RATE OF GROWTH

- The ***Rate Of Growth*** helps to find the behavior of an ***Algorithm***.
- The rate at which the running time increases as a function of input is called ***Rate Of Growth***.
- The ***Rate Of Growth Of An Algorithm*** is the rate at which its running time increases as a function of input '***n'***.

Example 1:

Let us Assume that a boy went to a shop to buy a car and a bicycle. If his friend sees him there and asks that what he is Buying? Then in general the boy will reply: "I am buying a car." This is because the cost of the car is high compared to the cost of the bicycle (approximating the cost of the bicycle to the cost of the car).

$$Total\ Cost = cost\ of\ the\ car + cost\ of\ bicycle$$

$$Total\ Cost\ \approx cost\ of\ car(approximation).$$

For the above mentioned example , we can represent the cost of the car and the cost of the bicycle in terms of function , and for a given function ignore the low order terms that are relatively insignificant (for a large value of input size , n).

As an example, in the case: $n^4 + 2n^2 + 100n + 500$, where n^4, $2n^2$, $100n$, and 500 are the individual costs of some function $f(n) = n^4 + 2n^2 + 100n + 500$ and approximate to n^4, since n^4 is the highest rate(power or exponent =4 is higher than others). It is written as:

$$n^4 + 2n^2 + 100n + 500\ \approx\ n^4$$

Example 2:

Now this example according to Algorithmic problem.

For two algorithms with run times of 1000 and 1005 seconds, a minor difference can be ignored.

In contrast, if the complexity of one algorithm is significantly larger than that of another, it is a matter of concern. The point is here to decide which is the better algorithm based on how the algorithm will behave when the input size is N is scaled higher (*i.e., N becomes larger and larger*). Thus, the rate of growth of an algorithm is the rate at which its running time increases as a function of input *n*.

Thus , the rate of growth approach can be summarized as follows:

1. Measure the complexity of an algorithm *t(n)* for larger values of *n*.
2. Compare it with the collection of functions that have the same growth rate so that the algorithm can be classified and ranked.

MEASURING LARGER INPUT

Rate of growth is helpful in understanding the behavior of the algorithms. For understanding this concept , let us consider a situation where the time complexity of the algorithms increases as the input size through the following examples.

Example 1

Consider three algorithms A,B and C. Their respective run-time complexity is given as follows: $T_A = 7n$, $T_B = 7\log_{10} n \;\; and \;\; T_C = n^2$. Compare the rates of growth when the input is scaled from n=100 to n = 10,000.

Solution:

Initially, the number of instances is 100. Later it is increased (or scaled) to n = 10,000. For the first algorithm A, whose complexity is given as 7n, the rate of growth is as follows:

$$\frac{T_A(10{,}000)}{T_A(100)} = \frac{7 \times 10{,}000}{7 \times 100} = 100 - - - - i$$

Similarly , the rate of growth for other algorithms can be obtained as follows:

$$\frac{T_B(10{,}000)}{T_B(100)} = \frac{7 \times \log_{10} 10{,}000}{7 \times \log_{10} 100} = \frac{7 \times \log_{10} 10^3}{7 \times \log_{10} 10^2} = \frac{3}{2} = 1.5$$
$$= 2(Approx) - ii$$

$$\frac{T_C(10{,}000)}{T_C(100)} = \frac{10{,}000^2}{100^2} = \frac{(10^4)^2}{(10^2)^2} = \frac{10^8}{10^4} = 10^{8-4} = 10^4 = 10{,}000 - iii$$

It can be observed that algorithm A grows by 100 when input is scaled up from 100 to 10,000. The second algorithm is logarithmic , which grows just by a factor of 2 . The growth of the algorithm is around 10,000. Therefore , the second algorithm is better.

Example 2

Let us consider four Algorithms with the following time complexities:

$$T_1(n) = n, \qquad T_2(n) = (\log_2 n), \qquad T_3(n) = n^2 \, , T_4(n) = 2^n$$

Input Size	n	$\log_2 n$	n^2	2^n
1	1	0	1	2
10	10	3.32	100	1024
100	100	6.64	10^4	$\approx 10^{30}$
1000	1000	9.96	10^6	$\approx 10^{300}$

Here, 2^n can become very large for a larger value of n.