traversals

```cpp
void BST::print(BNode<int> *t){
    if (t != NULL)
        print(t->left);
    cout << t->element;
    print(t->right);
}
}
```

AVL Trees

- **Motivation**: we want to **guarantee** $O(\log n)$ running time on the find/insert/remove operations.
- **Idea**: keep the tree balanced after each operation.
- **Solution**: AVL (Adelson-Velskii and Landis) trees.

- **AVL tree property**: for every node in the tree, the **height** of the left and right subtrees differs by at most 1.

**AVL trees: find, insert**

- AVL tree **find** is the same as BST find.
- AVL **insert**: same as BST insert, except that we might have to “fix” the AVL tree after an insert.
- These operations will take time $O(d)$, where $d$ is the depth of the node being found/inserted.
- What is the maximum height of an $n$-node AVL tree?

**AVL tree insert**

- Let $x$ be the **deepest** node where an imbalance occurs.
- Four cases to consider. The insertion is in the
  1. left subtree of the left child of $x$.
  2. right subtree of the left child of $x$.
  3. left subtree of the right child of $x$.
  4. right subtree of the right child of $x$.
- Idea: Cases 1 & 4 are solved by a **single rotation**. Cases 2 & 3 are solved by a **double rotation**.

**Single rotation example**
Single rotation in general

\[ X < b < Y < a < Z \]

Cases 2 & 3 — try a single rotation...

\[ X < b < Y < a < Z \]

Single rotation fails!

Double rotation, step 1

Double rotation, step 2

Double rotation in general

AVL tree: Running times

- **find** takes \( O(\log n) \) time, because height of the tree is always \( O(\log n) \).
- **insert**: \( O(\log n) \) time because we do a find \( O(\log n) \) time), and then we may have to visit every node on the path back to the root, performing up to 2 single rotations \( O(1) \) time each) to fix the tree.
- **remove**: \( O(\log n) \) time. Left as an exercise.