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
```