# Algorithms

What is algorithm?

* Algorithm is a step-by-step instruction design to perform a specific task or solve a problem.

What is program?

* It’s implementation of algorithm(s).

Priori Analysis

* It refers to **theoretical evaluation of algorithms based on design without even implementing it.**
* Key points:
  + Done on algorithm.
  + Language independent.
  + Hardware independent.
  + Time and space function is used.

Posteriori Analysis (Empirical Testing)

* It refers to **evaluation of algorithm’s performance based on actual implementation and execution.**
* Key points:
  + Done on program.
  + Language dependent.
  + Hardware dependent.
  + Watch time and bytes.

Characteristics of an Algorithm

* **Input:**
  + Should have 0 or more inputs
* **Output:**
  + Should have at least 1 or more outputs.
* **Definiteness:**
  + Should be clear and precisely defined.
* **Finiteness:**
  + Should execute after finite number of steps properly.
  + Should not continue in definitively.
* **Effectiveness:**
  + It means algorithm and algorithm’s each step is properly defined and also feasible and executable with resources available.

How to write an algorithm?

* Example:
  + Begin
    - temp <- a
    - a <- b
    - b <- temp
  + end
* **<-, :=, =** is used for assignment
* **(Begin, end), ({, })** is used for scope of function
* It has **no declaration or datatype**

How to analyze an algorithm?

* Based on time (it returns time function)
* Space
* Network
* CPU register
* Power

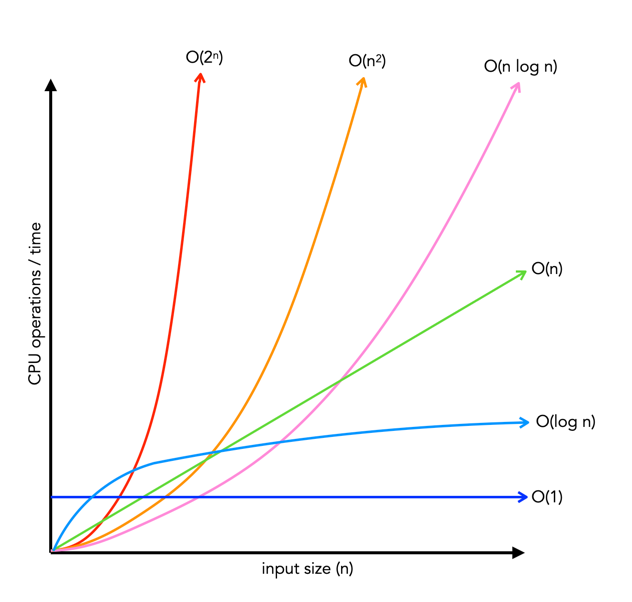
**Examples:**

* Sum of all elements:
  + Algorithm sum(a, n)
  + {
  + Sum = 0 -------🡪 1
  + for (i=0; i<n; i++) {-------🡪 (n+1)
  + Sum += a[i] -------🡪 n
  + }
  + Return sum -------🡪 1
  + }
  + Total time complexity: f(n) = 2n + 3 => n
  + Total space complexity: s(n) = 1 + n + 1 + 1 => n + 3 => n
* Sum of matrices:
  + Algorithm sum(a, b, n)
  + {
  + for(i=0; i<n; i++){ 🡪 n
  + For(j=0; j<n; j++){ 🡪 n \* (n + 1)
  + c[i, j] = a[i, j] + b[i, j] 🡪 n\*n
  + }
  + }
  + Return c
  + }
  + Total time complexity: f(n) = 2n2 + 2n + 2 => n2
  + Total space complexity: s(n) = n2 + n2  + n2 + 2 => n2
* Sum of all elements:
  + Algorithm sum(a, n)
  + {
  + Sum = 0 -------🡪 1
  + for (i=1; i<=n; i++) {-------🡪 (k where k != n && k < n)
  + Sum += a[i] -------🡪 k+1
  + }
  + Return sum -------🡪 1
  + }
  + Total time complexity: f(n) = k(k + 1)/2 where k < n but k2 > n
  + => **k = √n**
* Example:
  + For(i=1; i<n; i=i\*2)
    - Statement
  + i -> 1, 22, 23, 24….
  + 2k = n => **k = log2n => logn**
* Example:
  + For(i=1; i<n; i=i/2)
    - Statement
  + i -> 1, n/2, n/4, n/8….
  + n/2k = 1 => **k = logn**
* Example:
  + While(k<n)
    - K = k+i 🡪 k = 1,2,3,4,5….
    - i++
  + k < n, n = k(k + 1) / 2 => **k = √n**

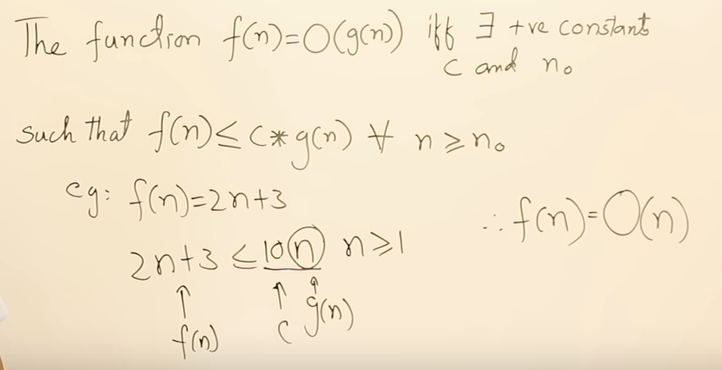
Types of Time functions:

* O(1)
  + Constant
* O(logn)
  + Logrithmic
* O(n)
  + Linear
* O(n2)
  + Quadratic
* O(n3)
  + Cubic
* O(nn)
* O(2n)
  + Exponential

Comparison of Time functions:

* O(n) < logn < √n < n < nlogn < n2  ….< 2n < nn
* 

Asymptiotic Notations

* ***O - Big-Oh (Upper bound)***
  + Gives worst case scenario of a algorithm’s growth rate.
* **Ω - Big-omega (Lower bound)**
  + Gives best case scenario of an algorithm’s growth rate.
* **ϴ - Theta (Average bound)**
  + Gives lower and upper bound of an algorithm’s growth rate i.e. average bound.
* Example:
  + 
  + C, g(n) can be logn, n2, nlogn, n3…. But we should only take closed one i.e. logn
  + Based on bound we’re using we can choose g(n).
  + Let use the example, f(n) = 2n2 + 3n + 4
    - **O(g(n))**
      * 2n2 + 3n + 4 <= c\*g(n), n >= 1
      * Where, c\*g(n) = 9n2
      * O(n2)
    - **Ω(g(n))**
      * 2n2 + 3n + 4 >= c\*g(n), n >= 1
      * Where, c\*g(n) = 1n2
      * Ω(n2)
    - **ϴ(g(n))**
      * 2n2 + 3n + 4 <= c\*g(n), n >= 1
      * Where, c\*g(n) = 9n2
      * ϴ(n2)
  + Another example, f(n) = n2logn
    - O(n2logn), Ω(n2logn), ϴ(n2logn)
  + Another example, f(n) = n! = n\*(n-1)\*(n-2)….
    - O(nn), Ω(1), ϴ(nn)