MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT9136: Algorithms and programming foundations in Python

Week 7: Binary Trees and BSTs

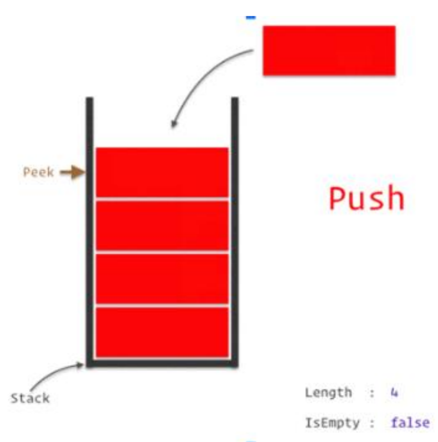
GROUP OF EIGHT AUSTRALIA

Agenda

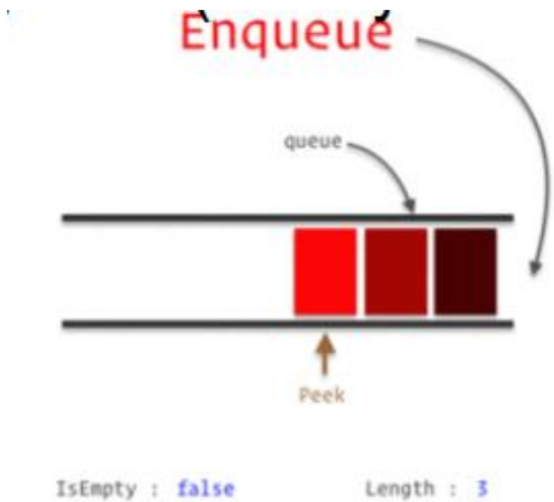
- Review of Week 6
- Synopsis
- Learning Objectives
- Fundamental Class Update
  - Nodes and Links
- Advanced Data Structure:
  - Binary Trees
- Abstract Data Type:
  - Binary Search Trees

Review of Week 6

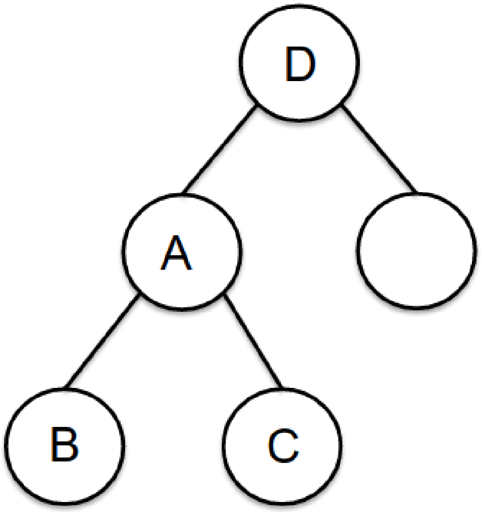
Stack



Queue



Heap



Week 7 Synopsis

- Week 7 is aimed to provide you with:
  - Advanced Data Structure and ADT:
    - Binary Trees and Binary Search Trees

Learning Objective

- Understand the concept of Binary Trees and Binary Search Tree

Fundamental Class Update:
Nodes and Links

Nodes

- Node:
  - Nodes for the data structure for today require two links rather than one.
  - There is a need for a left, and a right element, thus, the definition has changed:

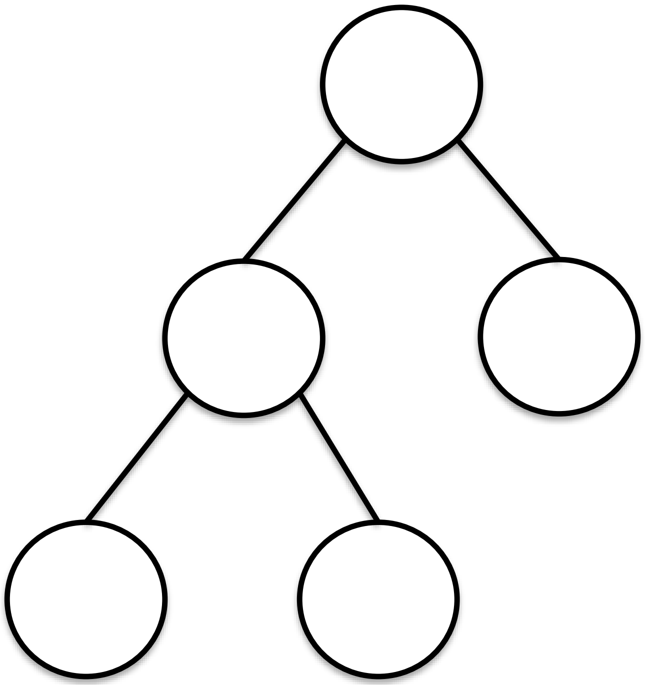
```
class Node:

    def __init__(self, item, left = None, right = None):
            self.item = item
            self.left = left
            self.right = right
```

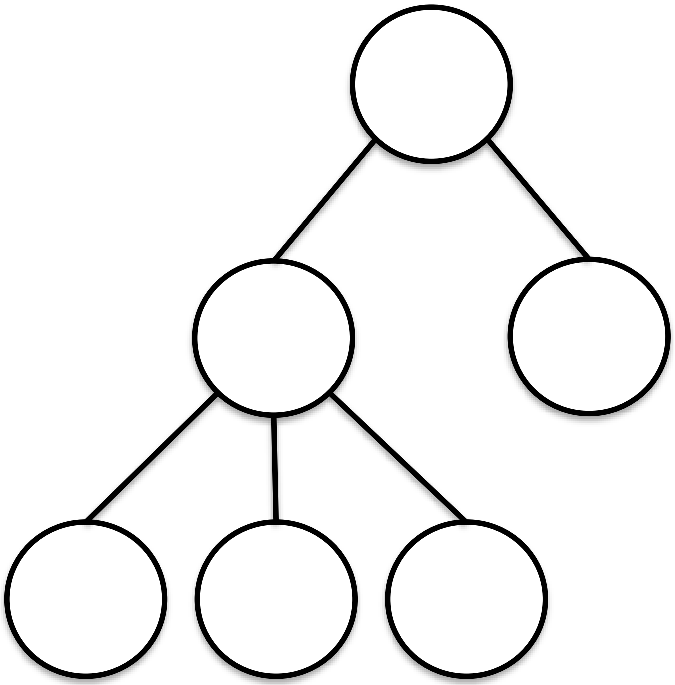
- This now allows for Binary Trees to be constructed.
  - If we wanted to make Trinary trees there would need to be three links. (Tri meaning 3)

Advanced Data Structure: Binary Trees

- A binary tree is a tree whose nodes have at most 2 children.

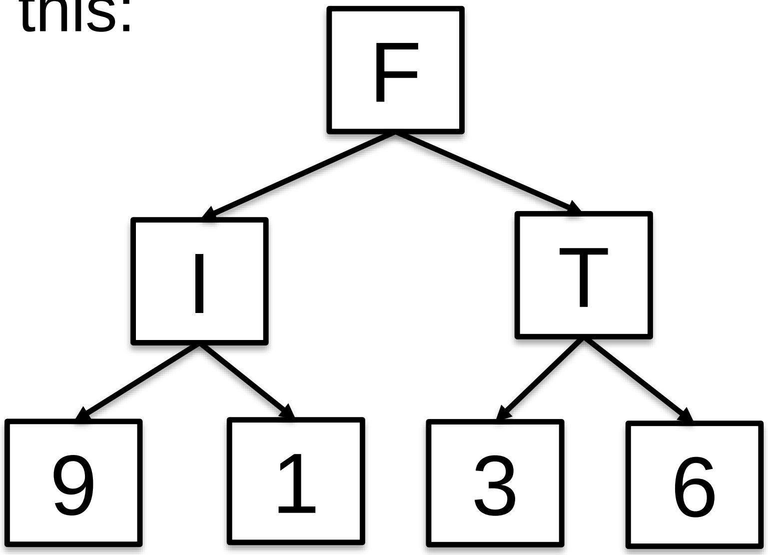


Binary Tree

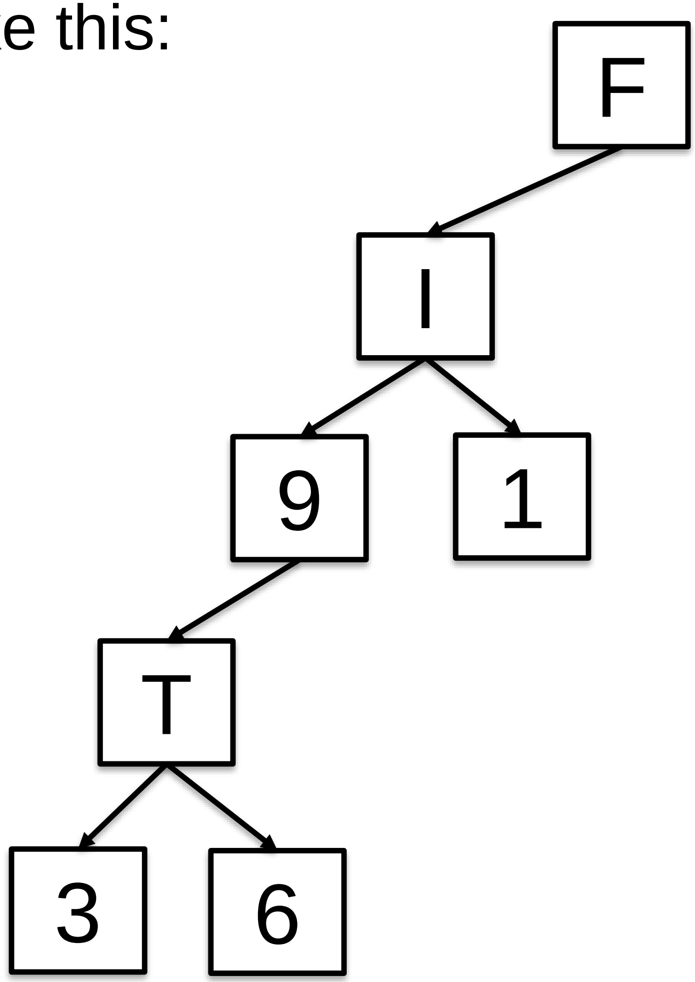


Non-Binary Tree

- Using a Binary Tree structure: our unit's name could look like this:



- Or like this:



- Ultimately the structure isn't useful without rules in place to mandate where items are placed.

- Trees (not just binary trees) are useful to show chains of actions from a preceding state.
  - Not just actions chosen, but all possible choices.
  - The rules imparted on the tree gives the tree its usefulness.

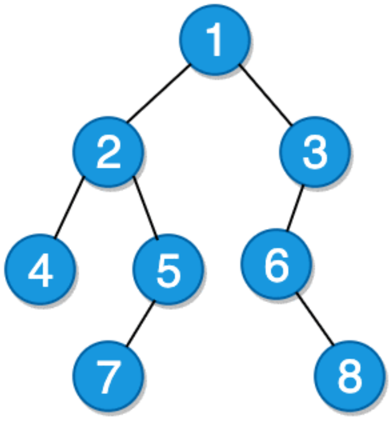
Abstract Data Type:
Binary Search Trees

Tree Search

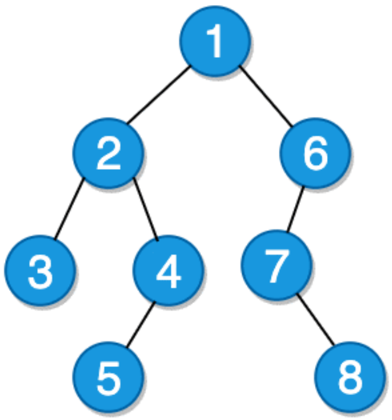
- Tree search refers to the process of finding one node in a tree data structure.

- Two tree search steps:
  - Breadth First Search (BFS)
  - Depth First Search (DFS)

BFS



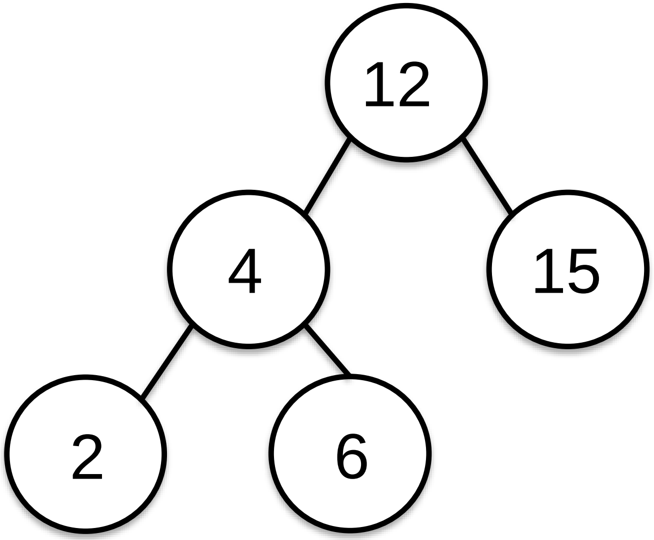
DFS



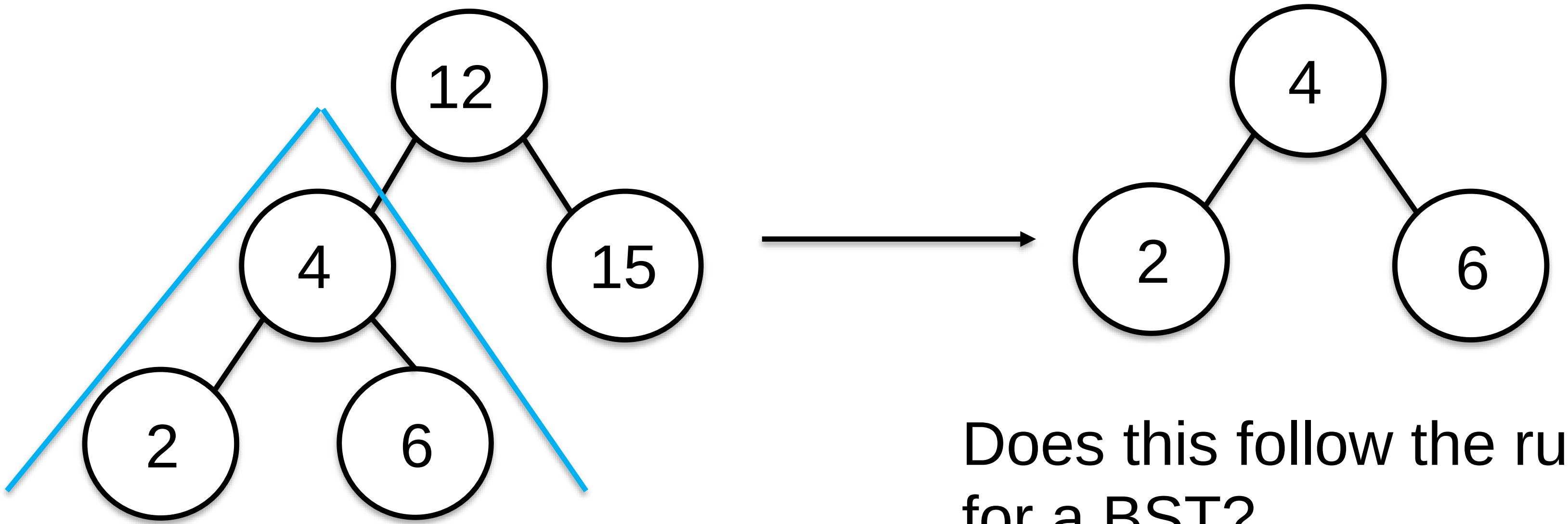
- **Binary Search Tree (BST)** is a node-based binary tree data structure which has the following properties:
  - The left subtree of a node contains only nodes with values smaller than the root.
  - The right subtree of a node contains only nodes with values larger than the root.
  - The left and right subtree each must also be a binary search tree

- Binary Search Trees exists to allow for fast searching for a dataset.

An Example BST

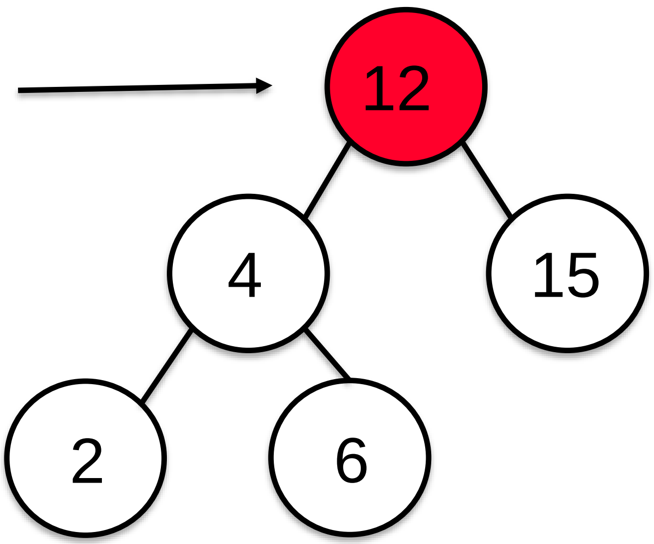


- The sub-trees of a BST are also a BST.



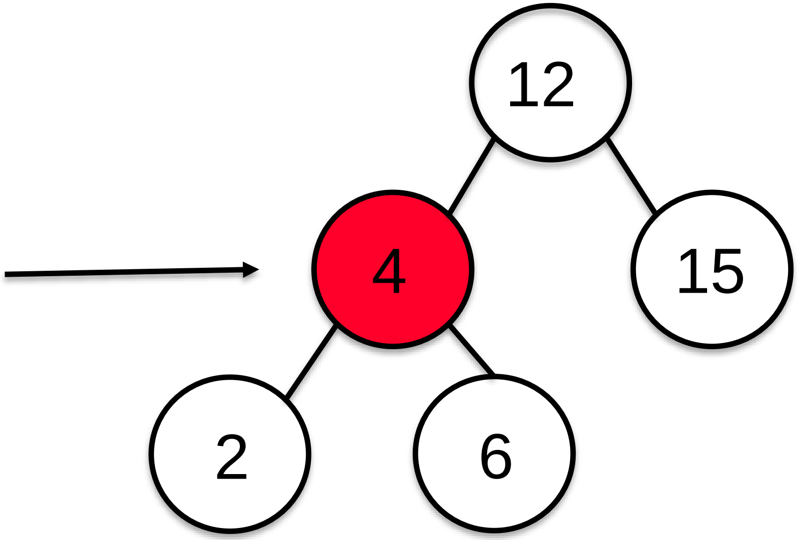
Does this follow the rules for a BST?

- Let's look for 6. To start searching, we start at the root:



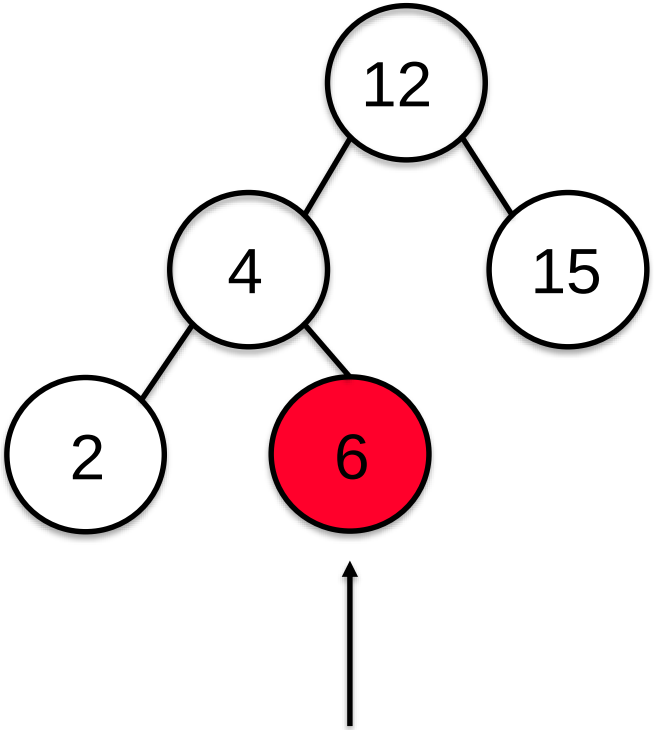
- Is the root what we want? No.
- Is 6 > 12? No.
  - 6 must exist to the left of 12.
- Go left.

- Continue the quest for 6:



- Is the root what we want? No.
- Is 6 > 4? Yes.
  - 6 must exist to the right of 4.
- Go right.

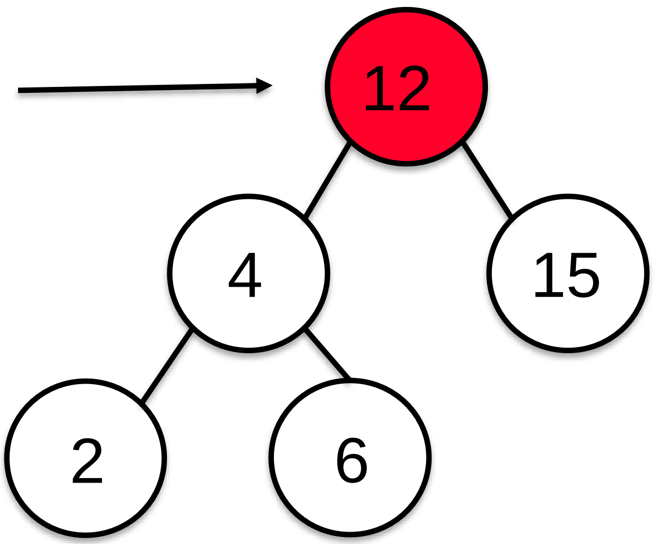
- Continue the quest for 6:



- Is the root what we want? Yes.
- Return True.

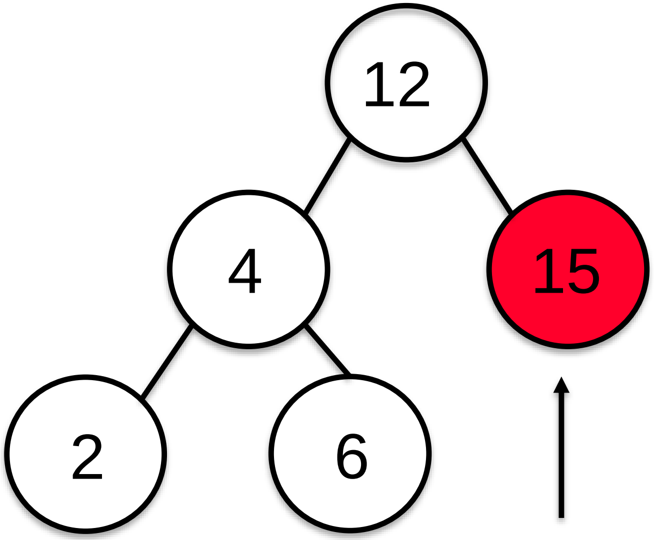
How do we find an element?

- Let's look for 14. To start searching, we start at the root:



- Is the root what we want? No.
- Is 14 > 12? Yes.
  - 14 must exist to the right of 12.
- Go right.

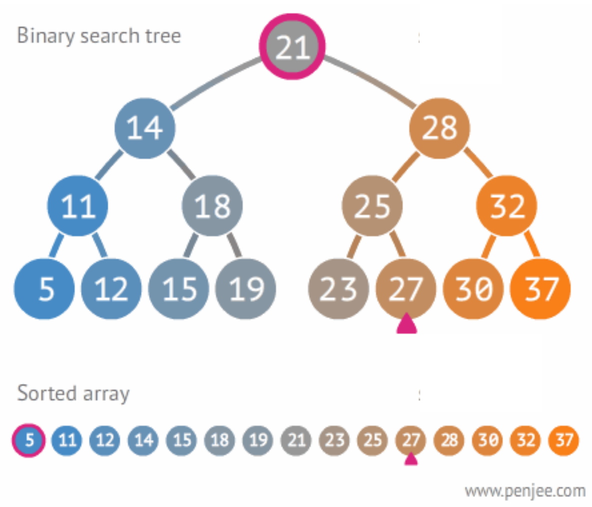
- Continue the quest for 14:



- Is the root what we want? No.
- Is 14 > 15? No.
  - 14 must exist to the left of 15.
- Cannot keep searching.
- Return False.

Why do we use BST?

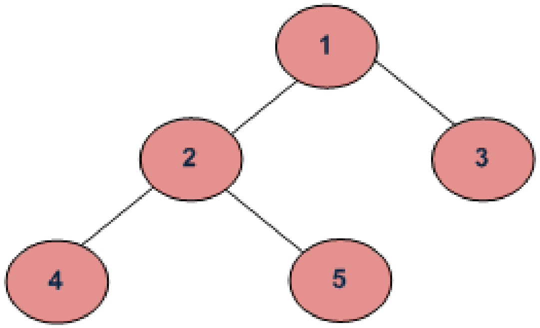
- Binary Search Trees is much more efficient for searching one element in a list



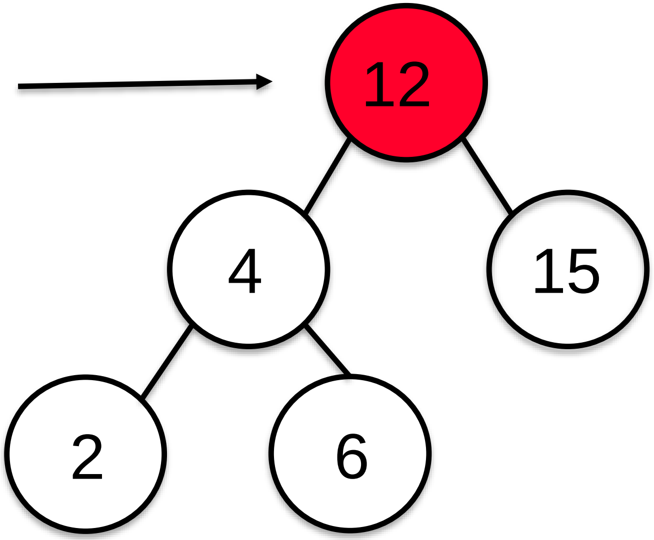
https://blog.penjee.com/5-gifs-to-understand-binary-search-tree/

- Find the **minimum** element:
  - All way to the left.
- Find the **maximum** element:
  - All way to the right.

- Tree traversal:
  - Refers to the process of visiting each node in a tree data structure, exactly once
  - Depth First Traversal (DFT),
  - Breadth First Traversal (BFT)

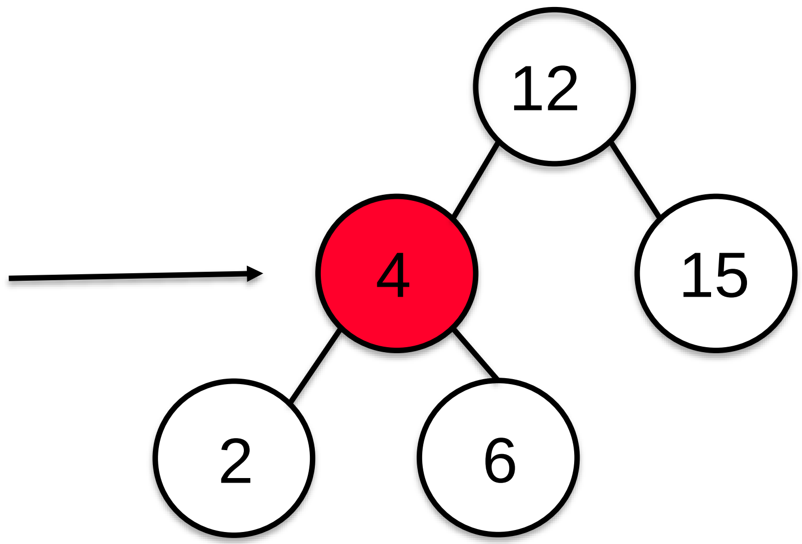


- First find the location. This is what we have already done. Lets try to add 5.



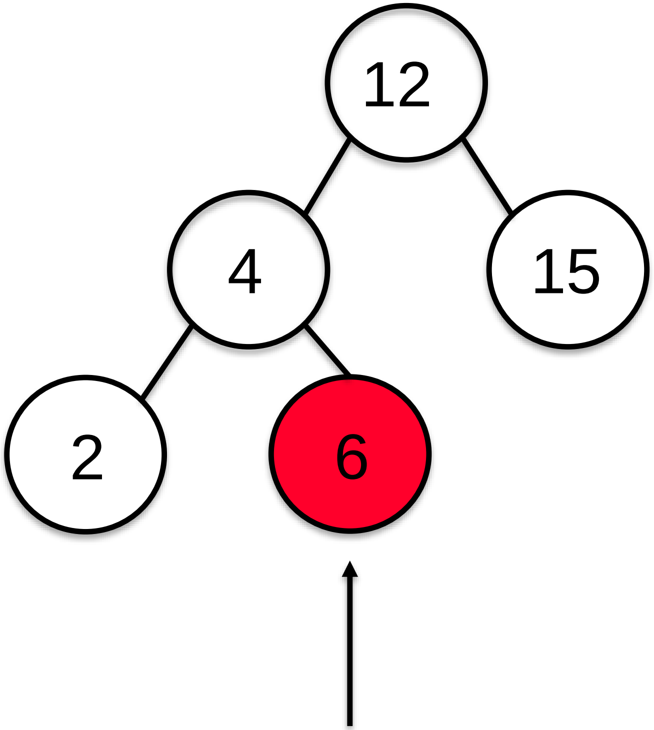
- Is the root what we want? No.
- Is 5 > 12? No.
  - 5 must be inserted to the left of 12.
- Go left.

- Continue the search:



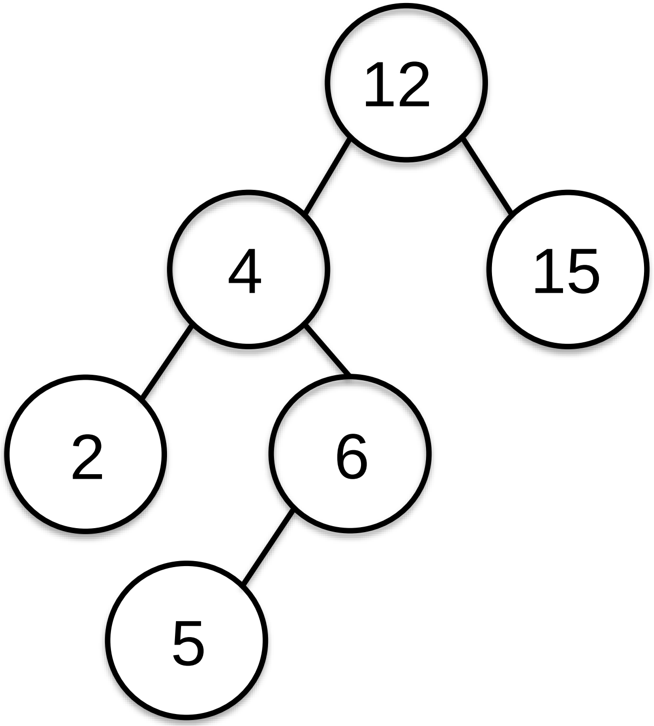
- Is the root what we want? No.
- Is 5 > 4? Yes.
  - 5 must be inserted to the right of 4.
- Go right.

- Continue the search:



- Is the root what we want? No.
- Is 5 > 6? No.
  - 5 must be inserted to the left of 6.

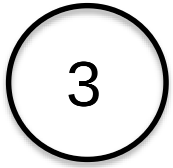
- We have the location!



- The left node does not exist.
- We have found the location for 5.
- Insert 5 here.

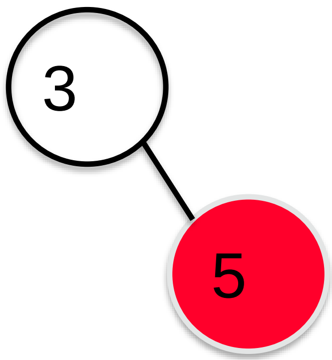
How do you construct from base?

- We have the list: [3,5,4,7,6,1]



- 3 is the root

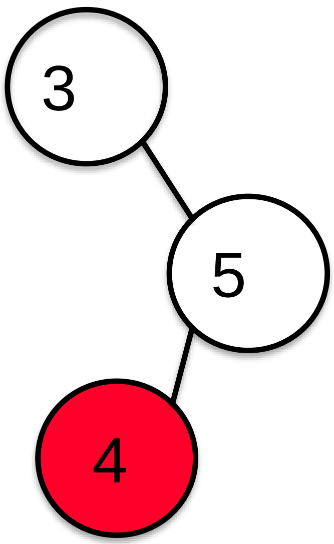
- We have the list: [3,5,4,7,6,1]



- Is 5 > 3? Yes.
  - 5 must be inserted to the right of 3.
- Go to the next element.

How do you construct from base?

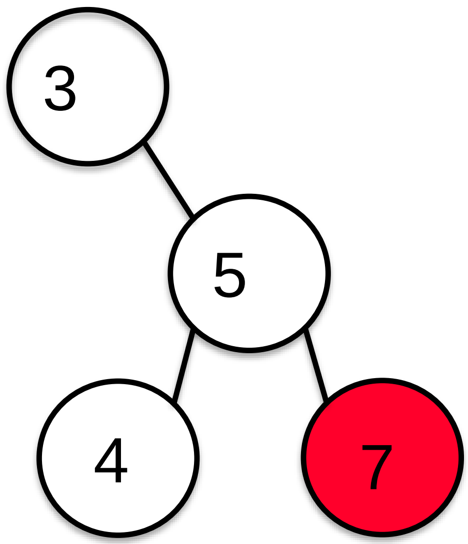
- We have the list: [3,5,4,7,6,1]



- Is 4 > 3? Yes.
  – 4 must be inserted to the right of 3.
- Is 4 > 5? No.
  – 4 must be inserted to the left of 5.

How do you construct from base?

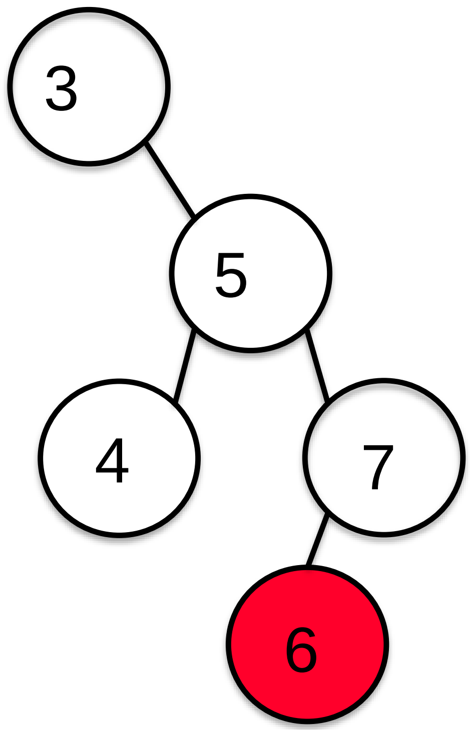
- We have the list: [3,5,4,7,6,1]



- Is 7 > 3? Yes.
  - 7 must be inserted to the right of 3.
- Is 7 > 5? Yes.
  - 7 must be inserted to the right of 5.

How do you construct from base?

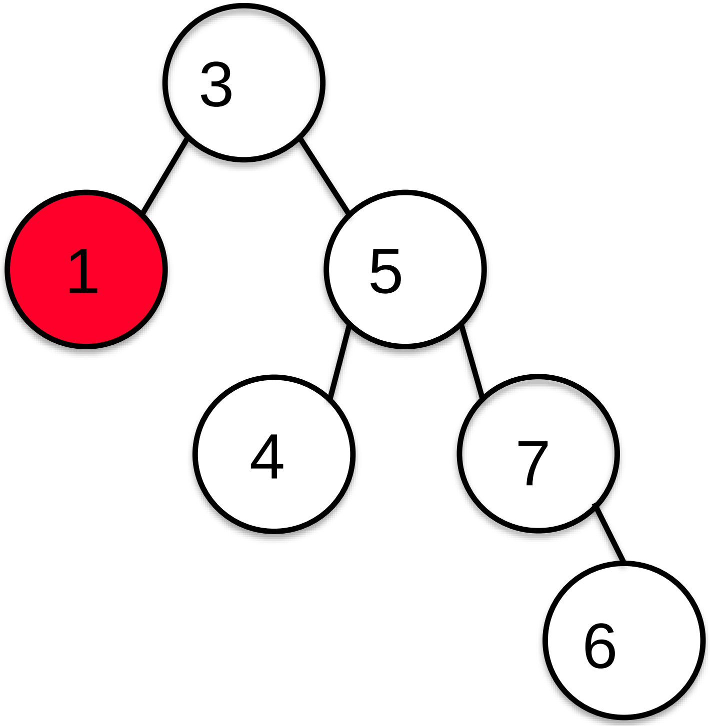
- We have the list: [3,5,4,7,6,1]



- Is 6 > 3? Yes.
  - 6 must be inserted to the right of 3.
- Is 6 > 5? Yes.
  - 6 must be inserted to the right of 5.
- Is 6 > 7? No.
  - 6 must be inserted to the left of 7.

How do you construct from base?

- We have the list: [3,5,4,7,6,1]



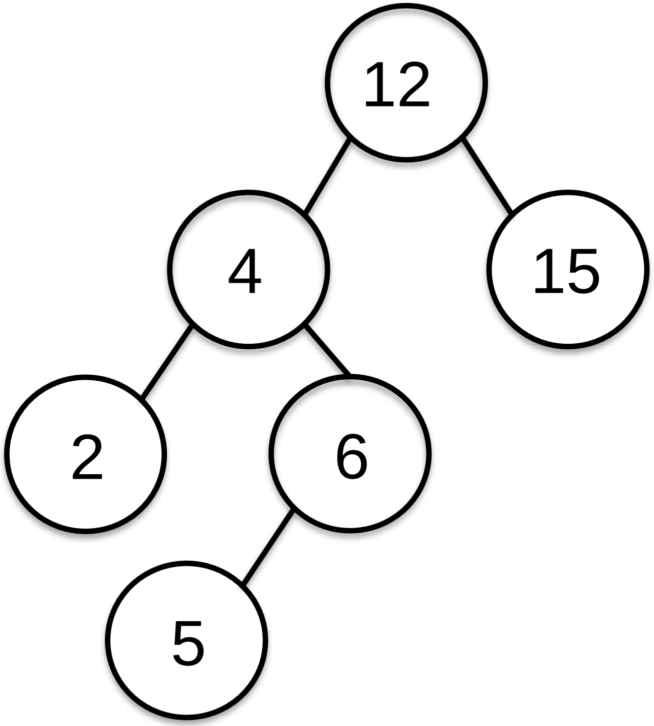
- Is 1 > 3? No.
  - 1 must be inserted to the left of 3.

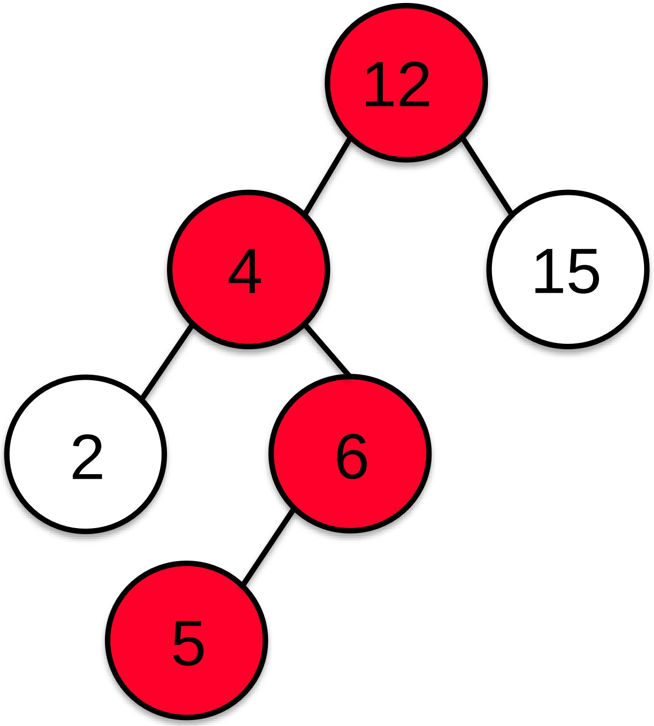
- What happens if we try to insert an element already there?
  - When we try to search for a location, if we come across the item in a node, stop searching.
  - Do not insert two elements of the same value into a search tree.

- BST allows for fast search times to discover if elements are present.
- How much time does it take? It depends on “balance”.

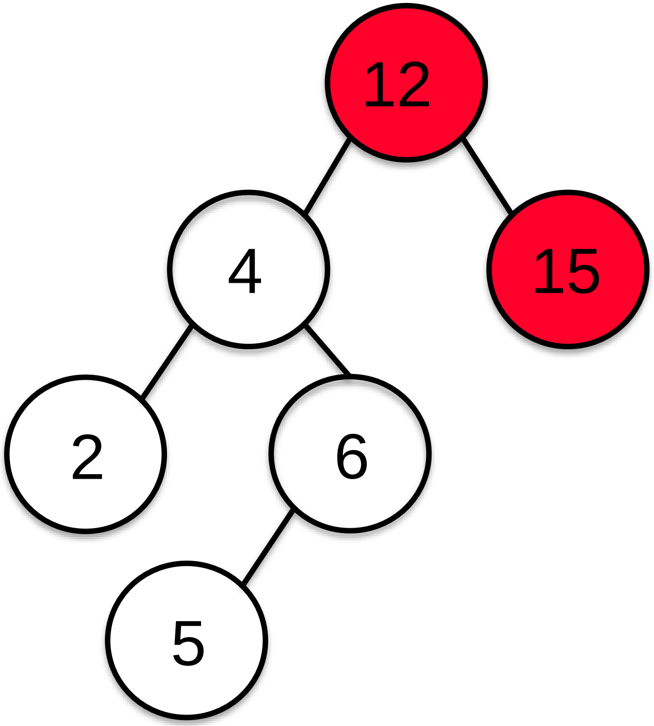
- A tree is considered balanced if the length of a branch is “close” to the length of all other branches.
  - For the sake of this unit, “close” will refer to +/- 1.

- The tree we just made is considered NOT balanced.





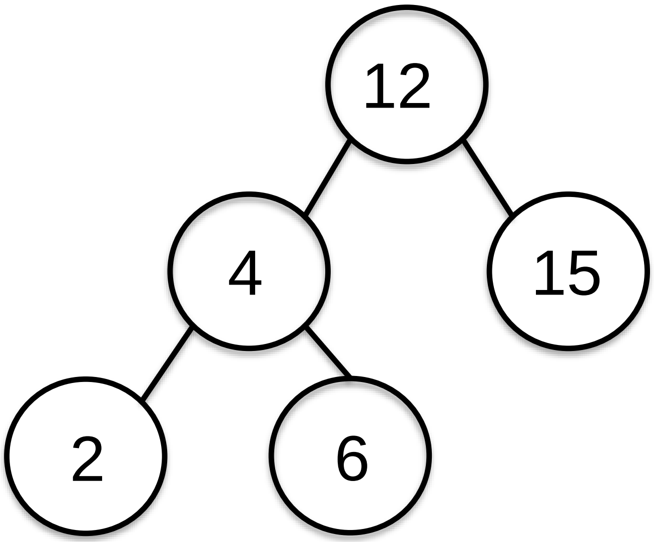
The depth of this branch is 3.

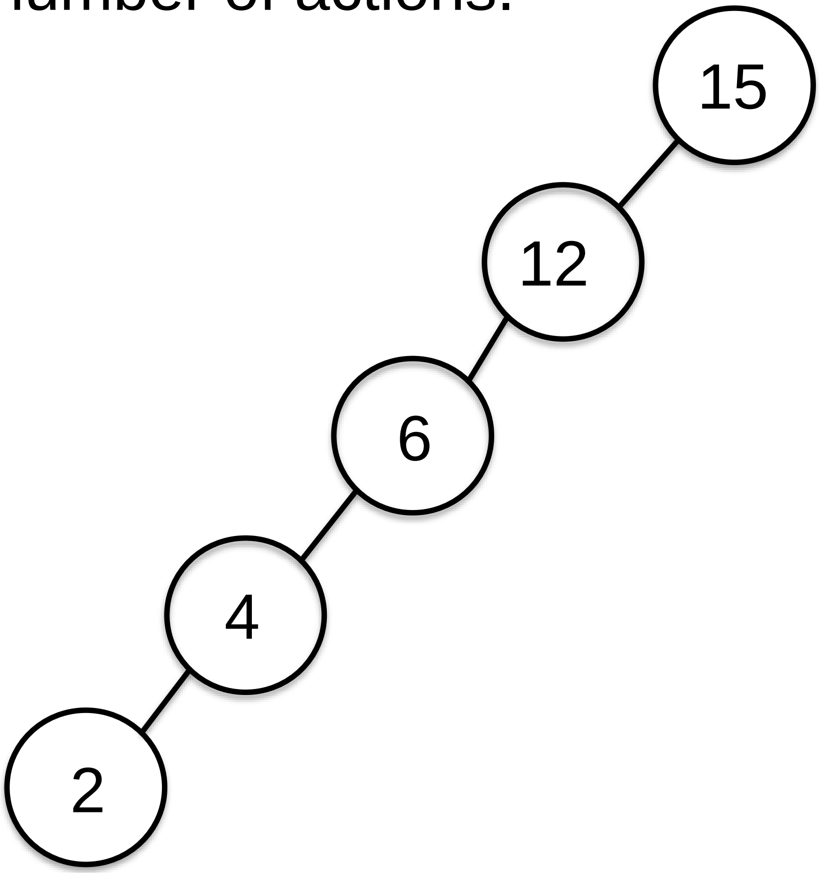


The depth of this branch is 1.

- 3-1 = 2, thus, because of the difference in depth, the tree is not balanced.
- Why should we care? Consider the following situations.

- Search for 2, count the number of actions.





- For the balance tree, there are 3 probes to find 2.
- For the imbalanced tree there are 5 probes.
- This suggests that balance has an impact on complexity for BSTs (and it does).
- Right now, the numbers are quite small, but for very large trees there is a significant increase in time taken to find elements.

- For a balanced tree, the complexity is $\boldsymbol{log_2\, n}$ because with every step down the tree, the search space possibilities are halved.

- For an imbalanced tree, the complexity is $\boldsymbol{n}$ because probing for an element that is not there, in the worst case, will require each element to be examined.

Rebalancing

- Is it worth rebalancing trees? Think about it for your workshop.

- Rebalancing trees is outside the scope of the unit but is a very interesting topic to go into.