# ***Time Complexity of a Computer Program***

* Notes based on YouTube video [Time complexity of a computer program](https://www.youtube.com/watch?v=V42FBiohc6c&list=PL2_aWCzGMAwI9HK8YPVBjElbLbI3ufctn)

What is Time Complexity ?

Let us look at the problem of checking if a number n is Prime or not. Below are two solutions for the problem.

|  |  |
| --- | --- |
| Algorithm 1 | Algorithm 2 |
| void is\_prime(int n) {  for i <- 2 to n-1 {  if n%i == 0 {  n is not prime  return  }  }  n is prime  return  } | void is\_prime(int n) {  for i <- 2 to sqrt(n) {  if n%i == 0 {  n is not prime  return  }  }  n is prime  return  } |
| Time Complexity : | Time Complexity : |

In algorithm 1, for loop runs times in the worst case and in algorithm 2, for loop runs times in the worst case.

Assuming that division operation takes 1 ms, time of execution, T, for the algorithms for different values of n will be as below:

|  |  |  |
| --- | --- | --- |
| n | T(Algorithm 1) in ms | T(Algorithm 2) in ms |
| 11 | 9 | 2 |
| 101 | 99 | 9 |
| 1000003 | 1000001 | 1000 |

For large values of n, time taken by algorithm 1 is much more than that for algorithm 2.

Time complexity is a simple measurement of how fast the time taken by a program increases if n grows. Time complexity of algorithm 1 and algorithm 2 are and respectively. The notation used is called “big oh” and will be explained later.

How to calculate running time ?

Some of the factors affecting actual running time of an algorithm are :

1. Single vs Multiple processors
2. Speed of Read/Write to memory
3. Processor architecture and other hardware configurations
4. Input

Time complexity analysis does not consider first three factors. It mainly deals with rate of growth of time taken by the program with respect to the input.

Time complexity analysis :

The following assumptions are made about the machine for the purpose of measuring time complexity :

* single processor
* 32 bit
* sequential execution
* 1 unit time for arithmetical and logical operations
* 1 unit time for assignment and return

|  |  |  |  |
| --- | --- | --- | --- |
| **Program** | **Cost** | **#Times executed** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Sum(a, b){  1.return a+ b  } | c = 2 | m = 1 |  |
| SumOfList(A, n) {  1.total = 0  2.for i <- 0 to n-1 {  3. total = total + A[i]  }  4.return  } | = 1  = 2  = 2  = 1 |  |  |

= K

Note : has term because the program will have 2 for loops and the inside for loop will be executed times.

Asymptotic Notations - Big Oh, Theta, Omega

The big-oh notation (O) that we saw above is an asymptotic notation. Theta and Omega () are other notations used. It is used to classify the running time of an algorithm into generic and broad classes or sets.

Suppose :

Algorithm 1 :

Algorithm 2 :

As , both algorithms will have a quadratic rate of growth and the values of coefficients and terms with lower powers of n will become insignificant.

Big-Oh (O) Notation

The Big-Oh notation gives the upper bound of the rate of growth of a function.

For example, suppose

If c = 8 and ,as can never be greater than .

Big-Omega () Notation

The Big-Omega () notation gives the lower bound of the rate of growth of a function.

For example, suppose

If c = 5 and ,as will be at least 1.

Theta () Notation

The Theta () notation gives the tight bound of the rate of growth of a function.

For example, suppose

If = 5, = 8 and ,

Time Complexity Analysis : General Rules

We analyze time complexity for :

1. Very large input size
2. Worst case scenario

Rule 1 :

1. Drop lower order terms
2. Drop constant multiplier

Eg.:

Rule 2 :

Eg.:

|  |
| --- |
| function() {  1.int a;  2.a = 5;  3.a++;  4.for(int i = 0; i < n; ++i) {  5.// simple statements  6.}  7.for(int i = 0; i < n; ++i) {  8.for(int j = 0; j < n; ++j) {  9.//simple statements  10.}  11.}  } |

|  |  |  |
| --- | --- | --- |
| T(Fragment 1 -> Lines 1 to 3)  = O(1) | T(Fragment 2 - > Lines 4 to 6)  = O() | T(Fragment 3 - > Lines 7 to 11)  = O() |

Rule 3 :

Eg.:

|  |
| --- |
| function() {  1.if{  2.for(int i = 0; i < n; ++i) {  3.// simple statements  4.}  5.}  6.else{  7.for(int i = 0; i < n; ++i) {  8.for(int j = 0; j < n; ++j) {  9.//simple statements  10.}  11.}  } |

|  |  |
| --- | --- |
| T(Fragment 1 - > Lines 2 to 4)  = O() | T(Fragment 2- > Lines 7 to 11)  = O() |

Rule 4 :