- Big O Notation
  - Constant time ($O(1)$) complexity
  - Linear Time ($O(n)$) complexity
  - Quadratic Time ($O(n^2)$) complexity
  - Logarithmic Time ($O(\log n)$) complexity
- Validating Algorithms - Deterministic Algorithm - Randomized Algorithm - Exact Algorithm - Approximate Algorithm
- Data Structures - Stacks - Queues - Trees - Types of Trees.
- Sorting and Searching Algorithms.

Big O Notation

Used to quantify the performance of various algorithms as input size grows.

Constant time ($O(1)$) complexity

An algorithm that takes the same amount of time to run independent of the size of the input data

```
def getFirst(my_list):
    return my_list[0]
```

The function `getFirst` will always take the same amount of time to retrieve the first element in a passed list

Linear Time ($O(n)$) complexity

An algorithm whose execution time is directly proportional to the size of the input

```
def getSum(my_list):
    sum = 0
    for item in list:
        sum += item
    return sum
```

Notice that in the main loop of the function `getSum` the number of iterations it performs increases linearly with an increasing value of n resulting in $O(n)$ complexity

Quadratic Time ($O(n^2)$) complexity

An algorithm whose execution time is proportional to the square of the input size

```
def getSum(my_list):
    sum = 0
    for row in my_list:
         for item in row:
            sum += 0
    return sum
```

Note that the nested inner loop within the other main loop gives this code a complexity of $O(n^2)$

Logarithmic Time ($O(\log n)$) complexity

An algorithm whose execution time is proportional to the logarithm of the input size, i.e.with each iteration the input size decreases by a constant multiple factor.

```
def searchBinary(my_list, item):
    first = 0
    last = len(my_list) - 1
    foundFlag = False
    while first <= last and not foundFlag:
        mid = (first + last) // 2
        if my_list[mid] == item:
            foundFlag = True
        else:
            if item < my_list[mid]:
                last = mid - 1
            else:
                first = mid + 1
    return foundFlag
```

Validating Algorithms

Deterministic Algorithm

A particular input always generates the exact same output

Randomized Algorithm

A random sequence of numbers is taken along with an input which alters the output every time the algorithm is run.

Exact Algorithm

Algorithms expected to produce a precise solution without introducing any assumptions or approximations

Approximate Algorithm

Algorithms that deal with a complex problem by making assumptions.

Data Structures

Stacks A linear data structure to store a one dimensional list. I can store items in either Last-in, First-out (LIFO) or First-In, Last-Out (FILO).

```
class Stack:
    def __init__(self) -> None:
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items) - 1]

    def size(self):
        return len(self.items)
```

Queues Stores n elements in a single dimensional structure. Elements are added and removed in First-in, First-out FIFO format. A queue has a rear and a front.

- Dequeue: Item is removed from the front of the queue.
- Enqueue: Item is added to the rear of the queue

```
class Queue(object):
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0, item)
```

```
10
11      def dequeue(self):
12          return self.items.pop()
13
14      def size(self):
15          return len(self.items)
```

Trees A data structure used to represent hierarchical relationships among data elements that need to be stored or processed.

- Node: An element in the tree.
- Root: Starting data element of a tree/ A node with no parent.
- Level of a node: Distance between a node and the root.
- Sibling nodes: Two nodes at the same level in the tree.
- Child node: A node is said to be a child node in relation to another if it directly linked to that node and it's node level is less than the proposed node.
- Parent node: A node is said to be a parent node in relation to another if it directly linked to that node and it's node level is greater than the proposed node.
- Degree of a node: The number of children a node has.
- Branches: Links between nodes in the tree.
- Degree of a tree: The greatest number of children a node in the tree has.
- Subtree: A section of a tree with a chosen node acting as a root node and all the preceding children of this node acting as the nodes of the new tree.
- Leaf: A node with no children.
- Internal node: Any node that is neither a root node or a leaf node, i.e. having at least one parent and one child.

Types of Trees.

- Binary Trees: A tree with a degree of two.
- Full tree: A tree with all nodes having either two or no children.
- Perfect tree: A tree with all leaf nodes being of the same level.
- Ordered tree: A tree in which the children of a node are organized in order according to some criteria, for example, left to right in ascending order in which nodes at the same level will increase in value while moving from left to right.

Sorting and Searching Algorithms.