# Data Structures and Algorithms

This document details how to learn data structures and algorithms

# Determine Data Structure or Algorithm to Learn

In this step, you will simply select a data structure or algorithm you want to learn. A good place to start if the books collection or the data structures and algorithms specialization.

# For Each Data Structure or Algorithm

For example, if you selected LinkedList

1. Select 1 of the following 4 programming languages
   1. Python
   2. JavaScript
   3. Java
   4. C#
2. Watch videos on what a linked list is
3. Determine the operations the linked list supports
4. Create a skeleton class that implements the linked list and all operations. This class should not implement any functionality yet.
5. Create a series of unit tests on the class to test whether all supported operations.
   1. Implement edge and corner cases such as passing null references, passing empty lists, passing objects that the class does not expect (for example pass a number where a string is expected).
   2. Implement stress test on the class and compare your code to code you stole from the internet. For example, a linked list implementation, your implementation should be coming back exactly as a valid implementation.
   3. Show the unit tests failing
6. Write pseudo code and comments for the implementation in the class
7. Run through the pseudo code in a memory model or memory diagram to figure out what will happen under the hood <https://www.youtube.com/watch?v=wAx2DgsHQg4>
8. Implement class for the linked list and show the unit tests passing.
9. Look for 1 additional question on the internet. For example, how to reverse a linked list.

# List of Data Structures and Algorithms

Data Structures

1. Queues Done
   1. Queue - Done
   2. Binary Heap - Done
      1. Max heap -Done
   3. Priority Queue Using Binary heap – Done All the same operations as binary max heap
2. Stack – Done in Java
3. Trees
   1. Binary Tree – Done in C#
   2. Trie
   3. AVL Tree
   4. Splay Tree
4. Heap
5. Stack
6. Disjoint Set
7. Hash Table
   1. Hash Function
   2. Distributed Hash Table
   3. Hash map
   4. Hash set
8. Graph

Algorithms

1. Rabin-Karp Algorithm
2. Heap sort
3. Breadth first search in trees
4. Graph algorithms
   1. Graph ordering - pre and post visit order
5. Directed Acyclic Graphs
6. Topological Sort
7. Strongly Connected Components
8. Most direct route in a graph
9. Breadth First Search
10. Shortest path tree
11. Dijkstra's Algorithm

## Queue (Completed)

<https://www.geeksforgeeks.org/queue-data-structure/>

Like [Stack](http://quiz.geeksforgeeks.org/stack-set-1/), [Queue](http://en.wikipedia.org/wiki/Queue_%28data_structure%29)is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO).  A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.  
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

**Operations on Queue:**  
Mainly the following four basic operations are performed on queue:

**Enqueue:**Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.  
**Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.  
**Front:**Get the front item from queue.  
**Rear:** Get the last item from queue.

## Binary Max Heap – Done in Javascript

What is a Binary Max Heap

Maintain the following invariants

Use an array to implement to binary max heap.

Any new items will be added to the right most leaf in the tree.

If the item added is smaller or equal to parent, do nothing as invariant is maintained. If item added is larger, then sift up until it can’t be sifted up anymore.

Operations

GetMax – Only return the max

InsertNode - Insert Node into right most leaf in the tree and sift up

Extract Max – Extract the max item by replacing last bottom leaf (right most leaf) with the root and then cutting the max item. Then sift down the root node until it gets to the right place. Be sure that the function returns the new state of the array.

Change Priority – if priority is increasing, then check if the priority is higher than the parent. If it is, then sift up. If priority is decreasing, then siftdown.

Remove Element – Change the priority of the element to max plus 1. Sift up and then run extract max.

Sift up – get parent and if smaller then swap up

Get parent -

## Stack – Done implemented in Java

1. Push (key) – Adds the key to the collection and return the key
2. Top(): return most recently added key
3. Pop() remove and return most recently added key
4. Empty() return whether there are any elements in the stack

Problem

Implement bracket matching solution using a stack

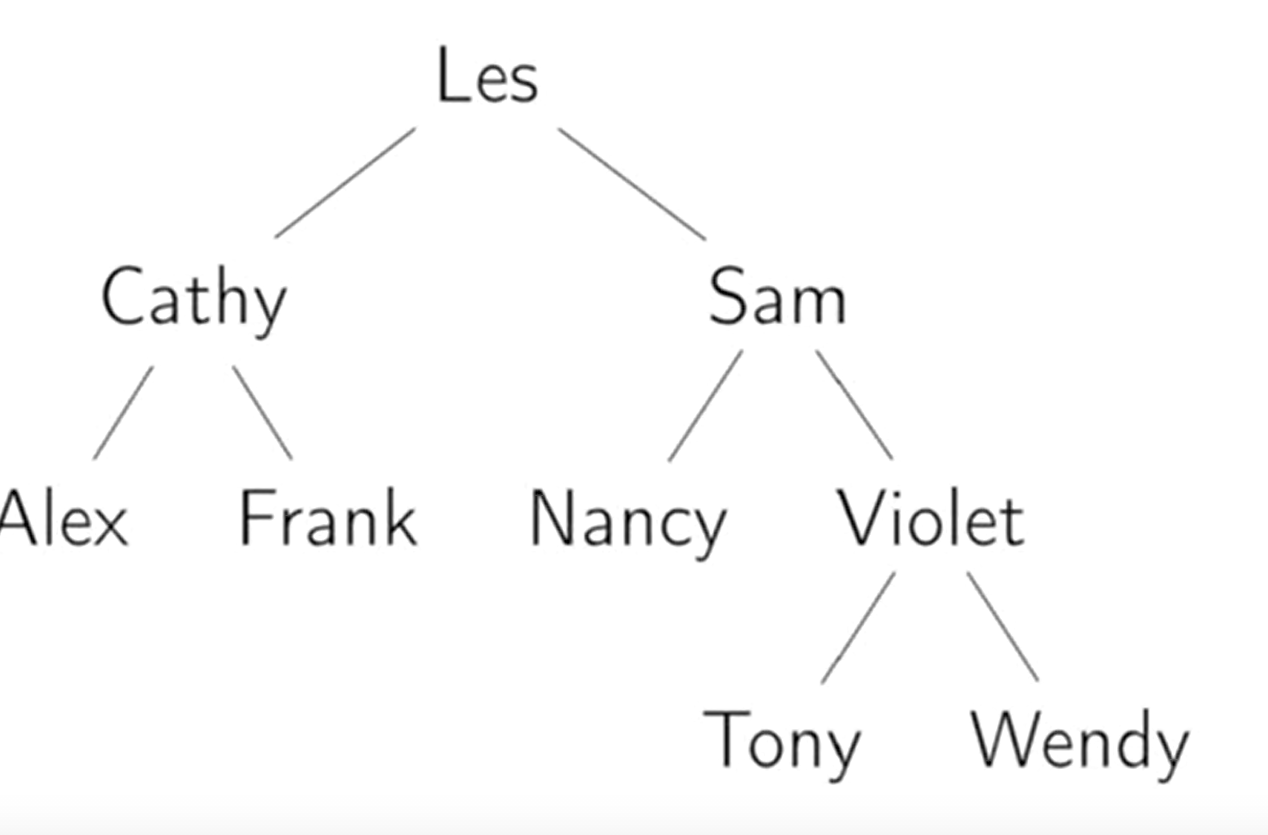
[]{{()}} is balanced etc.

## Binary Search Tree

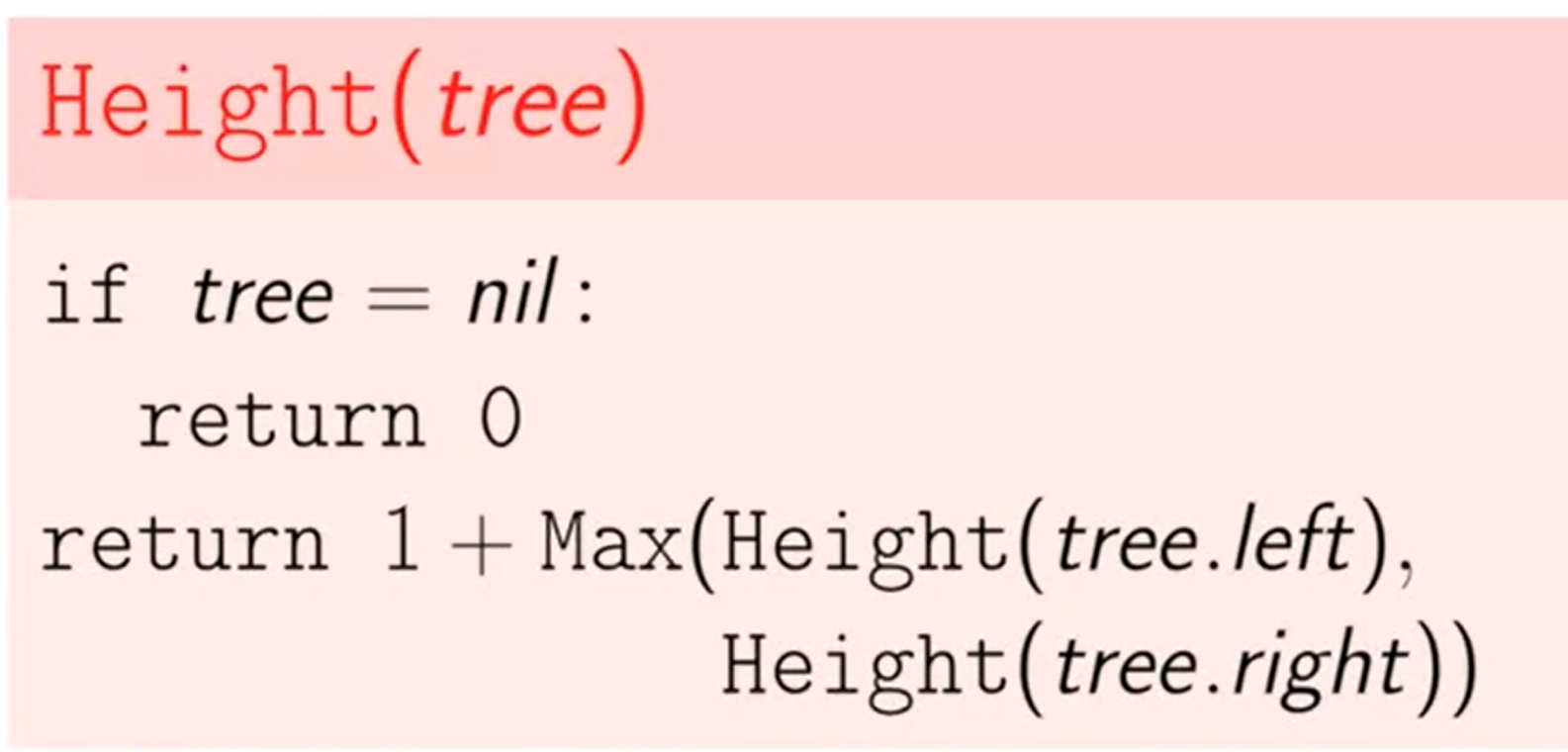
1. Create a simple hello world console application.
2. Create a binary search tree class, implement stub functions and import it in the console application.
3. Create a new instance of the BST class.
4. Create unit tests for all BST functions and show them failing

The following is the definition of Binary Search Tree(BST) according to [Wikipedia](http://en.wikipedia.org/wiki/Binary_search_tree)  
Binary Search Tree is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree.   
  There must be no duplicate nodes.



getHeight – Get the tree height.

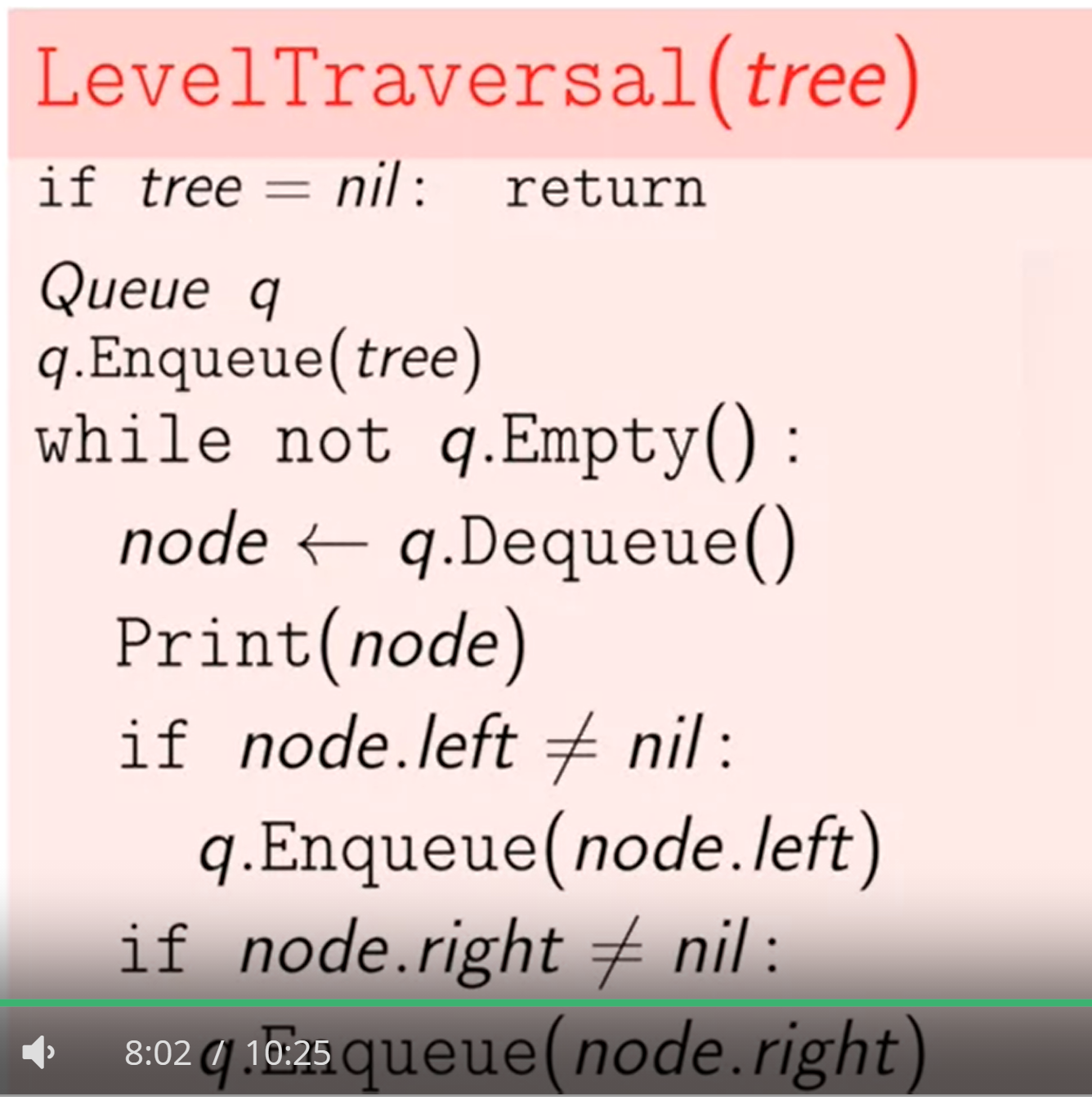


AddNode(value) – Traverse tree and add to a leaf

DeleteNode(value) – Tricky. Watch video again – Remove as last leaf is easy. Remove a node with one child is easy, just attach the child to the parent. In order to remove a node with two children, find the min in the right subtree of that node. That node will not have a left child. Replace the node value with the min node value and delete the one child min node.

NodeExists(value) – Similar algo as AddNode

BreadthFirstSearch –



InOrderTraversal – Recursive Traversal of left first, then print node and the right child

PreOrderTraversal – Recursively visit the node, then the left tree, then the right tree.

PostOrderTraversal – Recursively visit the left tree, the right tree and then the node.