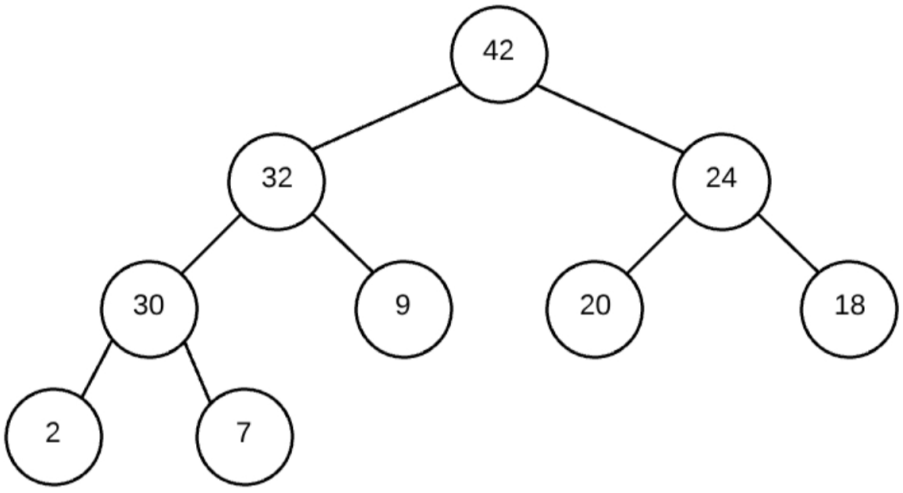
Binary Heap Introduction

5 minutes

Introduction to Heaps

Let's explore the **Heap** data structure! In particular, we'll explore **Binary Heaps**. A binary heap is a type of binary tree. However, a heap is not a binary *search* tree. A heap is a partially ordered data structure, whereas a BST has full order. In a heap, the root of the tree will be the maximum (max heap) or the minimum (min heap). Below is an example of a max heap:



Notice that the heap above does not follow search tree property where all values to the left of a node are less and all values to the right are greater or equal. Instead, the max heap invariant is:

- given any node, its children must be less than or equal to the node

This constraint makes heaps much more relaxed in structure compared to a search tree. There is no guaranteed order among "siblings" or "cousins" in a heap. The relationship only flows down the tree from parent to child. In other words, in a max heap, a node will be greater than all of it's children, it's grandchildren, its great-grandchildren, and so on. A consequence of this is the root being the absolute maximum of the entire tree. We'll be exploring max heaps together, but these arguments are symmetric for a min heap.

Complete Trees

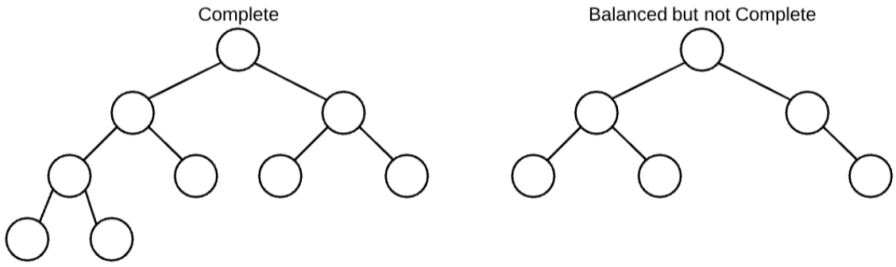
We'll eventually implement a max heap together, but first we'll need to take a quick detour. Our design goal is to implement a data structure with efficient

operations. Since a heap is a type of binary tree, recall the circumstances where we had a "best case" binary tree. We'll need to ensure our heap has minimal height, that is, it must be a balanced tree!

Our heap implementation will not only be balanced, but it will also be **complete**. To clarify, **every complete tree is also a balanced tree**, but not every balanced tree is also complete. Our definition of a complete tree is:

- a tree where all levels have the maximal number of nodes, except the bottom the level
- AND the bottom level has all nodes filled as far left as possible

Here are few examples of the definition:



Notice that the tree is on the right fails the second point of our definition because there is a gap in the last level. Informally, you can think about a complete tree as packing its nodes as closely together as possible. This line of thinking will come into play when we code heaps later.

When to Use Heaps?

Heaps are the most useful when attacking problems that require you to "partially sort" data. This usually takes form in problems that have us calculate the largest or smallest n numbers of a collection. For example: What if you were asked to find the largest 5 numbers in an array in linear time, O(n)? The fastest sorting algorithms are O(n logn), so none of those algorithms will be good enough. However, we can use a heap to solve this problem in linear time.

We'll analyze this in depth when we implement a heap in the next section!

One of the most common uses of a binary heap is to implement a "priority queue". We learned before that a queue is a FIFO (First In, First Out) data structure. With a priority queue, items are removed from the queue based on a priority number. The priority number is used to place the items into the heap and pull them out in the correct priority order!

✓ Mark As Complete

Finished with this task? Click **Mark as Complete** to continue to the next page!