

---

## 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

```
1 def getFirst(my_list):  
2     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

```
1 def getSum(my_list):  
2     sum = 0  
3     for item in list:  
4         sum += item  
5     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

```
1 def getSum(my_list):  
2     sum = 0  
3     for row in my_list:  
4         for item in row:  
5             sum += 0  
6     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.

```
1 def searchBinary(my_list, item):
2     first = 0
3     last = len(my_list) - 1
4     foundFlag = False
5     while first <= last and not foundFlag:
6         mid = (first + last) // 2
7         if my_list[mid] == item:
8             foundFlag = True
9         else:
10            if item < my_list[mid]:
11                last = mid - 1
12            else:
13                first = mid + 1
14    return foundFlag
```