As always, sit with a partner and work through these together.

**Activity #1**: Fill in the missing values:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>add</th>
<th>removeMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Array</td>
<td></td>
<td>O(n)</td>
</tr>
<tr>
<td>Sorted Array</td>
<td></td>
<td>O(1)</td>
</tr>
<tr>
<td>Unsorted Linked List</td>
<td>O(1)</td>
<td></td>
</tr>
<tr>
<td>Sorted Linked List</td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>Hash Table</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Activity #2a**: Draw how inserting the node below into the heap would look keeping heap properties in mind:

```
  2
 / 
5   
/   /
6   7
/  /  \
11 12 8
```

**Activity #2b**: What is the runtime for insert()? Please explain.
Activity #3a Also keeping heap properties in mind, draw what you removeMin() will look like:

![Binary Tree Diagram]

Activity #3b What is the runtime for removeMin()? Please explain.

Activity 4: Improving Pseudocode

1. Make an improvement to this code
2. Briefly summarize the improvement in the space provided

function LCA(u, v):
    lca = null
    udepth = T.depth(u)
    vdepth = T.depth(v)
    if (T.isroot(u) == true) or (T.isroot(v) == true) then
        lca = T.root
    while (lca == null) do
        if (u == v) then
            lca = u
        else if udepth > vdepth then
            u = T.parent(u)
            udepth = udepth - 1
        else if vdepth > udepth
            v = T.parent(v)
            vdepth = vdepth - 1
        else
            u = T.parent(u); udepth = udepth - 1
            v = T.parent(v); vdepth = vdepth - 1
    return lca

Improvements: