# **Complexity Analysis**

**Time Complexity:** Time complexity represents how number of operations will increase of an algorithm for increasing input size.

Reason why we need to calculate time complexity:

* We can compare our different different algorithm or strategy with help of time complexity.
* We can build a better system by analyzing time complexity.

Strategy for calculating Time Complexity:

* Avoid constant
* Avoid lower value
* Find time complexity for worst case scenario.

Example:

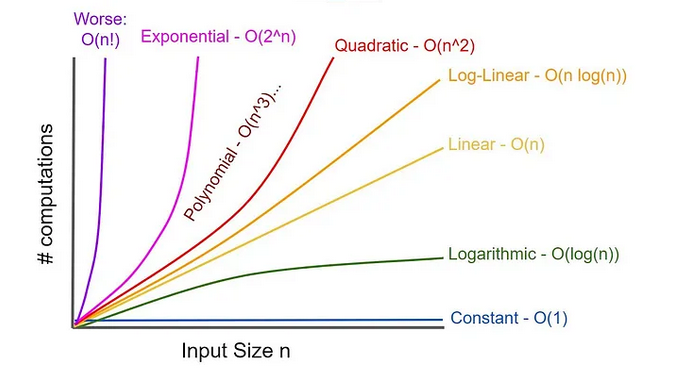
O(5n3+10n2+500)

We need to consider only O(n3) rest will be ignored because of rules.

Some Notations:

1. Omega(Ω): for representing best case/lower bound.
2. Theta(**θ**): for representing average case.
3. Big oh(O): for representing worst case/upper bound.

**Big oh(O):**



1. **Constant or O (1):** when we need to only execute fixed number of operations whatever the input is.   
   Example: printing something, access any index of an array and iterating loop in fixed number of times.
2. **Logarithmic or O (log(n)):** When we are making our array half every time.  
   suppose we have  
    1\*2\*2\*2\*2…x=100  
    =>1\*2x = 100  
    => x=log2(100) or log2(n)  
   Example: Binary Search, Divide and Concore
3. **Linear or O (n):** when we visit every element of an array or looping n times.  
   Example: Linear Search
4. **Log-Linear or O (nlog(n)):** we get this in sorting  
   Example: Merger sort, Heap sort
5. **Quadratic or O (n2):** we get this when we iterate a loop inside another loop where both loops rounding n times.  
   Example: Nested loop, Traversing matrix
6. **Exponential or O (2n):** we see this type of complexity in recursion
7. **Factorial or O(n!**): We see this in Brute force Permutation.

**Problem solving constrains:** By seeing constrains in problems we can assume that what should be the worst complexity to solve the problem.

Here 1s~108 operation

* If n>108 then we need to solve the problem in O(logn) or less.
* If n<=108 then we can think about O(n) or less.
* If n<=106 then we have option to choose O(nlogn).
* If n<=104 then we can think about O(n2).
* If n<=500 then O(n3).
* If n<=25 then we have O(2n)
* If n<=12 then O(n!)