Priority Queues with Binary Heaps

One important variation of the queue is the **priority queue**. A priority queue acts like a queue in that items remain in it for some time before being dequeued. However, in a priority queue the logical order of items inside a queue is determined by their "priority". Specifically, the highest priority items are retrieved from the queue ahead of lower priority items.

We will see that the priority queue is a useful data structure for specific algorithms such as Dijkstra's shortest path algorithm. More generally though, priority queues are useful enough that you may have encountered one already: message queues or tasks queues for instance typically prioritize some items over others.

You can probably think of a couple of easy ways to implement a priority queue using sorting functions and arrays or lists. However, sorting a list is $O(n \log n)$. We can do better.

The classic way to implement a priority queue is using a data structure called a **binary heap**. A binary heap will allow us to enqueue or dequeue items in $O(\log n)$.

The binary heap is interesting to study because when we diagram the heap it looks a lot like a tree, but when we implement it we use only a single dynamic array (such as a Python list) as its internal representation. The binary heap has two common variations: the **min heap**, in which the smallest key is always at the front, and the **max heap**, in which the largest key value is always at the front. In this section we will implement the min heap, but the max heap is implemented in the same way.