### Scoreboard

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Insertion</th>
<th>Deletion</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted Vector/array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted vector/array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linked list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash Maps</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What else might we want to do with a data structure?

### Priority Queues

- **Basic operations**
  - Insert
  - Remove extremal

- **What properties must the data have?**

- **Applications**
  - Event-driven simulation: Colliding particles
  - AI
  - Operating systems
  - Statistics
  - Graph searching
  - Data Compression:
  - Physics

### Priority Queue

- Compression motivates the study of the ADT priority queue
  - Supports two basic operations
    - Insert
    - Getmin
  - Implementations may allow getmin separate from delete
    - Analogous to top/pop, front/dequeue in stacks, queues
  - Code below sorts. Complexity?

```java
class PriorityQueue {  
    public static void sort(ArrayList<String> a) {  
        PriorityQueue pq = new PriorityQueue();  
        pq.addAll(a);  
        for(int k=0; k < a.size(); k++)  
            a.set(k, 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>Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balanced binary search tree</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
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

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 <, ==, >

Creating Heaps

- 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

Array-based heap

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

- 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();
    while (1 < loc &&
           elt.compareTo(myList[loc/2]) < 0) {
        myList[loc] = myList[loc/2];
        loc = loc/2; // go to parent
    }
    // what’s true here?
    myList.set(loc, elt);
}
```

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