Quiz 11

A. Heaps

A priority queue is one of the main uses. People do not use Heaps to search, though!

<table>
<thead>
<tr>
<th></th>
<th>search</th>
<th>insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>BST</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

We could liken a heap to a weakly ordered tree, or a partially ordered tree.

Two kinds: minheap and maxheap (book shows minheap)

Heap property: the parent is greater than either of its children, but no relationship exists between leftchild and rightchild. That is, no ordering between them.

Requires that we have a complete binary tree.

Demo of heapsort:


Exercise: do some percUp and percDown operations.

Heapcalc.py – a calculator (demo it)
0 levels $2^0$ nodes = 1
1 level $2^1$ nodes = 2
2 level $2^2$ nodes = 4

$2^k$ nodes at $k$ level

$\sum_{i=0}^{m} 2^i = 2^0 + 2^1 + 2^2 + \ldots + 2^{m-1} + 2^m$

$\Rightarrow 2^{m+1} - 1 = 2^{3+1} - 1$
$2^4 - 1 = 16 - 1$
$= 15$
B. AVL trees


The height of a tree – the maximum number of hops (lines) from the root to a leaf.

The trick is that you never let a tree get too far out of balance. The left and right subtrees of any node have a height that do not differ more than 1.

Algorithm to redistribute nodes in a BST such that the height of two subtrees is more or less equal. You pay extra time whenever you insert or delete. Searching is always $O(\log_2(N))$. So an AVL tree would be best if you didn't modify it often.

Demo:  https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

C. Trees in other uses

C1. File systems

Demo the pathnames in terminal window

C2. B-trees, B+ trees, B* trees

Database "files" are really bushy trees.

In reality there may be 25 or more keys in a single block. The insertion and deletion algorithms are complicated and time-consuming!
Below is a tree. Is it a minheap? If not, why not?

Following is a minheap in array format. Put all numbers into the tree picture below:
Here is the algorithm `percUp(i)`: (for a minheap)

```
while i divided by 2 is greater than 0
    see whether list[i] is smaller than list[i // 2]
    (that is, is the value at list[i] smaller than the value at its parent position?)
    if so, then exchange them
    now make i the parent of i
```

Here's the code:

```
while i // 2 > 0:
    if list[i] < list[i // 2]:
        list[i], list[i // 2] = list[i // 2], list[i]
    i = i // 2
```

Below is a minheap. Do a percUp on the last position (which holds 8 currently.) Cross out old values and write new values

![Minheap diagram]

Suppose the size is 8. Do `percUp(8)` (This means start at location 8) This will move 2 to the "top of the tree" which is position 1 in the array. Remember this is a minheap. Show the parents of nodes with arrows. Cross out old values and write the new values below.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>20</td>
<td>21</td>
<td>36</td>
<td>32</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
The size is now 15. Do percUp(15). Remember this is a minheap.

\[
\begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
4 & 7 & 5 & 8 & 10 & 6 & 12 & 18 & 19 & 14 & 9 & 13 & 7 & 50 & 3 \\
\end{array}
\]

\[3\uparrow 2 = 1\]  \[7\uparrow 2 = 3\]  \[15\uparrow 2 = 7\]

Do percUp(8) on the array version of the heap below. Cross out values as you exchange them.

\[
\begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
3 & 7 & 5 & 8 & 10 & 6 & 12 & 13 & & & & & & & \\
\end{array}
\]

\[8\uparrow 4 = 2\]

\[\text{no swap done!}\]

Take the following random list of 15 numbers and arrange them into a maxheap. This is where the parent's value is \textit{greater than} both children's values. There is more than right trecc!

\[5, 18, 70, 47, 16, 26, 92, 40, 69, 38, 6, 2, 90, 42, 13\]

\[
\begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
92 & 90 & 70 & 69 & 47 & 42 & 40 & 38 & 26 & 18 & 16 & 13 & 6 & 5 & 2 \\
\end{array}
\]
Below is percDown(i):

Keep going while i*2 is less than or equal to the size of the array
Look at the values of the two children of i and pick the smaller.
Let "minchild" be the index of the smaller of the two children
if the value at i is bigger than the value at the "minchild" position
then swap the two values
set i to be the "minchild" position

Here's the code for minheaps.

```python
def percDown(i):
    while i * 2 < size:
        if i * 2 + 1 > size:  # then there is only a left child, no right child
            minchild = i * 2
        else:
            if list[i*2] < list[i*2 + 1]:  # if left child value < right child value
                minchild = i * 2
            else:
                minchild = i * 2 + 1
        # we now know which of the two children has the smaller value
        if list[i] > list[minchild]:
            temp = list[i]
            list[i] = list[minchild]
            list[minchild] = temp
        i = minchild
```

Below is a minheap. Do percDown(1) which starts with 18 at the root of the tree and moves
down. Cross out old values and write in new ones as you do this.
Below is a heap (another minheap.) Do a percDown(1). This will move 18 to its proper level.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>9</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>8</td>
<td>9</td>
<td>20</td>
<td>19</td>
<td>13</td>
<td>15</td>
<td>16</td>
<td>14</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Below is an unordered list of numbers in an array. Notice it is not a heap. Draw arrows from each node to its two children. Label the left child arrow with L and the right child arrow with R. To keep the diagram clean, use both the top and the bottom.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>14</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>11</td>
<td>70</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Identify at least one node and its two children that violate the heap property. Do not choose 1.

nodes 2 & 5  \[ [2] = 4 \]  \[ [5] = 3 \]  \[ 4 > 3 \]

Below is a function that builds a heap out of an unordered list like this:

```python
def heapify(somelist):
    i = len(somelist) // 2
    while i > 0:
        percDown(i)
        i -= 1
```

Why does the algorithm start with \( i = \text{len}(\text{somelist}) // 2 \)? Why not \( i = \text{len}(\text{somelist}) \)? (Hint: think about tree children.)

Because none of these nodes have children. Suppose \( i > \text{len}/2 \). \( 2i > \text{len} \)

Perform percDown(3) on the following minheap. Draw arrows to indicate children, and cross out old values.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
<td>25</td>
<td>17</td>
<td>26</td>
<td>30</td>
<td>28</td>
<td>40</td>
<td>18</td>
<td>27</td>
<td>31</td>
<td>32</td>
<td>36</td>
<td>29</td>
<td>50</td>
</tr>
</tbody>
</table>

no change!
Perform percDown(1) on the following **maxheap**.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60</td>
<td>41</td>
<td>20</td>
<td>40</td>
<td>17</td>
<td>19</td>
<td>5</td>
<td>37</td>
<td>12</td>
<td>16</td>
<td>10</td>
<td>18</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Change 60 to 25

25 > 20

25 > 2, so no swap

25 > 3