YAQDS: Yet another …

- **What is the dequeue policy for a Queue?**
  - Why do we implement Queue with LinkedList
    - Interface and class in Java 5 `java.util`
  - Can we remove an element other than first?

- **How did queue help in word-ladder/shortest path?**
  - First item enqueued/added is the one we want
  - What if different element is “best”?

- **PriorityQueue is like a queue, but different dequeue policy**
  - Best item is dequeued, queue manages itself to ensure operations are efficient
Why use PriorityQueue?

- Implementation of several algorithms facilitated by using pq, efficient implementation helps ensure algorithm efficiency
  - Mapquest, Googlemap, shortest path
    - How is this like word-ladder? How different?
  - Connecting all outlets in a house with minimal wiring

- Data compression facilitated by using priority queue
  - Alltime best assignment in a Compsci 100 course?
    - Subject to debate, of course
  - From A-Z, soup-to-nuts, bits to abstractions
Data Compression

- **Compression is a high-profile application**
  - .zip, .mp3, .jpg, .gif, .gz, ...
  - What property of MP3 was a significant factor in what made Napster work (why did Napster ultimately fail?)

- **Why do we care?**
  - Secondary storage capacity doubles every year
  - Disk space fills up quickly on every computer system
  - More data to compress than ever before
More on Compression

- What’s the difference between compression techniques?
  - .mp3 files and .zip files?
  - .gif and .jpg?
  - Lossless and lossy
- Is it possible to compress (lossless) every file? Why?
- Lossy methods
  - Good for pictures, video, and audio (JPEG, MPEG, etc.)
- Lossless methods
  - Run-length encoding, Huffman, LZW, ...

11 3 5 3 2 6 2 6 5 3 5 3 5 3 10
Priority Queue

- Compression motivates the study of the ADT priority queue
  - Supports two basic operations
    - add/insert -- an element into the priority queue
    - remove/delete -- the minimal element from the priority queue
  - Implementations may allow getmin/peek separate from delete
    - Analogous to top/pop, peek/dequeue in stacks, queues

- See PQDemo.java,
  - code below sorts, complexity?

```java
Scanner s;
PriorityQueue<String> pq =
    new PriorityQueue<String>();
while (s.hasNext()) pq.add(s.next());
while (pq.size() > 0) {
    System.out.println(pq.remove());
}
```
## Priority Queue Implementations

- **Implementing priority queues: average and worst case**

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted vector</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted vector</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balanced tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Heap has O(1) find-min (no delete) and O(n) build heap**
## Priority Queue implementations

- Implementing priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unsorted vector</strong></td>
<td>0(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>Sorted vector</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Search tree</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Balanced tree</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Heap</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Heap has O(1) find-min (no delete) and O(n) build heap
## Priority Queue implementations

- Implementing priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted vector</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted vector</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Search tree</td>
<td></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Balanced tree</td>
<td></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

- Heap has $O(1)$ find-min (no delete) and $O(n)$ build heap
## Priority Queue implementations

- Implementing priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted vector</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted vector</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Search tree</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Balanced tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Heap has $O(1)$ find-min (no delete) and $O(n)$ build heap
### Priority Queue Implementations

- Implementing priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unsorted vector</strong></td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>Sorted vector</strong></td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Search tree</strong></td>
<td>log n</td>
<td>log n</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td><strong>Balanced tree</strong></td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td><strong>Heap</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Heap has O(1) find-min (no delete) and O(n) build heap
Priority Queue implementations

- Implementing priority queues: average and worst case

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted vector</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted vector</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Search tree</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Balanced tree</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>Heap</td>
<td>$O(1)$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
</tbody>
</table>

- Heap has $O(1)$ find-min (no delete) and $O(n)$ build heap
PriorityQueue.java (Java 5)

- What about objects inserted into pq?
  - For "min-heap", what properties must inserted objects have, e.g., insert non-comparable?
  - Change what minimal means?
  - Implementation uses heap

- If we use a Comparator for comparing entries we can make a min-heap act like a max-heap, see PQDemo
  - Where is class Comparator declaration? How used?
  - What if we didn't know about Collections.reverseOrder?
    - How do we make this ourselves?
Sorting w/o Collections.sort(...) 

```java
public static void sort(ArrayList<String> a) {
    PriorityQueue<String> pq =
        new PriorityQueue<String>();
    pq.addAll(a);
    for(int k=0; k < a.size(); k++) a.set(k, pq.remove());
}
```

- How does this work, regardless of pq implementation?
- What is the complexity of this method?
  - add $O(1)$, remove $O(\log n)$? If add $O(\log n)$?
  - heapsort uses array as the priority queue rather than separate pq object.
  - From a big-Oh perspective no difference: $O(n \log n)$
    - Is there a difference? What’s hidden with $O$ notation?
Priority Queue implementation

- **PriorityQueue** uses heaps, fast and reasonably simple
  - Why not use inheritance hierarchy as was used with Map?
  - Trade-offs when using HashMap and TreeMap:
    - Time, space
    - Ordering properties, e.g., what does TreeMap support?
- **Changing method of comparison when calculating priority?**
  - Create object to replace, or in lieu of `compareTo`
    - **Comparable** interface compares `this` to passed object
    - **Comparator** interface compares two passed objects
  - Both comparison methods: `compareTo()` and `compareTo`
    - Compare two objects (parameters or self and parameter)
    - Returns -1, 0, +1 depending on <, ==, >
Creating Heaps

- Heap is an array-based implementation of a binary tree used for implementing priority queues, supports:
  - add/insert, peek/getmin, remove/deletemin, O(???)

- Using array minimizes storage (no explicit pointers), faster too --- children are located by index/position in array

- Heap is a binary tree with shape property, heap/value property
  - shape: tree filled at all levels (except perhaps last) and filled left-to-right (complete binary tree)
  - each node has value smaller than both children
Array-based heap

- store “node values” in array beginning at index 1
- for node with index k
  - left child: index $2k$
  - right child: index $2k+1$

- why is this conducive for maintaining heap shape?
- what about heap property?
- is the heap a search tree?
- where is minimal node?
- where are nodes added? deleted?
Thinking about heaps
Thinking about heaps

- Where is minimal element?
Thinking about heaps

- Where is minimal element?
  - Root, why?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a minheap? Why?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a minheap? Why?
  - O(n), but ½ N?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - $O(n)$, but exact?
- What is complexity of find max in a minheap? Why?
  - $O(n)$, but $\frac{1}{2} N$?
- Where is second smallest element? Why?
Thinking about heaps

- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a minheap? Why?
  - O(n), but \( \frac{1}{2} \) N?
- Where is second smallest element? Why?
  - Near root?
Adding values to heap

- to maintain heap shape, must add new value in left-to-right order of last level
  - could violate heap property
  - move value “up” if too small

- change places with parent if heap property violated
  - stop when parent is smaller
  - stop when root is reached

- pull parent down, swapping isn’t necessary (optimization)
Adding values, details (pseudocode)

```java
void add(Object elt)
{
    // add elt to heap in myList
    myList.add(elt);
    int loc = myList.size()-1;

    while (1 < loc &&
            elt < myList.get(loc/2)){
        myList.set(loc,myList.get(loc/2));
        loc = loc/2; // go to parent
    }
    // what's true here?

    myList.set(loc,elt);
}

array myList
```

CPS 100
Removing minimal element

- Where is minimal element?
  - If we remove it, what changes, shape/property?
- How can we maintain shape?
  - "last" element moves to root
  - What property is violated?
- After moving last element, subtrees of root are heaps, why?
  - Move root down (pull child up) does it matter where?
- When can we stop "re-heaping"?
  - Less than both children
  - Reach a leaf
Anita Borg 1949-2003

- “Dr. Anita Borg tenaciously envisioned and set about to change the world for women and for technology. ... she fought tirelessly for the development technology with positive social and human impact.”
- “Anita Borg sought to revolutionize the world and the way we think about technology and its impact on our lives.”