TUTORIAL 4

Recap: binary search trees (BSTs)

The motivation for BSTs comes from **sorted arrays**. If we have an arbitrary array with n elements, then searching for an element in the array is a worst-case $O(n)$ operation. But if the array is sorted, we can use **binary search** to do this search in worst-case $O(\log n)$ time:

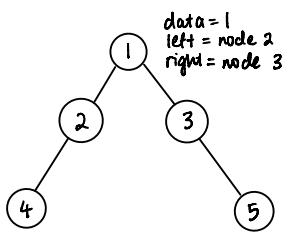
Search for 6 in $\{3\ 6\ 11\ 7\ 8\ 1\ 4\} \Rightarrow O(n)$

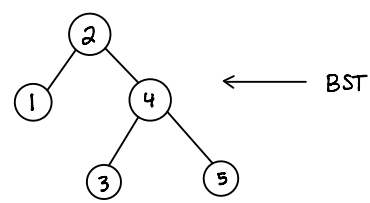
$\{1\ 3\ 4\ 6\ 7\ 8\ 11\} \Rightarrow O(\log n)$

Keeping an array sorted is **hard** and **expensive** though. (More on this late in the term)

⇒ Think hierarchically instead of laterally!

A **binary tree** is a data structure consisting of nodes, where **each node has 0, 1 or 2 children.**





Each node consists of some data and references to its left and right child node.

If the nodes of a binary tree have the property that

$$\text{left data} < \text{node data} < \text{right data}$$

then we call it a **binary search tree**. BSTs allow us to insert and search for a node in $O(\log n)$ time if done properly. In this way, we can think about them as generalisations of the idea of binary search.

Some terminology: