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 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, ...

Priority Queue

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

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

  ```java
  Scanner s;
  PriorityQueue pq = new PriorityQueue();
  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) average</th>
<th>Insert worst</th>
<th>Getmin (delete) worst</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?
  - If `deletemin` is supported, 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's a static inner class? A non-static inner class?

- In Java 5 there is a `Queue` interface and `PriorityQueue` class
  - The `PriorityQueue` class also uses a heap

**Sorting w/o Collections.sort(...)**

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

- How does this work, regardless of queue 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 `compare()`
    - Compare two objects (parameters or self and parameter)
    - Returns -1, 0, +1 depending on `<, ==, >`

**Heap Definition**

- Heap is an array-based implementation of a binary tree used for implementing priority queues, supports:
  - `insert, findmin, deletemin: complexities?`
- 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`