What is algorithm?

18 August 2025 19:09

In mathematics and computer science, an algorithm is a finite sequence of well-defined Computer Implementable instruction typically to solve a class of problems or to perform a computation step by step procedure.

Algorithms are an unambiguous specification for performing calculations, data processing, automated reasoning, and other task.
Will accept zero or more input, but generate at least one output.

Properties of Algorithm

- **Should terminate in finite time.**
- **Unambiguous.**
- **Input 0 or more, output at least one.**
- **Every instruction in algorithm should be effective.**
- **It should be deterministic.**

Difference between Algorithm and Program

Algorithm	**Program**
Written in design phase.	Written in implementation phase.
Needs domain knowledge.	Needs programming knowledge.
Can be written in any natural language.	Can be written in programming language.
Independent of hardware and software.	Dependent of hardware and software.
We analysis algorithm for time and space.	We test programs for faults.

Problem solving cycle

- Problem definition: understand problem.
- Constraints and conditions: Understand constraints, if any.
- Design strategy. (Algorithmic strategy.)
- Express and develop the algorithm.
- Coding.
- Testing and debugging.
- Installation.
- Maintenance.

Need for Analysis

We do analysis of algorithm to do performance comparison between different algorithm to figure out which one is best possible option.

Insertion=>Selection=>Bubble=>Shell(Bucket)=>Merge=>heap=>quick=> and many more.

What parameters can be considered for comparison between cars?
Maruti Suzuki Brezza vs Hyundai Venue

Following are the parameters which can be considered while analysis of an algorithm.

- **Time**
- **Space**
- Bandwidth
- Register
- Battery power

Type of Analysis

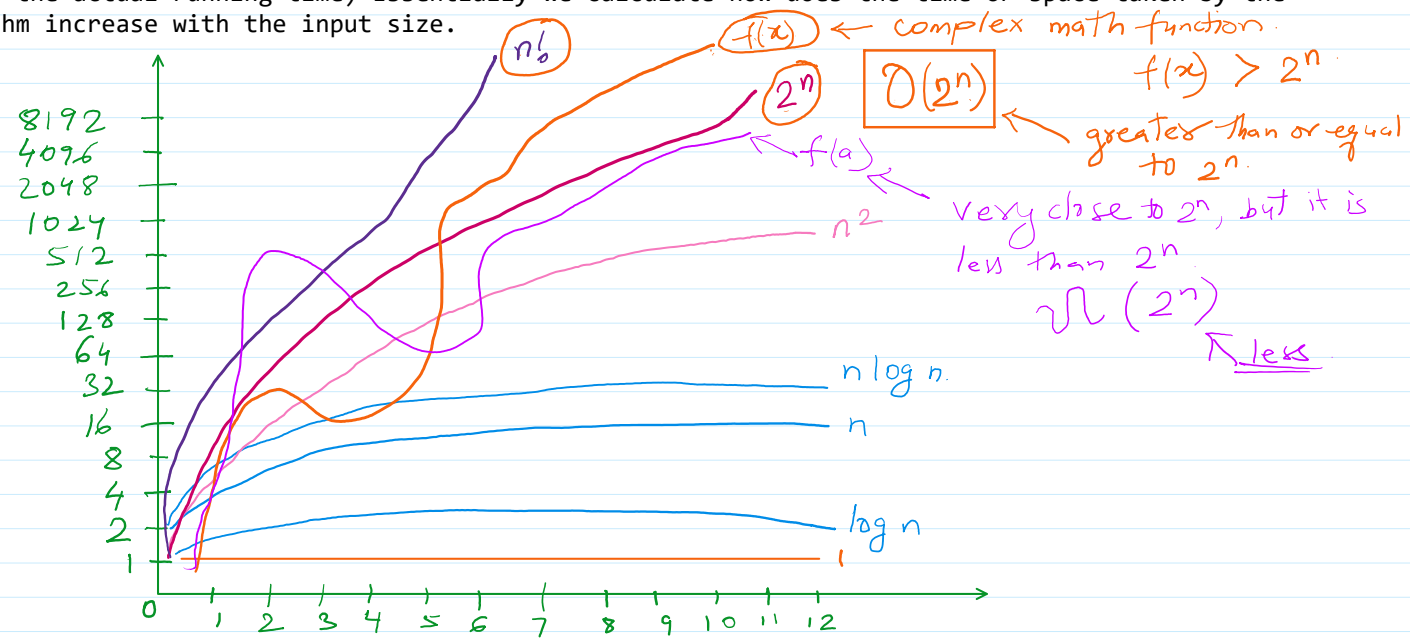
- **Experimental or Apostrium or Relative analysis**
- **Apriori or Independent Analysis or Absolute Analysis**

Experimental or Apostrium or Relative analysis: Means analysis of algorithm after it is converted to code. Implement both the algorithm and run the two programs on your computer for different inputs and see which one takes less time.

Advantage: Exact values, no rough.

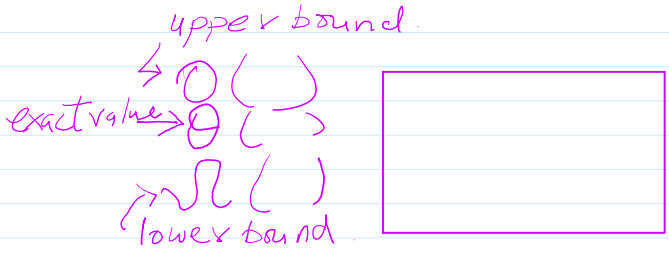
Disadvantage: Final result instead of depending on the algorithm, depends on many other factors like background software and hardware, programming language and even the temperature of the room.
It is expensive.

Apriori or Independent Analysis or Absolute Analysis: We do analysis using asymptotic notations and mathematical tools of only algorithm that is before converting it into the program of a particular programming language.
It is a determination of order of magnitude of a statement.
In asymptotic analysis, we evaluate the performance of an algorithm in terms of input size (We do not measure the actual running time) Essentially we calculate how does the time or space taken by the algorithm increase with the input size.



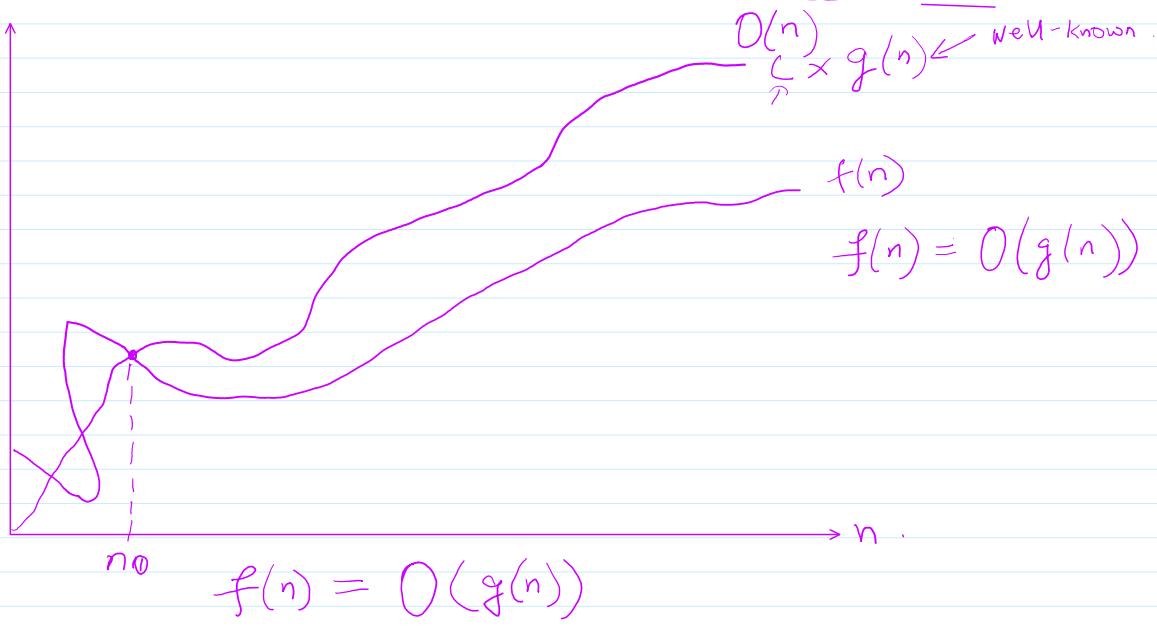
- Asymptotic analysis is not perfect, but that's the best way available for analyzing algorithms.
- It might be possible that those large inputs are never given to your software, and algorithm, which is asymptotically slower always performs better for your particular situation. So you may end up choosing an algorithm that is asymptotically slower but faster for your software.
- **Advantage: Uniform results depends only on algorithm.**
- **Disadvantage:** Estimated or approximated value. No accurate and precise value.

Asymptotic Notations
Asymptotic notations are abstract notations for describing the behavior of algorithm and determine the rate of growth of A function.



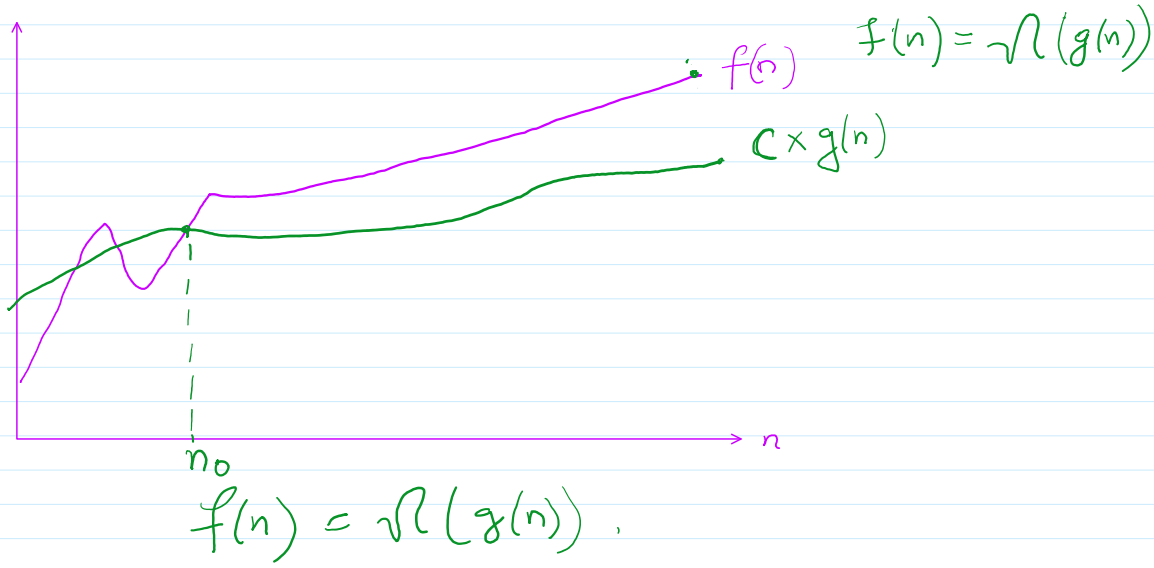
Big O notation

- The big O notation defines an upper bound of an algorithm. It bounds a function only from above.
- The big O notation is useful when we only have upper bound on the time complexity of an algorithm.
- Many times we easily find an upper bound by simply looking at the algorithm.
- O(g(n)) = {f(n): There exist positive constant C and n0 such that 0 <= f(n) <= C*g(n) for all N > N0}



Omega Ω Notation

- Just as we go, notation provides an asymptotic upper bound on a function Omega notation provides an asymptotic lower bound.
- Omega notation can be useful when we have lower bound on time complexity of an algorithm.
- For a given function, g(n) We denote by Ω(g(n)) the set of functions.
- Ω(g(n)) = { f(n): There exist a positive constant C and n0 such that 0 <= C*g(n) <= f(n) for all n>=n0}

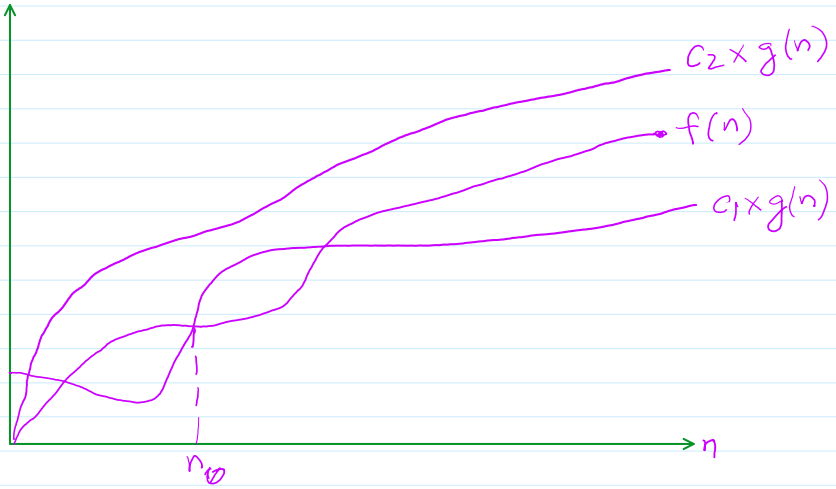


03b8

Theta notation 'θ'

- The Theta notation bounds a function from above and below, so it defines the exact asymptotic behavior.

- For a given function g(n) we denote θ(g(n)) is the following set of functions
- θ(g(n)) = {f(n): There exist a positive constant c1, c2 and n0
  such that 0 <= c1*g(n) <= f(n) <= c2*g(n)
- The above definition means if f(n) is θ(g(n)), then the value f(n) is always between c1*g(n) and c2 *g(n) for large values of n (n >= n0)
- The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.



$f(n) = g(n)$

$f(n) = \Theta(g(n))$

$f(n) = \Theta(g(n))$.