Priority Queues with Binary Heaps

In earlier sections you learned about the first-in first-out data structure called a queue. One important variation of a queue is called a **priority queue**. A priority queue acts like a queue in that you dequeue an item by removing it from the front. However, in a priority queue the logical order of items inside a queue is determined by their priority. The highest priority items are at the front of the queue and the lowest priority items are at the back. Thus when you enqueue an item on a priority queue, the new item may move all the way to the front. We will see that the priority queue is a useful data structure for some of the graph algorithms we will study in the next chapter.

You can probably think of a couple of easy ways to implement a priority queue using sorting functions and lists. However, inserting into a list is $O(n)$ and 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 both enqueue and 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 list as an 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. We leave a max heap implementation as an exercise.

❮ (TreeTraversals.html) ❯ (BinaryHeapOperations.html)

Mark as completed

aversals.html)

Next Section - Binary Heap Operations (Bina