Top of Form

**ALGORITHM**

**Informal Definition:**

An Algorithm is any well-defined computational procedure that takes some value or set of values as Input and produces a set of values or some value as output. Thus, algorithm is a sequence of computational steps that transforms the i/p into the o/p.

**Formal Definition:**

An Algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms should satisfy the following criteria.

1. INPUT : Zero or more quantities are externally supplied.
2. OUTPUT : At least one quantity is produced.
3. DEFINITENESS : Each instruction is clear and unambiguous.
4. FINITENESS : If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
5. EFFECTIVENESS : Every instruction must very basic so that it can be carried out, in principle, by a person using only pencil &paper.

**Issues or study of Algorithm:**

* How to device or design an algorithm --> creating and algorithm.
* How to express an algorithm --> definiteness.
* How to analysis an algorithm --> time and space complexity.
* How to validate an algorithm --> fitness.
* Testing the algorithm --> checking for error.

**Algorithm Specification:**

Algorithm can be described in three ways.

**1) Natural language like English:**

We should ensure that each & every   statement is definite.

**2) Graphic representation called flowchart:**

This method will work well when the algorithm is small& simple.

**3) Pseudo-code Method:**

In this method, we should typically describe algorithms as program, which resembles language like Pascal & algorithm.Bottom of Form

Top of Form

**Algorithm Complexity**

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

·       **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.

·       **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

**Space Complexity**

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

·       A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.

·       A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain the concept −

Algorithm: SUM(A, B)

Step 1 - START

Step 2 - C ← A + B + 10

Step 3 - Stop

Here we have three variables A, B, and C and one constant. Hence S(P) = 1 + 3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

**Time Complexity**

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem. While analyzing an algorithm, we mostly consider time complexity and space complexity. Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Similarly, Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Time and space complexity depend on lots of things like hardware, operating system, processors, etc. However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm.

Let’s start with a simple example. Suppose you are given an array A and an integer x and you have to find if x exists in array A.

Simple solution to this problem is traverse the whole array A and check if the any element is equal to x.

for i : 1 to length of A

if A[i] is equal to x

return TRUE

return FALSE

Each of the operation in computer take approximately constant time. Let each operation takes c time. The number of lines of code executed is actually depends on the value of x. During analyses of algorithm, mostly we will consider worst case scenario, i.e., when x is not present in the array A. In the worst case, the if condition will run N

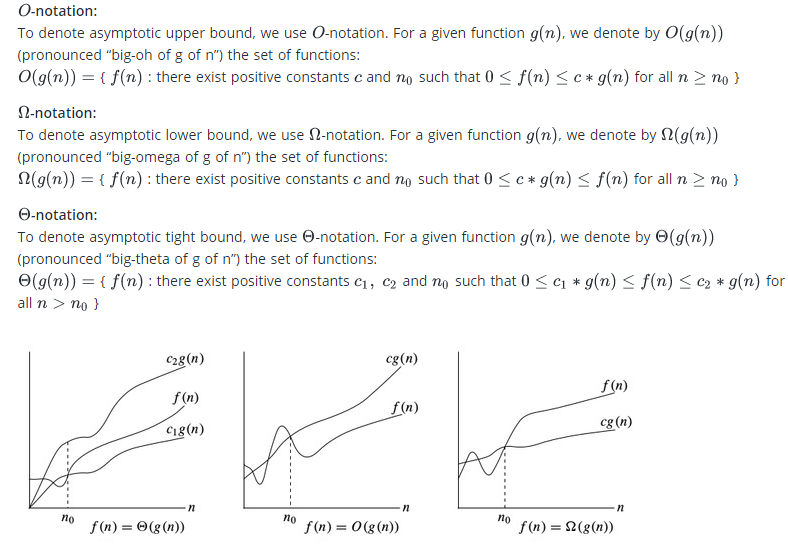
times where N is the length of the array A. So in the worst case, total execution time will be (N∗c+c). N∗c for the if condition and c for the return statement (ignoring some operations like assignment of i .

As we can see that the total time depends on the length of the array A. If the length of the array will increase the time of execution will also increase.

**Order of growth:**  is how the time of execution depends on the length of the input. In the above example, we can clearly see that the time of execution is linearly depends on the length of the array. Order of growth will help us to compute the running time with ease. We will ignore the lower order terms, since the lower order terms are relatively insignificant for large input. We use different notation to describe limiting behavior of a function.

Bottom of Form

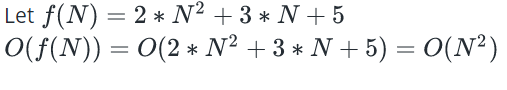
Top of Form

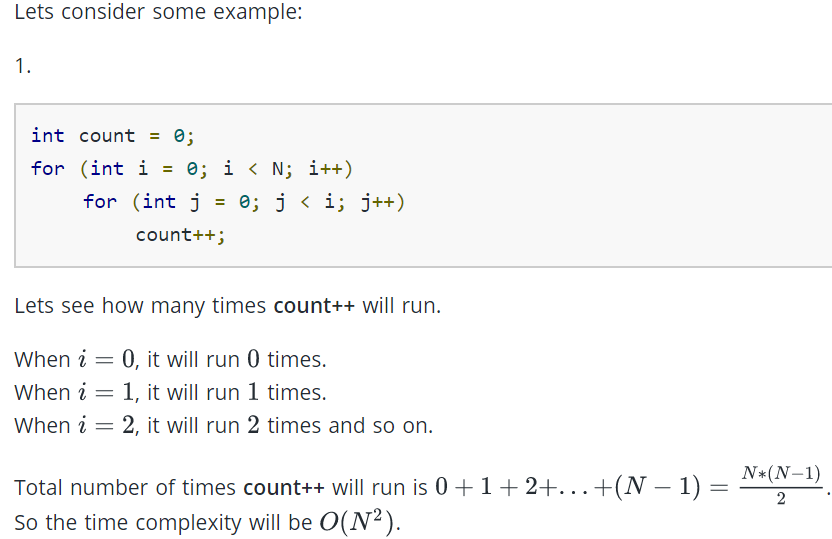


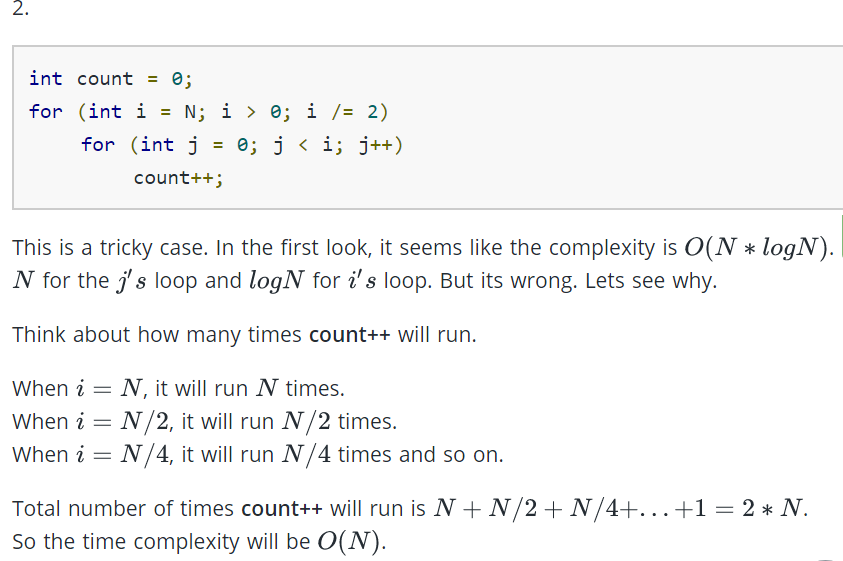
**Time complexity notations**

While analyzing an algorithm, we mostly consider O-notation because it will give us an upper limit of the execution time i.e. the execution time in the worst case.

To compute O-notation we will ignore the lower order terms, since the lower order terms are relatively insignificant for large input.







**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

* 1. <http://119.235.48.116/POC/CMRITCSE/A.Y-2018-19/II/II-II/DAA/DAA_UNIT-1.pdf>
  2. <https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/>

**Lecture Video:**

1. <https://youtu.be/0IAPZzGSbME>
2. <https://youtu.be/FbYzBWdhMb0>
3. <https://youtu.be/xGYsEqe9Vl0>
4. <https://youtu.be/9TlHvipP5yA>
5. <https://youtu.be/9SgLBjXqwd4>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**