**Announcements**

* Midterm grading will happen this Saturday, stay tuned for grades
* No quiz today, but will have a quiz tomorrow
* Keep eye out for self-reflection tomorrow
* In today’s lab, if you see the word priority, it corresponds to the priority value!

**Motivation**

* Let’s say we had a list of Fish, and we always wanted to find the Fish with the smallest size and do something
* Can we use a data structure that we’ve already seen before (Queue, Stack, Tree, etc.) and have it still be fast?
* There’s a data structure for this, and it’s called a priority queue! Things with higher priority will get processed first
  + Think of an emergency room
* This can be implemented with a heap.

**Heaps**

* Heaps will keep the higher priority items at the top (so we can access them more quickly), so we must associate each item with a priority value. However, it is up to the heap to decide whether highest priority means largest priority value or smallest priority value.
  + In an emergency room, priority value can be determined by how severe an injury is (higher priority value has more priority)
  + In a line at the grocery store, priority value can be determined who gets there first (lower priority value has more priority)
* Notice the difference between **priority** and **priority value** in the previous statement.
* Heaps are usually implemented as binary trees with the following two properties:
  + **Complete**: There will be no empty spaces within the heap other than on the right-hand side of the bottommost level. Consequence: height will be Theta(logN) where N is the number of nodes!
  + **(Min/Max)-Heap property**: for a particular node n, the children of n must have (greater/lesser) priority value than n; thus, the root will always contain the (lowest/highest) priority value element
* Methods (of a min-heap)
  + peek(): returns (but does not remove) the item with the minimum priority value
  + removeMin(): returns (and does remove) the item with the minimum priority value
    - Take the item in the bottom-rightmost position and replace the value at the root
    - Bubble down the new root value
  + insert(T item, int priorityVal): Insert the item with priority value priorityVal into the heap
    - Insert the item in the bottom-rightmost position
    - Bubble up the new inserted value
* Bubbling (of a min-heap)
  + Bubble up: while the priority value of a particular node n is less than the priority value of its parent, swap the two!
  + Bubble down: while the priority value of a particular node n is greater than the priority value of its child/children, swap the two! (always pick the lesser of the two children if both have priority value less than the current node)
* Runtimes
  + peek(): constant time, we know exactly where the minimum priority element is
  + removeMin(): O(logN), after we replace the root, we need to bubble our item down
  + insert(T item, int priorityVal): O(logN), after we insert, we will need to bubble our item up

**Representation**

* How do we even represent this idea?
* Number each element, starting from 1, left to right top to bottom, this will represent the index of the item in the array! (notice that we do not start at 0)
* For a particular node at index i:
  + Parent is at index i/2
  + Left child is at index 2i
  + Right child is at index 2i + 1
  + What would the indexing look like if we started at 0 instead of 1?