Big O Notation and Time and Space Complexity

# What is Big O notation and time and space complexity

Big O Notation is a method used to measure an algorithm’s efficiency. Time complexity is measured in Big O and measures the amount of time and algorithm takes to run. Space complexity is also measured in Big O and measures the amount of space on algorithm takes based on its input size.

**NOTE:** When finding the time complexity or space complexity using Big O notation, constants and coefficients are ignored in an expression. For example, the expression 6n + 4 in Big O is O(n) because the coefficient of 6n – 6 – is ignored and the constant of 4 is removed.

# Different types of time complexity:

## O(1) – Constant time and space complexity

### For time

This means that an algorithm usually takes a constant amount of time to run despite the input size.

Example (in python):

arr1 = [“h”, “a”, “o”]

print(arr[1])

The above program has an O(1) time complexity. This is because arrays have contiguous memory; each block of memory is consecutive and can be accessed by a user. This makes accessing an element in an array an arithmetic operation and arithmetic operations are of O(1) time complexity.

### For space

This means the amount of memory used when running an algorithm is constant

Example (in python):

def sum(a, b):

return a + b

For ref: 1 byte = 8 bits

In python, a single digit integer uses 4 bytes (32 bits). This function will always have O(1) space

complexity.

## O(n) – Linear time and space complexity

### For time

This means that the amount of time an algorithm takes to process an input is directly proportional to the size of the input.

Example (in python):

for i in range(10):

print(i)

Above, is a for loop which iterates through a list of values **linearly.**

### For space

This means the amount of memory used when running an algorithm increases linearly.

Example (in python):

def total\_of\_arr(arr):

length = len(arr)

sum = 0

for i in range(len(arr)):

sum += arr[0]

return sum

The code above finds the sum of all the values in an integer array without using the sum function. When addressing a problem, we must first find all the variables along with the amount of memory which each variable takes up.

arr – 4n space since n is the length of the array and since the array is an integer array, each value in the value is a 4-byte integer.

length – a 4 byte integer

sum – a 4 byte integer

i (the iterator) – a 4 byte integer

Now, we have the expression, 4n + 4 + 4 + 4 (bytes). If we evaluate this expression to Big O form, we get rid of the constant values so the 3 fours are removed and the coefficient of 4 is ignored and we result in a value of O(n).

## O(n2) – Quadratic time and space complexity

### For time

This means that the amount of time an algorithm takes to process an input increases quadratically – the greater the input, the less efficient the algorithm is because the greater the value of ‘n’ in O(n2) the greater the output value of n2.

Example (in python):

for i in range(10):

for j in range(10):

print(str(i) + “ ”+ str(j))

The program above uses a for loop in a for loop. This means one loop iterates linearly through a

range of values in another loop which also iterates linearly through a range of values. This means

each loop individually is of O(n) time complexity; however, since the second loop runs in the first

loop, the time complexity is O(n2)

### For space

This means the amount of memory used when running an algorithm increases quadratically.

O(log n) – Logarithmic time and space complexity

### For time

This means that the amount of time an algorithm takes to process an input, increases extremely slowly in relation to the input size, as shown in the graph below. O(log n) has a default base of 2, and therefore is used in binary search algorithms.

### For space

This means the amount of memory used when running an algorithm increases logarithmically with a base of 2.

