Recall binomial heaps.

Left unfinished: why is Build \( \Theta(n) \)
rather than \( O(n \log n) \)

Bit cost of incrementing a binary counter from 0 to \( n \):
worst case cost of single increment on \( k \) bit counter is \( \Theta(k) \)

\[ \begin{array}{c}
+ & 10111 \\
\hline
\end{array} \]
\( k = 5 \)

\[ \begin{array}{c}
11000 \\
\hline
\end{array} \]
Cost is 4

\( \max \) # bits

So \( n \) increments cost \( O(n \log n) \)

Better bound. As we increment,
- the 2\(^{st}\) bit (least significant bit) flips every time
- the 2\(^{st}\) bit flips every other time \( \left( \frac{n}{2} \right) \) times
- the 2\(^{st}\) bit flips every 2\(^i\) times \( \left( \frac{n}{2^i} \right) \) times

Total cost

\[ \sum_{i=0}^{k} \left( \frac{n}{2^i} \right) \leq n \sum_{i=0}^{k} \frac{1}{2^i} \]

\[ < n \sum_{i=0}^{k} \frac{1}{2^i} = 2n \]

This applies to creating a binomial heap of \( n \) items
- cost is \( \Theta(n) \)
Definition: sequence of \( m \) operations

Total cost is \( T(m) \)

Then the amortized cost per operation is \( \frac{T(m)}{m} \) (average)

Potential Method for amortized analysis

Idea: accounting trick

Potential = savings in bank account

[true] cost

[artificial] charge

if charge > cost — extra goes in bank
charge < cost — extra comes out of bank.

\( \Phi_i \) — potential after \( i \)th operation.

\( \Phi_i = \Phi_{i-1} + \text{charge}(i) - \text{cost}(i) \)

Potential & charge are artificial — we define one and then the other is determined

Could define potential & get charge:

\( \text{charge}(i) = \text{cost}(i) + \Phi_i - \Phi_{i-1} \)
Thm. If final potential $\geq$ initial potential
then amortized cost $\leq$ max charge.

Pf. \[
\frac{1}{m} \sum_{i=1}^{m} \text{charge}(i) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(i) + \frac{1}{m} \sum_{i=0}^{m-1} \Phi_i \]
\[= \sum_{i=1}^{m} \text{cost}(i) + \Phi_m - \Phi_0 \geq 0 \]
\[\leq \frac{1}{m} \sum_{i=1}^{m} \text{charge}(i) \leq \text{max charge}\]

To do potential analysis
- invent potential/charge s.t.
  \[\Phi_m \geq \Phi_0\] (the bank is never "in the red")
  and max charge is small.
Revisit incrementing binary counter using potential method:

1. Cost is high when many 1's $\rightarrow$ 0's
   - Only one bit changes 0 $\rightarrow$ 1

2. Pay for this $\uparrow$ and extra $\$1$ for when this 1 $\rightarrow$ 0.
   - Every change 1 $\rightarrow$ 0 is paid from potential.

3. Charge $(i)$ = $2 - 1$ for 0 $\rightarrow$ 1 change
   - 1 for future 1 $\rightarrow$ 0 change

By this, amortized cost $\leq$ max charge = 2

So long as $\Phi_m \geq \Phi_0$

What is potential?

<table>
<thead>
<tr>
<th>Cost</th>
<th>Charge</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

By observation (and we could prove this formally)

Potential $\Phi_i = \# 1$'s in counter after $i$th op

So $\Phi_0 = 0$ $\Phi_m \geq 0$ so this applies
Alternatively, we can define $\Phi_i$ = # 1's in counter
Then it is obvious that $\Phi_i \geq \Phi_0$.
And we need to prove that charge = 2
(since amortized cost $\leq 2$ by thm)

Suppose $i$th op. changes $t_i$ bits $1 \rightarrow 0$
\[ \begin{array}{c|c}
011111 & t_i \\
100000 & \end{array} 
\]
\[ \text{1 bit: } 0 \rightarrow 1 \]

\[ \text{cost}(i) = t_i + 1 \]

$\Phi_i = \Phi_{i-1} - t_i + 1$

since $\Phi_i$ = # 1's in counter.

\[ \text{charge}(i) = \text{cost}(i) + \Phi_i - \Phi_{i-1} \text{ by def} \]

\[ = t_i + 1 - t_i + 1 = 2 \]
Mergeable Heaps
- main operation is merge.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binomial Heap</th>
<th>Lazy Bin. Heaps</th>
<th>Fibonacci Heaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>( \log(n) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>deleteMin</td>
<td>( \log(n) )</td>
<td>( \Theta(\log n) )</td>
<td>( A \ O(\log n) )</td>
</tr>
<tr>
<td>merge</td>
<td>( \log(n) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>decreaseKey</td>
<td>( \log(n) )</td>
<td>( \Theta(\log n) )</td>
<td>( A \ O(1) )</td>
</tr>
<tr>
<td>build</td>
<td>( \Theta(n) )</td>
<td>( O(n) )</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>

\( A = \text{amortized} \).

Lazy Binomial Heaps
How do we improve merge and insert — be lazy
- don't bother combining trees, just allow
  multiple trees of same rank.

Catch up on work when do a deleteMin
- combine so that we only have one tree of each rank.
Deletemin
- look at all roots to find min
- delete that root - its children become separate trees
- consolidate
  for rank = 1 to max rank (≤ O(log n))
    while there are ≥ 2 trees of this rank
      link 2 to one tree

recall: rank = degree of root
        = height of tree.

Worst case cost of Deletemin Θ(n)
because could have n singleton trees

Thm Lazy Binomial heaps have 0(log n)
amortized cost for Deletemin, O(1) for Insert, Merge.

Pf. (magic) \( \Phi_i = \# \text{trees} \) (why? It works!)
\( \Phi_0 = 0 \quad \Phi_m ≥ 0 \)

So \( \Phi_m ≥ \Phi_0 \)

formula for charge(i) = cost(i) + \( \Phi_i - \Phi_{i-1} \)

let's look at other ops. first:
merge cost is O(1), #trees - same
Insert cost is O(1), #trees increases by 1
DeleteMin
\[ r = \text{degree of node containing } \min \leq O(\log n) \]
\[ t = \# \text{ trees before DeleteMin} = \Phi_{i-1} \]
Consolidate is called on:
\[ t-1 + r \text{ trees} \]
\[ \text{cost is } \frac{t-1+r}{\text{each link decreases } \# \text{ trees}} + O(\log n) \]
\[ \text{loop } l \text{ max rank } \leq O(\log n) \]
\[ \text{& keep track of } \# \text{ trees of each rank}. \]
\[ \Phi_i \leq O(\log n) \text{ because we end up with a Binomial heap which has } O(\log n) \text{ trees.} \]
Amortized cost \[ \leq \max \text{ charge } \leq \text{cost}(i) + \Phi_i - \Phi_{i-1} \]
Thin
\[ = (t-1+r + O(\log n) - t) \leq r + O(\log n) \]
\[ \leq O(\log n) \]
So DeleteMin - \( O(n) \) worst case
but \( O(\log n) \) amortized.
Fibonacci heaps - improve lazy binomial trees to reduce DecreaseKey to $O(1)$ amortized (from $O(\log n)$)
Recall DecreaseKey used bubble-up $O(\text{height of tree})$
Instead, cut the tree

\[ \text{e.g.} \]

\begin{align*}
\text{dec. to 1} \quad 12 & \rightarrow \\
8 & 6 \quad 1 \\
10 & 12 \\
13 & \quad \\
13 & \quad \\
\end{align*}

Danger: # trees increases
# nodes per tree changes (not just $2^i$)
So must do cuts carefully to keep $\text{Delete Min}$ at $O(\log n)$

Reason for name: no. of nodes in a Fibonacci tree of rank $k$ is $\geq (k+2)^{th}$ Fibonacci number $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

This gives rank $\in O(\log n)$ because $f_k \in \Theta(\phi^k)$

See [CLRS] for more details.