CSC290 Blog Post Sample 1

A `binary_search_tree` is a data structure that organizes data in a way that is more efficient on average than a standard data array. (...)

A `binary_search_tree` (called a tree from here on out) is made up of a network of `nodes` that connect to one another. A tree starts with one `node` that is called the `root`, which branches out in 2 directions. Each `node` contains 3 pieces of information. The first is a piece of data that we are storing in the tree, and its associated index or `key`. The second is some other `node` whose `key` is less than the `root`. We call this the `left_branch`. The final piece of information is some other `node` whose `key` is greater than the `root`. We call this the `right_branch`.

If we want to find a particular item in the tree we look at the `key` of the `root` and use it to figure out where to go. If the desired item has a `key` that is less than the `root` then we go to the `left_branch` and repeat the process, if the desired item's `key` is greater we go to the `right_branch` and repeat the process.

However, this brings up the question of which `node` to make into the `root` of the tree. To maximize efficiency we want the tree to have roughly the same number of `nodes` connected to the `left_branch` and the `right_branch` of the `root`. This is called a balanced tree.

When creating a balanced tree, we pick the `root` by selecting the `node` that contains the `key` that corresponds to the middle of the data being stored. The `node` connected on the `left_branch` will be the midpoint between the `root` and the beginning of the data, and the `node` on the `right_branch` will be the midpoint between the `root` and the end of the data.

We then connect each `node` to its own `left_branch` and `right_branch`, based on the same criteria, and repeat the process until all the data is stored in the tree. (...)

CSC290 Blog Post Sample 2

What are binary search trees? Is it at all related to the binary search algorithm? These are all questions that may be asked by someone who is trying to understand a binary search tree, and I will be answering these questions today.

To put it simply, a binary search tree is a data structure. Instead of using a built in data structure such as a list, we can create our own data structure such as a binary search tree. Within a binary search tree, data is stored in what is called a node within a tree. A node can have two children nodes, one on the left of the parent node and one on the right of the parent node. The root node; the first node in the tree, does not have a parent node and the leaf nodes; the last nodes in the tree, do not have children nodes.

The structure of a binary search tree is simple, given a parent node with a value, if you have a new value check if this new value is larger or smaller than the parent nodes value. If the new nodes value is smaller, make that node the left child of the parent node. If the new nodes value is larger, make that node the right child of the parent node. Using this pattern, we can locate any value we want to find in this tree and insert values in the tree.

The name binary search tree implies that this data structure is somewhat related to binary search. To keep it simple, binary search finds values by dividing the problem in half until it finds the correct value. Now, using a binary search tree to find a value seems extremely similar to the binary search algorithm. If we have a value, look at the left side of the root node and the right side of the root node. Knowing the nodes values on the left and right side, one must only go search through the left child's tree or the right child's tree. This would essentially divide the problem in half just like the binary search algorithm.