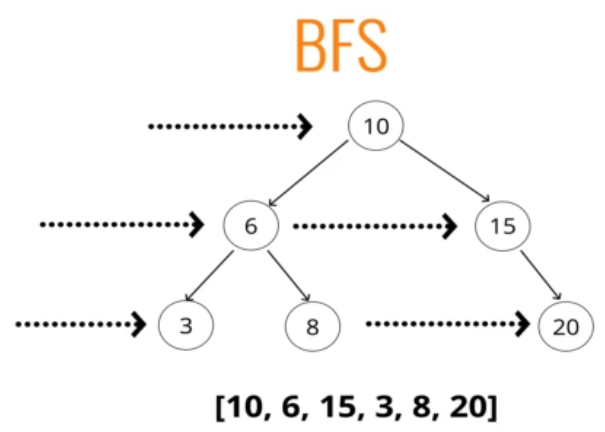
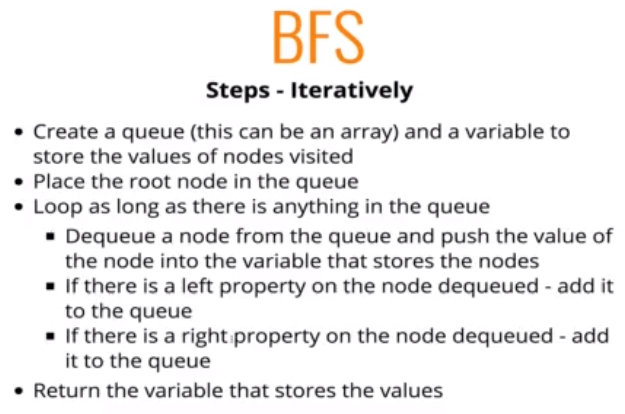
JavaScript Algorithms and Data Structures Masterclass

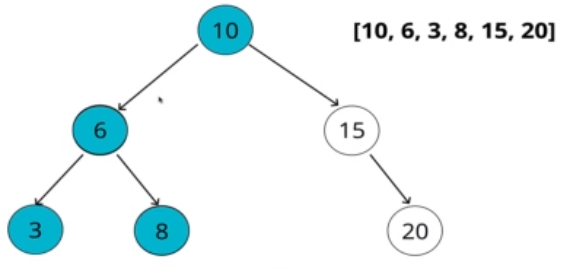
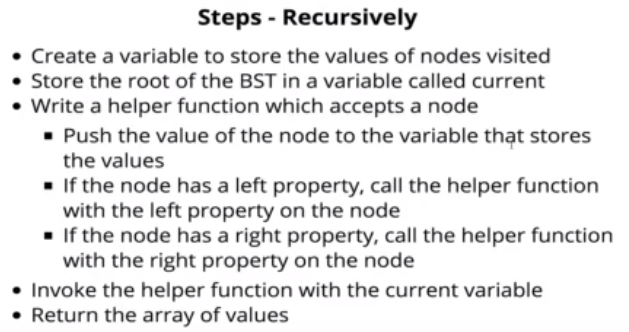
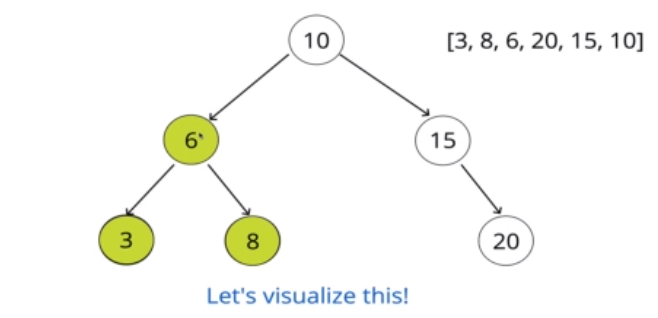
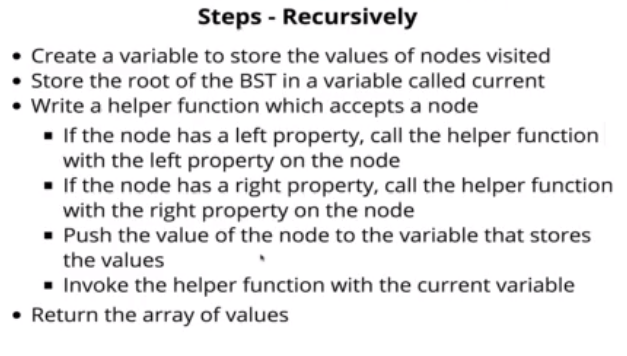
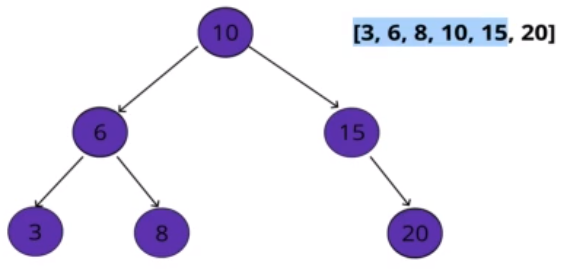
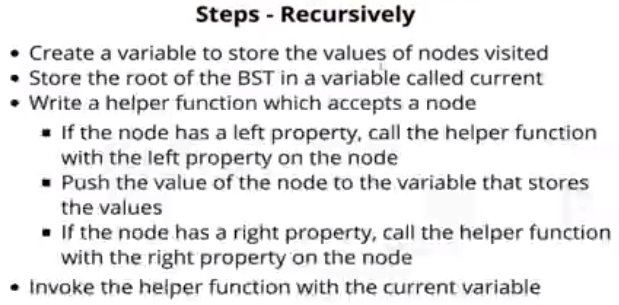
# Section 23: Tree Traversal

## Approaches to visiting every node once in a Generic Tree?

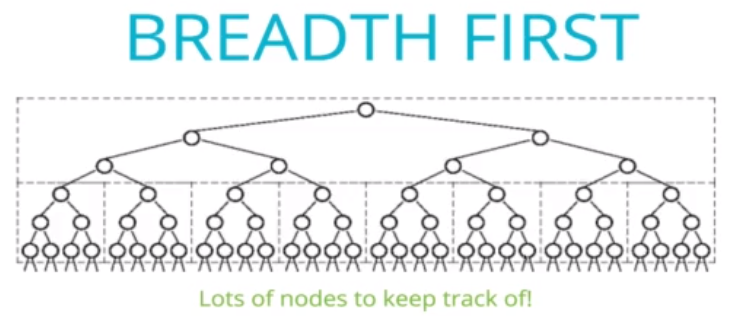
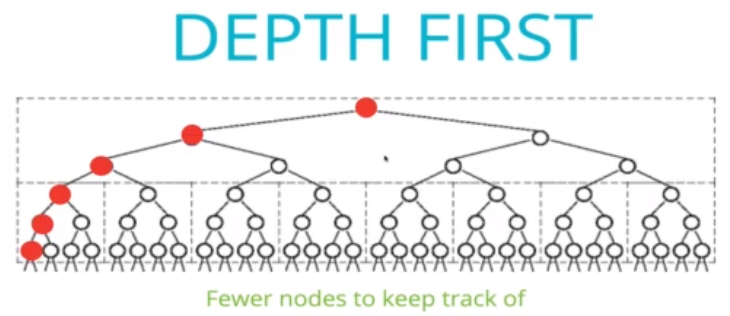
### *Breadth First Search (BFS)-* Sibling Search

* Check each sibling on the same level first (*breadth*), before moving to the next level (*depth*)
  + **Diagram**:
    - 
* **Psuedo-code**:
  + 

### *Depth First Search (DFS)* – Top Down Search

* **Pre-order DFS**
  + Traversaloccurs in a top-down manner, but you do all of left side, then when reached bottom, do the right side
  + **Diagram**:
    - 
  + **Psuedo-code**:
    - 
* **Post-Order DFS**
  + The root is the last node visited. Left, then right, then up to the parent node
    - **Diagram**:
      * 
    - **Psuedo-code**:
      * 
        + In this case, the push occurs last instead of first
* **In-Order DFS**
  + Going left to right, you traverse this tree in order
  + Traverse all the way to the left first, then start ‘visiting’/pushing in the values, then traverse right
    - **Diagram**:
      * 
  + **Psuedo-code**:
    - 
      * The push into the visited occurs in between

## Comparison of BFS and DFS

* This is situational, but Time Complexity is the SAME for both
  + Wide tree:
    - **BFS** tracks a lot of nodes
      * Since you are going through each level
      * 
    - **DFS** tracks fewer nodes
      * 
  + Thin Tree:
    - BFS has fewer nodes to track:
      * 
    - DFS tracks more nodes due to the recursion stack as you go through each level
* Comparison of DFS Variants
  + In-order DFS
    - Used primarily in BSTs since it returns ordered data
  + Pre-Order DFS
    - Used primarily used to ‘export’ a Tree so it can be easily reconstructed/copied
      * Easy to rebuild the tree again

## Recap:

* 