Performance Analysis

- ***Algorithm***
  A clearly specified set of instructions a computer follows to solve a problem.
- **Two criteria are used to judge the performance of algorithms:**
  1. Time Complexity
  2. Space Complexity

Time Complexity

- ***Generally, the amount of time that an algorithm needs to run to completion.***
- **Technically, it is usually measured by the number of**
  - **comparison tests**
  - **assignment statements.**
- **These measures are independent of any particular computer system and depend only the design and implementation of the algorithm.**

The other terms used for this criterion are *computational complexity*, *running time* and *execution time*.

- **It always depends on the amount of input data i.e. *running time* of an algorithm is a function of the input size.**
  If n is the size of input data, then T(n) specifies the running time of the algorithm.

- **Example**
  - **The running time of the algorithm that finds the minimum value in an unordered array is**
    T(n) = n – 1
    where n is the size of the array.
  - **Consider the code fragment**

```
count = 0;
for (i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        count++;
```

The total number of steps is

$$1+\sum_{i=1}^{n+1}1+\sum_{i=1}^{n}\sum_{j=1}^{i+1}1+\sum_{i=1}^{n}\sum_{j=1}^{i}1$$

$$=n+2+\sum_{i=1}^{n}(i+1)+\sum_{i=1}^{n}i$$

$$= n + 2 + \sum_{i=1}^{n}(i+1) + \sum_{i=1}^{n} i \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$= n + 2 + \frac{n(n+1)}{2} + n + \frac{n(n+1)}{2}$$

$$= n^2 + 3n + 2$$

Thus time function for the given code fragment is

$T(n) = n^2 + 3n + 2$

- **Consider the following function**

```
int square(int n) {
    int answer = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            answer++;
    return answer;
}
```

- **The time function for this method is**

$$T(n) = 1 + \sum_{i=0}^{n} 1 + \sum_{i=0}^{n}\sum_{j=0}^{n} 1 + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1$$

$$= 2n^2 + 3n + 2$$

❖**Exercise**

- **Compute the time function for the multiplication of two matrices of order mXn**

Growth Rate of Algorithms

Growth rate of an algorithm tells how quickly the execution time requirement of an algorithm grows as the problem size *n* increases

- **It determines the inherent efficiency of an algorithm independently of such factors as particular computer and implementations**
- **It enables to compare algorithms**
- **Algorithm efficiency is typically a concern for large problems only, and growth rate can help to measure this efficiency.**
- **If algorithm requires time proportinal to *f*(*n*), then algorithm A is said to be of order *f*(*n*), and is denoted as O(*f*(*n*)).**

$f(n)$ is called growth rate function of Algothm A, and the notation O($f(n)$) is called Big-O notation

Examples

- **Running time of an algorithm for finding the minimum value in an ordered array is**

$$T(n) = n - 1$$

When n = 100 T(n) =99 impact of 2^{nd} term is 1%

n = 1000 T(n) =999 impact of 2^{nd} term is .1%

n = 10000 T(n) =9999 impact of 2^{nd} term is .01%

It is clear that the 1st term "n" has the greatest impact i.e. the dominant term is n. So the execution time of the algorithm is proportional to n i.e. growth rate function of the algorithm is f(n) = n and in Big-O notation the order of the algorithm is O(n)

- **Assume that the execution time of an algorithm is**

$$T(n) = 10n^3 + n^2 + 40n + 80$$

When n = 100 T(n) = 10000000 + 14080 = 10014080
Impact of 2^{nd} part is 0.14%

When n = 1000 T(n) = 10000000000 + 1040080
Impact of 2^{nd} part is 0.01%

It shows that the dominant term is $10n^3$. So the execution time of the algorithm is proportional to n^3 . And the algorithm is $O(n^3)$.

- **$T(n) = (n+1)^{1/2} + n + 5$**

  The dominant term is n, so the execution time of the algorithm is proportional to n, and it is O(n).

- **$T(n) = n^{3/2} + n + 5$**

  The dominant term is $n^{3/2}$, so the growth rate function of the algorithm is $f(n) = n^{3/2}$ and it is $O(n^{3/2})$.