***Big O Notation:***

This concept gives you one way of describing how the time it takes to run a function grows as the size of input grows.

given\_array = [1, 4, 3, 2,…,10]

**def findSum**(given\_array):

total = 0

**for each i in given\_array:**

total += i

**return total**

**How much time it takes to run the function?**

**How does the runtime of this function grow?**

To answer these question we use Big O Notation and Time Complexity.

In the above when you increase the number of elements in an array the time also increases i.e, when the elements of array are 5 it takes X microseconds, when the elements are increased to 10 it taks around 2X microseconds. By this we can say that time increases linearly with the elements.

This pattern is called linear time complexity.

There are many different pattern such as constant time complexity i.e, time does not increases with the elements.

Quadratic time complexity is where graph of time rises drastically with increased elements.

**Linear time complexity**: O(n)

**Constant time complexity**: O(1)

**Quadratic time complexity**: O(n**2**)

How to calculate complexity?

Step 1: To find the fastest growing term in a given function.

Eg: T = aN +b

Here b is constant so aN is the fastest growing term.

Step 2: Take out the coefficient i.e, N

* O(n)