Pros and cons of tree data structures

Selecting transcript lines in this section will navigate to timestamp in the video

- Just like all of the data structures we've seen before, sets, binary search trees, and heaps all have their pros and cons. Sets are great if you need to constantly check if a certain value exists in the set and do not need duplicate values. In any other situation, you might want to use a different data structure. Sets are intentionally limited, so there isn't much more you can do with them. If your data has an order and you will be inserting, deleting, searching and accessing items a lot, then binary search trees are great. They maintain sorted order while staying fast for insertion, deletion and accessing. If the tree is balanced, then it take O of log N time because with every node we encounter, we eliminate another half of the tree. If unbalanced, then it could take up to linear time because each node could all be on the one side. However, there is some overhead here because of their creation and management and also the idea of continuously rebalancing the tree, so it's important to keep that in mind. Heaps are a type of binary tree that are great for priority queues. It takes constant time to find the min or max and inserting takes O of log N time because again, we could end up making log N swaps but they'll only be on side of the tree. Cons, searching and deleting take O of N time because we have to scan all the elements and there's no guaranteed specific order. Deleting requires having to traverse the whole tree to access the element first and then delete it where the delete operation itself requires O of log N time due to the swapping of elements that might take place.