs with a given frequency count the Insert operation takes on average $O(x)$ and the Delete operation takes on average $O(y)$.

- **Amortized TC**: Insert $O(x)$ and Delete $O(y)$: Every time the algorithm is run the Insert operation takes on average $O(x)$ and the Delete operation takes on average $O(y)$.
Mergeable Heap

- ADT defined over a totally ordered universe. Operations are:
  - **Make heap**\((h)\): Create a new, empty heap, named \(h\).
  - **Find Min**\((h)\): Return the min item in heap \(h\). If \(h\) is empty then return the special item called “null”.
  - **Insert**\((x, h)\): Insert item \(x\) in heap \(h\), not previously containing it.
  - **Delete min**\((h)\): Delete the minimum item from heap \(h\), and return it. If the heap is initially empty then return “null”.
  - **Meld**\((h_1, h_2)\): Return the heap formed by taking the union of disjoint heaps \(h_1\) and \(h_2\). This operation destroys \(h_1\) and \(h_2\).
Heap-Ordered Binary Tree (Skew Heaps)

Binary tree whose nodes are items.

Tree is arranged in a heap order, if $p(x)$ is the parent of $x$, then the item stored at $p(x)$ is less than the item stored at $x$.

Implementation of Operations

- *Make Heap*: $O(1)$ time by just setting the root of $h$ to null.
- *Find Min($h$)*: Return the item stored in the root of $h$.
- *Insert($x$, $h$)*: Make $x$ a single node heap and meld it with $h$.
- *Delete min($h$)*: Delete the root and replace $h$ with the meld of its left and right
\textbf{Meld}(h_1, h_2)

- Form a single tree by traversing the right paths of \(h_1\) and \(h_2\), merging them into a single right path with items in increasing order.

- The left subtrees of nodes along the **merge path** do not change.

- Swap the left and right children of every node on the merge path except at the lowest level.
MELD Algorithm

Procedure meld(val $h_1, h_2$)
   if $h_2 = \text{null}$ then return $h_1$
   else return $\text{xmeld}(h_1, h_2)$;
end

Procedure $\text{xmeld}(val \ h_1, h_2$)
   // $h_2$ is not null //
   if $h_1 = \text{null}$ then return $h_2$;
   if $\text{item}(h_1) > \text{item}(h_2)$ then $h_1 \leftarrow h_2$;
   ( lchild($h_1$), rchild($h_1$) ) $\leftarrow$
      ( $\text{xmeld}(\text{rchild}(h_1), h_2$), lchild($h_1$) );
   return $h_1$
End of Procedure
Definitions

- $S$: Collection of Skew Heaps.
- $\Phi(S)$: Potential of $S$.
- $m$ operations with times $t_1, t_2, ..., t_m$.
- $a_i$ amortized time for operation $i$.
- $\Phi_i$: Potential after operation $i$.
- $\Phi_0$: Initial potential.
- $\sum t_i = \sum (a_i - \Phi_i + \Phi_{i-1}) = \Phi_0 - \Phi_m + \sum a_i$
- $\Phi_0$ is initially zero.
- $\Phi_i$ is non-negative.
Idea

- High Potential: Remaining operations may be expensive.
- Low Potential: Remaining operations are inexpensive.
- Amortized bound: $O(\log n)$ time per operation.

Definitions

- $wt(x)$: Number of descendants of $x$ (incl. $x$).
- Non-root $x$ is heavy if $wt(x) > wt(p(x))/2$.
- Non-root $x$ is light otherwise.
- Node $x$ is right if it is a right child.
- Node $x$ is left if it is a left child.
**Results**

**Lemma 1:** Of the children of any node, at most one is heavy.

**Lemma 2:** On any path from node $x$ down to a descendant $y$, there are at most $\lfloor \log (\text{wt}(x)/\text{wt}(y)) \rfloor$ light nodes, not counting $x$. In particular, any path in an $n$-node tree contains at most $\lfloor \log n \rfloor$ light nodes.

**Proof:** If there are $k$ light nodes not including $x$ along the path from $x$ to $y$, then

$$\text{wt}(y) \leq \frac{\text{wt}(x)}{2^k} \Rightarrow$$

$$k \leq \log (\text{wt}(x)/\text{wt}(y)).$$

**Potential of a Skew Heap:** Total number of right heavy nodes in it.
**Definitions**

- Let $n_1$ and $n_2$ be the number of nodes in $h_1$ and $h_2$, resp.

- Number of light nodes on the right path of $h_1$ ($h_2$) is at most $\lfloor \log n_1 \rfloor$ ($\lfloor \log n_2 \rfloor$).

- Let $k_1$ and $k_2$ be the number of heavy nodes on the right path of $h_1$ and $h_2$, resp.

- Let $k_3$ be the number of new right heavy nodes in the resulting heap. Clearly $k_3 \leq \lfloor \log n \rfloor$. 
Bounds

- Number of nodes on the merge path is at most
  \[ 2 + \lfloor \log n_1 \rfloor + k_1 + \lfloor \log n_2 \rfloor + k_2 \leq 1 + 2\lfloor \log n \rfloor + k_1 + k_2 \]

- Increase in potential because of the meld is
  \[ k_3 - k_1 - k_2 \leq \lfloor \log n \rfloor - k_1 - k_2 \]

- Amortized cost is \( 3\lfloor \log n \rfloor + 1 \).
Operation May Take $\Omega(n)$ Time

- Insert
  
  $n, n+1, n-1, n+2, n-2, n+3, \ldots, 1, 2n$.
  
- For $n = 5$, the sequence is 5, 6, 4, 7, 3, 8, 2, 9, 1, 10.
  
- Then Delete-min.

\[
\begin{align*}
15 & \rightarrow 16 \\
5 & \rightarrow 5 \\
6 & \rightarrow 5 \\
14 & \rightarrow 4 \\
5 & \rightarrow 6 \\
4 & \rightarrow 17 \\
5 & \rightarrow 5 \\
6 & \rightarrow 7 \\
4 & \rightarrow 7 \\
5 & \rightarrow 5 \\
6 & \rightarrow 6 \\
4 & \rightarrow 3 \\
5 & \rightarrow 4 \\
6 & \rightarrow 7 \\
3 & \rightarrow 4 \\
5 & \rightarrow 7 \\
5 & \rightarrow 6
\end{align*}
\]
Heaps-15

$\begin{array}{c}
\text{18} \\
\text{3} \\
\text{4} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6} \\
\rightarrow \\
\text{3} \\
\text{4} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6} \\
\text{12} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6} \\
\rightarrow \\
\text{2} \\
\text{3} \\
\text{9} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6} \\
\text{19} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6} \\
\rightarrow \\
\text{2} \\
\text{3} \\
\text{9} \\
\text{8} \\
\text{7} \\
\text{5} \\
\text{6}
\end{array}$