Efficiency of Algorithms

Algorithm efficiency

- It is defined as a **function of the number of elements being processed** and **the type of loop being used.**

f(n)=efficiency

- The study of algorithm efficiency therefore focuses on loops.

IMP NOTE

- The efficiency of algorithm **is measured in terms of both time and space** but **inexpensive memory has reduced the significance of space efficiency.**
- **Therefore more emphasis is given on time efficiency.**

Basic concepts of algorithm efficiency for different types of loops:

- **Linear loops:**
  - The efficiency of linear loops is directly proportional to the number of iterations.
  - Efficiency
    - **f(n)=n**
  - In a linear loop, the controlling variable either adds or subtracts.
  - **Eg:**

- Logarithmic loops:
  - In a logarithmic loop, the controlling variable is either multiplied or divided in each iteration.
  - Efficiency
    - **f(n)=logn**
  - **Eg:**

- Nested Loops:
  - Loops that contain loops are called **nested loops.**
  - **Total iterations= outer loop * inner loop**
    iterations iterations
  - 3 nested loops are:
    - Linear logarithmic loop :
    - Quadratic loop:
    - Dependant quadratic loop:

Nested loops

- **Linear logarithmic loop**
  - The generalized efficiency is f(n)=**nlogn**
  - Eg:
- **Quadratic loop**
  - **The number of iterations of the inner loop = The number of iterations of the outer loop**
  - The generalized efficiency is **$f(n)=n^2$**
  - **Eg:**

Nested loops

- **Dependant quadratic loop**
  - **The number of iterations of the inner loop depends on the outer loop.**
  - The generalized efficiency is **f(n)=n(n+1)/2**
  - **Eg:**

Big-O Notation

- **A dominant factor in the equation usually detemines the order of magnitude of the result**.Therefore we don't need to determine the complete measure of efficiency.
- This factor is the big-O as in " on the order of" and expressed as **O(n)- i.e Order of n.**

- Steps in deriving the big-O notation:
  - In each term , set the coefficient of the term to 1.
  - Keep the largest term in the function and discard the others.
- Terms are ranked from lowest to highest as follows:
  - **logN N NlogN N^2 N^3…N^k 2^N N!**

growth-rate functions have the following intuitive interpretations

These are:

- 1
  - The time requirement is constant and,therefore, independent of the problem's size.
- $\log_2 N$
  - The time requirement for a **logarithmic** algorithm increases/decreases slowly as the problem size increases/decreases respectively.
- N
  - The time requirement for a **linear** algorithm increases directly with the size of the Problem.

Cont…

growth-rate functions have the following intuitive interpretations

- $N\log_2 N$
  - The time requirement increases more rapidly than a linear algorithm. Such algorithms usually divide a problem into smaller problems that are each solved separately.
- N^2
  - The time requirement for a **quadratic** algorithm increases rapidly with the size of the problem. Algorithms that use two nested loops are often quadratic.
- N^3
  - The time requirement for a **cubic** algorithm increases more rapidly with the size of the problem than the time requirement for a quadratic algorithm. Algorithms that use three nested loops are often cubic and are practical only for small problems.

growth-rate functions have the following intuitive interpretations

- 2^N
  - As the size of a problem increases, the time requirement for an exponential algorithm usually increases too rapidly to be practical.

Growth Rates Compared

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296

Time complexity analysis

- In time complexity, we analyse for
  - N-> infinity(for large values of n)
  - No constants
- 3 Notations exist:
  - Big O
  - Theta
  - omega

Big O

- In this,
  - f(n)<= c g(n)

    Where c is a constant
- Graph
- In most cases , we use big O notation because it gives the time complexity in the worst case.
- Big O gives the longest time that an algorithm takes for its completion of execution.(worst case)

Theta notation

- In this,
  - c1 g(n)<=f(n)<= c2 g(n)

    Where c1 and c2 are constants and

    n= integer
- Graph
- Theta is the best notation because it is tight bound expression
- Average time an algorithm takes for completing its execution.

Omega notation

- In this,
  - c1 g(n)<=f(n)
  - Where c1 is a constant
- Graph
- Omega notation gives the lower bound of growth of function
- Minimum time an algorithm takes for completing its execution.

Difference between Big O, Omega and theta

- Big O:
  - longest time an algorithm takes for completing its execution.
- Omega:
  - Minimum time an algorithm takes for completing its execution.
- Theta:
  - Average time an algorithm takes for completing its execution.