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 $\Theta(k)$

\[ \begin{array}{c}
+ & 1 & 0 & 1 & 1 & 1 \\
& & & & & \\
\hline
1 & 1 & 0 & 0 & 0 \\
\end{array} \]

\[ k = 5 \]

Cost is 4

So $n$ increments cost $\Theta(n \log n)$

Better bound. As we increment,
- the $2^0$th bit (least significant bit) flips every time
- the $2^1$st bit flips every other time
- the $2^i$th bit flips every $2^i$ times

Total cost
\[
\sum_{i=0}^{k} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \sum_{i=0}^{k} \frac{1}{2^i}
\]

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

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 $T(m)/m$ (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:

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

Pf. $\sum_{i=1}^{m} \text{charge}(i) = \sum_{i=1}^{m} \text{cost}(i) + \sum_{i=0}^{m-1} \Phi_i$

= $\sum_{i=1}^{m} \text{cost}(i) + \Phi_m - \Phi_0$

$\sum_{i=1}^{m} \text{charge}(i) \geq \sum_{i=1}^{m} \text{cost}(i) \geq 0$

amortized cost = $\frac{\sum_{i=1}^{m} \text{cost}(i)}{m} \leq \frac{\sum_{i=1}^{m} \text{charge}(i)}{m} \leq \text{max charge}$

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.
Revist incrementing binary counter using potential method.

Cost is high when many 1's -> 0's
only one bit changes 0->1

Pay for this and extra $1 for when this 1 -> 0.
Every change 1->0 is paid from potential.

charge \( i \) = 2 - 1 for 0->1 change
1 for future 1->0 change

By thm amortized cost \( \leq \max \text{charge} = 2 \)
so long as \( \Phi_m \geq \Phi_0 \)

What is potential?

<table>
<thead>
<tr>
<th></th>
<th>cost</th>
<th>charge</th>
<th>potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>2</td>
<td>2</td>
<td>( \frac{1}{2} )</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>( \frac{1}{2} )</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

by observation (will prove soon):
potential \( \Phi_i = \# \text{1's in counter after ith op} \)
so \( \Phi_0 = 0 \) \( \Phi_m \geq 0 \) so thm applies
to be more formal, verify relation between charge and \( \Phi \).

Suppose \( i \)th op. changes \( t_i \) bits \( 1 \rightarrow 0 \)

\[
\begin{array}{c}
\text{1 bit} \\
0 \rightarrow 1 \\
\hline
100000
\end{array}
\]

\[ t_i \]

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

\[ \Phi_i = \Phi_{i-1} - t_i + 1 \quad \text{using } \Phi_i = \#1's \text{ in counter}. \]

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

\[ = t_i + 1 - t_i + 1 = 2 \]

So, yes, \( \text{charge} = 2 \).
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>A O(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>O(log n)</td>
<td>A O(1)</td>
</tr>
<tr>
<td>build</td>
<td>O(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 allowmultiple trees of same rank.

Catch up on work when do a Deletemin
- combine so that we only have one tree of each rank.

theoretical improvement
for MST
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 \(\Theta(n)\)
because could have \(n\) singleton trees

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

pf: (magic) \(\Phi = \#\) trees (why? It works!)
\(\Phi_0 = 0\), \(\Phi_m \geq 0\)

So \(\Phi_m \geq \Phi_0\).

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

\[ r = \text{degree of node containing the min} \in O(\log n) \]

\[ t = \# \text{trees before DeleteMin} \]

\[ = \Phi_{i-1} \]

consolidate is called on:

\[ t - 1 + r \quad \text{trees} \]

\[ \text{cost is} \leq \frac{t - 1 + r}{\text{each link decreases} \# \text{trees}} + O(\log n) \]

\[ \text{loop } l \quad \max \text{rank} \in O(\log n) \]

\[ \& \quad \text{keep track of} \# \text{trees of each rank} \]

\[ \Phi_i \in O(\log n) \quad \text{because we end up with a Binomial heap which has } O(\log n) \text{ trees.} \]

amortized cost \( \leq \text{max charge} \leq \text{cost}(i) + \Phi_i - \Phi_{i-1} \)

thin

\[ \leq (t - 1 + r + O(\log n)) - (t) \leq r + O(\log n) \]

\[ \in O(\log n) \]

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 $O$ (height of tree)
Instead, cut the tree
e.g.

```
        7
       / \
      6   8
     / \
    10 12
   /   \
  13   9
```

```
        12
       / \
      8   6
     / \
    10 12
   /   \
  13   9
dec. to 1
```

Danger: # trees increases
# nodes per tree changes (not just $2^i$)
So must do cuts carefully to keep DeleteMin at $O(\log n)$
Reason for name: no. of nodes in a Fibonacci tree of
rank $k$ is $\geq (k+2)^{th}$ Fibonacci number $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

$f_{k+2}$

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

See [CLRS] for more details.