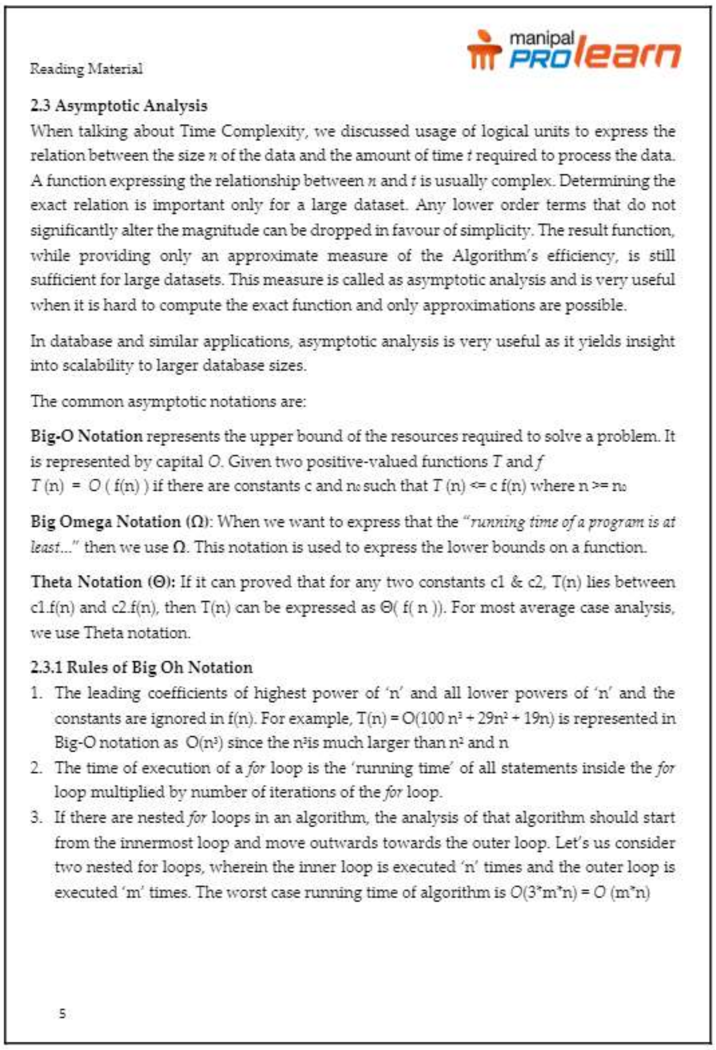

2.3 Asymptotic Analysis

When talking about Time Complexity, we discussed usage of logical units to express the relation between the size *n* of the data and the amount of time *t* required to process the data. A function expressing the relationship between *n* and *t* is usually complex. Determining the exact relation is important only for a large dataset. Any lower order terms that do not significantly alter the magnitude can be dropped in favour of simplicity. The result function, while providing only an approximate measure of the Algorithm's efficiency, is still sufficient for large datasets. This measure is called as asymptotic analysis and is very useful when it is hard to compute the exact function and only approximations are possible.

In database and similar applications, asymptotic analysis is very useful as it yields insight into scalability to larger database sizes.

The common asymptotic notations are:

Big-O Notation represents the upper bound of the resources required to solve a problem. It is represented by capital *O*. Given two positive-valued functions *T* and *f*
$T(n) = O(f(n))$ if there are constants c and n_0 such that $T(n) <= c\,f(n)$ where $n >= n_0$

Big Omega Notation (Ω): When we want to express that the *"running time of a program is at least..."* then we use Ω. This notation is used to express the lower bounds on a function.

Theta Notation (Θ): If it can proved that for any two constants c1 & c2, T(n) lies between c1.f(n) and c2.f(n), then T(n) can be expressed as Θ(f(n)). For most average case analysis, we use Theta notation.

2.3.1 Rules of Big Oh Notation

1. The leading coefficients of highest power of 'n' and all lower powers of 'n' and the constants are ignored in f(n). For example, $T(n) = O(100\,n^3 + 29n^2 + 19n)$ is represented in Big-O notation as $O(n^3)$ since the $n^3$is much larger than n^2 and n
2. The time of execution of a *for* loop is the 'running time' of all statements inside the *for* loop multiplied by number of iterations of the *for* loop.
3. If there are nested *for* loops in an algorithm, the analysis of that algorithm should start from the innermost loop and move outwards towards the outer loop. Let's us consider two nested for loops, wherein the inner loop is executed 'n' times and the outer loop is executed 'm' times. The worst case running time of algorithm is O(3*m*n) = O (m*n)

Views : 26327

370 2