Geometric Progression →

$$S = a + ar + ar^2 + \dots ar^{n-1}$$

$$S \cdot r = ar + ar^2 + \dots ar^n$$

$$r \cdot S - S = ar^n - a$$

$$\Rightarrow S(r-1) = ar^n - a$$

$$\Rightarrow S = \frac{a(r^n - 1)}{r - 1}$$

→ Execution Time not a good parameter to compare

→ In 1 sec a machine can run 10^8 iterations.

\# of iterations → don't depend on:
1) Machine
2) Programming Language
⋮

\# of iterations ↓ Depends on the algorithm design.

→ Asymptotic → How will an algorithm perform for large inputs.

Time Complexity → Rate of growth of # of iterations w.r.t input size.

↳ = O(# iterations)

Steps to find TC →

1) Calculate # of iterations w.r.t input size.
2) Ignore lower order terms.
3) Ignore constant co-efficient.

Ex: # iterations $= 5n \log n + 80n + 3n^3$

$= O(n^3)$

Common Time Complexities →

$\log n < \sqrt{n} < n < n \log n < n\sqrt{n} < n^2 < n^3 < 2^n < n! < n^n$

- Contribution of lower order terms to time complexity decreases significantly as the input size increases.

→ Problems with O( ) notation →

1) Lower order term can have large contribution.

ex → $f(n) = n^2 + 10^{18} n$

$TC = O(n^2)$

2) Can't compare two algorithms with similar O( ) value.

ex → $f_1(n) = n^2 + 10^{18} n$ | $f(n) = n^2 + 50$

- high value of lower order terms.

→ Importance of constraints →

TC of the algorithm to be used can be foundout from input size constraint.

→ Space Complexity →

Rate of growth of space w.r.t input size.

int → 4 bytes.
long → 8 bytes.
short → 2 1 byte.