Idea of Algorithmic Efficiency In Python

Algorithm: **An Algorithm is a method or procedure for accomplishing a specific task. In other words the algorithm is a systematic collection of steps which used to solve any problem and these steps can be implement in programming language-Python. The algorithm should be efficient and effective in nature.**

Factors affect the Algorithm:

The quality or performance of an algorithm is depends on many internal and external factors.

1. **Internal factors: These factors specify the efficiency of algorithm in terms of….**

- **Time required to run**
- **Space (Memory) required to run**

These internal factors are studied and measured in order to determine the efficiency or complexity of algorithm.

2. **External factors: These factors affect the performance of algorithm. It includes following factors…**

- **Size of input to algorithm**
- **Speed of computer**
- **Quality of Compiler**

The external factors are controllable to some extent.

Computational Complexity

Computation involves the "problem to solve" and "algorithm to solve that problem". Where complexity involves the study of factors to determine, how much resource is sufficient for performance of used algorithm to solve the problem?

Depending on the resources there are two types of complexity

1. **Temporal Complexity (Time needed to run algorithm.)**
2. **Space Complexity (Memory/Storage needed to run algorithm)**

Efficiency Vs Effectiveness

Effectiveness mean that the algorithm carried out its desired output correctly.

Efficiency mean that the algorithm should be correct with the best possible performance for all types of inputs.

Estimate Complexity: Big-O Notation

To every algorithm there may be three cases for producing output or its performance.

A. Worst–Case
B. Average-Case
C. Best-Case

Big-O Notation is simplified analysis of an algorithm's efficiency. It depicts the growth rate of an algorithm. The growth rate determine the performance of algorithm when its input size grow. Big –O notation is very useful in comparing the performance of two or more algorithm. Complexity estimate in terms of following factors.

A. In term of Input size, N
B. Machine – Independent (Machine configuration)
C. Basic Computer Steps (Logic used in algorithm)
D. Time and space

Calculate Complexity for Loops:

Running time of a loop is approximately equal to the "Running time of body statements of loop multiplied by number of iteration"

Loop executes n times.

Example:

```
for I in range(n):
    sum=sum+I
```

All statements in this loop takes constant time. It is C

So total time taken by the execution of loop is:

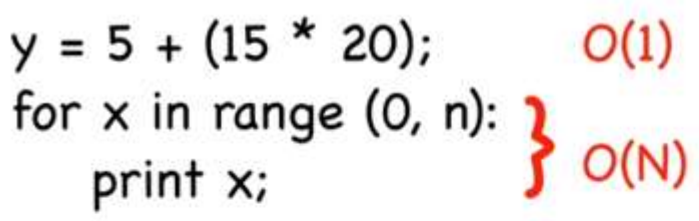
Total Time= C * n = Cn   i.e. O(n)

Where..

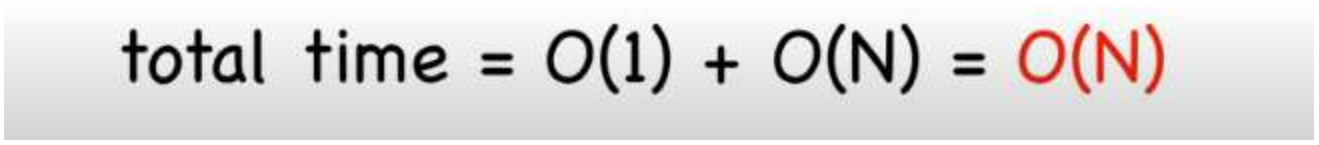
C is time taken for one iteration.

n is total number of iterations.

Cn / O(n) is total time taken for all iterations.

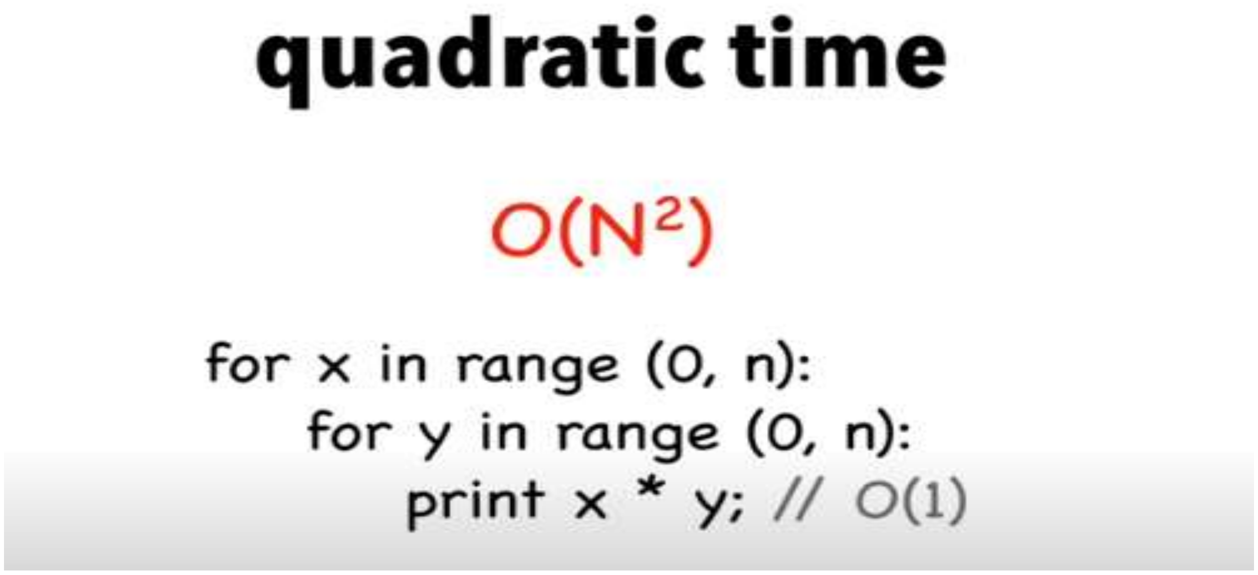


```
y = 5 + (15 * 20);              O(1)
for x in range (0, n):  }
    print x;            }       O(N)
```



total time = O(1) + O(N) = O(N)

Calculate Complexity for nested Loops:



quadratic time

$O(N^2)$

```
for x in range (0, n):
    for y in range (0, n):
        print x * y; // O(1)
```

Explanation:

Outer Loop executes n times.

Inner Loop executes n times.

```
for X in range (n):
    for Y in range(n):
        print(X * Y )
```

All statements in this loop takes constant time. It is C

So total time taken by the execution of nested loop is:

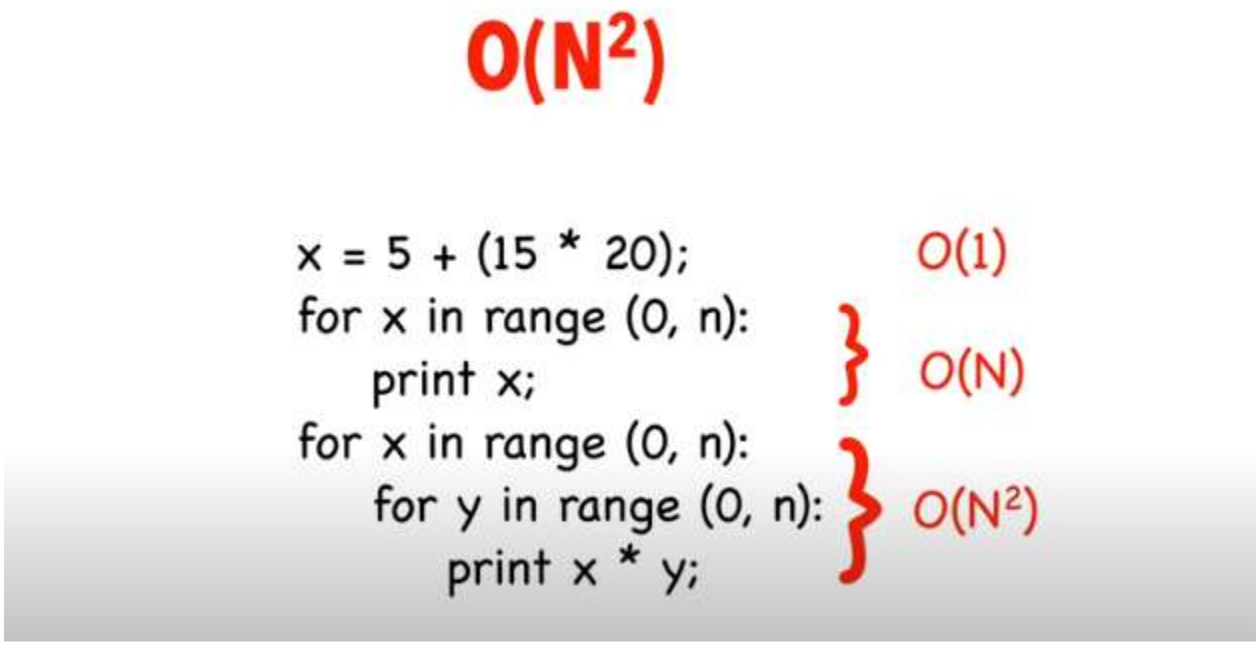
Total Time= C * n * n = Cn^2 i.e. $O(n^2)$

Where..

C is time taken for one iteration.

n^2 is total number of iterations(both for inner & outer loop).

Cn^2 / $O(n^2)$ is total time taken for nested loop.



```
x = 5 + (15 * 20);
for x in range (0, n):
    print x;
for x in range (0, n):
    for y in range (0, n):
        print x * y;
```

Explanation:

Total Time = O(1) + O(N) + O(N^2)

That Mean Total Time = O(N^2)

{Note: - Considering only dominant term, i.e. N^2}

Calculate Complexity for if-else:

To computer the complexity of if-else statement, we consider the worst case running time. Which mean we consider the total time as given below

Time taken by test + time taken by either of block or else part.

(Whichever is larger)

Example:

```
if (A>B):                       # takes time as constant Co
    return False                # takes time as constant C1
else:
    for I in range(n):          # for loop runs n times
        if(A<B):                # takes time as constant C2
            return False        # takes time as constant C3
```

Time taken by else part = (C2+ C3) * n

Total time Taken: C0 + C1 + (C2 + C3) * n   that is O(N)

Some Important points:

- **Constants matters in performance of algorithm.**
- **Be cognizant of best case or average case.**