**What is an Algorithm?**

**A: An algorithm is the step-by-step instruction to solve a given problem.**

Let us consider the problem of preparing an omelet. For preparing omelet, general steps we follow are:

1. Get the frying pan
2. Get the foil
3. Do we have oil?
4. If yes, put it in the pan.
5. If no, do we want to buy oil?
6. If yes, then go out and buy.
7. If no, we can terminate.
8. Turn on the stove, etc…

**Why Analysis of Algorithms?**

**A:** Algorithm analysis helps us determining which of them is efficient in terms of time and space consumed.

For Example: To go from one city to another, there can be many ways of accomplishing this: by flight, by bus, by train and also by bicycle. Depending on the availability and convenience we choose the one that suits us. Similarly, in computer science multiple algorithms are available for solving the same problem.

**Goal of Analysis of Algorithms A:** The goal of analysis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of others factors (e.g, memory, developers’ efforts, etc.)

**What is Running Time Analysis? A:** It is the process of determining how processing time increases as the size of the problem (input size) increases. Input size is the number of elements in the input and depending on the problem type the input may be of different types. The following are the common types of inputs.

* Size of the Array
* Polynomial degree
* Number of elements in a matrix
* Number of bits in binary representation of the input
* Vertices and degree in a graph

**How to Compare Algorithms? A:** To compare algorithms, there are ways to do that:

**Execution times?** *Not a good measure* as execution times is specific to a particular computer.

**Number of statements executes?** *Not* *a good measure, because* the number of statements varies with the programming language as well as the style of the individual programmer.

**Ideal Solution?** Let us assume that we expressed running time of given algorithm as a function of the input size *n. (i.e., f(n) )* and compare these different functions corresponding to running times. This kind of comparison is independent of machine time, programming style, etc.

**What is the Rate of Growth? A:** The rate at which the running time increases as a function of input. It is called as **Rate of Growth.**

|  |  |  |
| --- | --- | --- |
| **Time Complexity** | **Name** | **Example** |
| 1 | Constant | Adding an element to the front of a linked list |
| logn | Logarithmic | Finding an element in an sorted array |
| n | Linear | Finding an element in an unsorted array |
| nlogn | Linear Logarithmic | Sorting n items by ‘divide-and-conquer’-Mergesort |
| n2 | Quadratic | Shortest path between two nodes in a graph |
| n3 | Cubic | Matrix Multiplication |
| 2n | Exponential | The Towers of Hanoi problem |

**Types of Analysis**

To analyze the given algorithm we need to know what input the algorithm takes less time (performing well) and on what inputs the algorithm takes long time.

We have already seen that an algorithm can be represented in the form of an expression. That means we represent the algorithm with multiple expressions: one for the case where it takes less time and other for the case where it takes the more time.

In general the first case is called the ***best case***and second case is called the ***worst case***of the algorithm. To analyze the algorithm we need some kind of syntax and that forms the base for asymptotic analysis/notation.

There are three types of analysis:

* **Worst case**
* Defines the input for which the algorithm takes long time.
* Input is the one for which the algorithm runs the slower.
* **Best case**
* Defines the input for which the algorithm takes lowest time.
* Input is the one for which the algorithm runs the fastest.
* **Average case**
* Provides a prediction about the running time of the algorithm.
* Assumes that the input is random

**Lower Bound <= Average Time <= Upper Bound**

* Proving an upper bound means you have proven that the algorithm will use no more than some limit on a resource.
* Proving a lower bound means you have proven that the algorithm will use no less that some limit on a resource.
* “Resource” in this context could be time, memory, bandwidth, or something else.

**Asymptotic Notation**

Having the expressions for the best, average case and worse cases, for all the three cases we need to identify the upper and lower bounds. To represent these upper and lower bounds we need some kind of syntax and that is we need some kind of notation. Let us assume that the given algorithm is represented in the form of function f(n).