Heaps

"You think you know when you can learn, are more sure when you can write, even more when you can teach, but certain when you can program."
- Alan Perlis

Priority Queue

- Recall priority queue
  - elements enqueued based on priority
  - dequeue removes the highest priority item
- Options?
  - List? Binary Search Tree? Clicker 1
  
<table>
<thead>
<tr>
<th>Linked List enqueue</th>
<th>BST enqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>O(N)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>O(logN)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>O(1)</td>
<td>O(logN)</td>
</tr>
</tbody>
</table>

Another Option

- A heap
  - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)
- A complete binary tree
  - all levels have maximum number of nodes except deepest where nodes are filled in from left to right
- Maintains the heap order property
  - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree

Clicker 2

- In a max heap with no duplicates where is the largest value?
  A. the root of the tree
  B. in the left-most node
  C. in the right-most node
  D. a node in the lowest level
  E. none of these
Example Min Heap

Enqueue Operation

- Add new element to next open spot in array
- Swap with parent if new value is less than parent
- Continue back up the tree as long as the new value is less than new parent node

Enqueue Example

- Add 15 to heap (initially next left most node)

Enqueue Example

- Swap 15 and 52
Enqueue Example

- Swap 15 and 17, then stop

```
12
  15
  17 37
  19 25
45 21 52
```

Internal Storage

- Interestingly heaps are often implemented with an array instead of nodes

```
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
12 17 15 19 52 37 25 45 21
```

Enqueue Example

- Insert the following values 1 at a time into a min heap:
  16 9 5 8 13 8 8 5 5 19 27 9 3

In Honor of Elijah, The Meme King, Spring 2020
**PriorityQueue Class**

```java
public class PriorityQueue<E extends Comparable<? super E>> {

    private int size;
    private E[] con;

    public PriorityQueue() {
        heap = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }

    public void enqueue(E val) {
        if ( size >= con.length - 1 )
            enlargeArray( con.length * 2 + 1 );

        size++;
        int indexToPlace = size;
        while ( indexToPlace > 1
            && val.compareTo( con[indexToPlace / 2] ) < 0 ) {
            con[indexToPlace] = con[indexToPlace / 2]; // swap
            indexToPlace /= 2; // change indexToPlace to parent
        }
        con[indexToPlace] = val;
    }

    private void enlargeArray(int newSize) {
        E[] temp = getArray(newSize);
        System.arraycopy(con, 1, temp, 1, size);
        con = temp;
    }
}
```

**PriorityQueue enqueue**

```java
public void enqueue(E val) {
    if ( size >= con.length - 1 )
        enlargeArray( con.length * 2 + 1 );

    size++;
    int indexToPlace = size;
    while ( indexToPlace > 1
        && val.compareTo( con[indexToPlace / 2] ) < 0 ) {
        con[indexToPlace] = con[indexToPlace / 2]; // swap
        indexToPlace /= 2; // change indexToPlace to parent
    }
    con[indexToPlace] = val;
}
```

**Enqueue Example**

- **Add 15 to heap** (initially next left most node)
  - 12
  - 17
  - 15
  - 19
  - 52
  - 37
  - 25
  - 45
  - 21
  - 15

  10 / 2 = 5 (index of parent)

- **Swap 15 and 52**
  - 12
  - 17
  - 15
  - 19
  - 52
  - 37
  - 25
  - 45
  - 21
  - 52

  5 / 2 = 2 (index of parent)
Enqueue Example With Array Shown

- Swap 15 and 17

```
12  
15   15
19   17  37  25
45  21  52
```

\[
2 / 2 = 1 \text{ (index of parent)}
\]

Dequeue -> remove 12

```
12  
15  
19  
17 
37 
25
```

Dequeue

- min value / front of queue is in root of tree
- swap value from last node to root and move down swapping with smaller child unless values is smaller than both children
Dequeue Example

- Swap 35 into root (save 12 to return)

<table>
<thead>
<tr>
<th>0</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>12</td>
<td>15</td>
<td>13</td>
<td>17</td>
<td>23</td>
<td>45</td>
<td>53</td>
<td>45</td>
<td>21</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dequeue Example

- Min child?
  - 1 * 2 = 2 -> 15
  - 1 * 2 + 1 = 3 -> 13
  - Swap with 13

<table>
<thead>
<tr>
<th>0</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>35</td>
<td>15</td>
<td>13</td>
<td>17</td>
<td>23</td>
<td>45</td>
<td>53</td>
<td>45</td>
<td>21</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dequeue Example

- Swap 35 into root (save 12 to return)

<table>
<thead>
<tr>
<th>0</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>35</td>
<td>15</td>
<td>13</td>
<td>17</td>
<td>23</td>
<td>45</td>
<td>53</td>
<td>45</td>
<td>21</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Dequeue Example**

- Min child?
- $3 \times 2 = 6 \rightarrow 45$
- $3 \times 2 + 1 = 7 \rightarrow 53$
- Less than or equal to both of my children!
- Stop!

<table>
<thead>
<tr>
<th>0</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>13</td>
<td>15</td>
<td>35</td>
<td>17</td>
<td>23</td>
<td>45</td>
<td>53</td>
<td>45</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dequeue Code**

```java
public E dequeue( ) {
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while ( hole * 2 < size && ! done ) {
        int child = hole * 2;
        // see which child is smaller
        if ( con[child].compareTo( con[child + 1] ) > 0 )
            child++;    // child now points to smaller
        // is replacement value bigger than child?
        if (con[size].compareTo( con[child] ) > 0 ) {
            con[hole] = con[child];
            hole = child;
        }
        else
            done = true;
    }
    con[hole] = con[size];
    size--;
    return top;
}
```

---

**Clicker 3 - PriorityQueue Comparison**

- Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree
- What will result be?
  - A. Heap takes half the time or less of BST
  - B. Heap faster, but not twice as fast
  - C. About the same
  - D. BST faster, but not twice as fast
  - E. BST takes half the time or less of Heap