

Lecture 1 - Introduction to Design and analysis of algorithms

What is an algorithm?

Algorithm is a *set of steps to complete a task*.

For example,

Task: to make a cup of tea.

Algorithm:

- add water and milk to the kettle,
- boil it, add tea leaves,
- Add sugar, and then serve it in cup.

What is *Computer algorithm*?

"a set of steps to accomplish or complete a task that is described precisely enough that a computer can run it".

Described precisely: very difficult for a machine to know how much water, milk to be added etc. in the above tea making algorithm.

These algorithms run on computers or computational devices. For example, GPS in our smartphones, Google hangouts.

GPS uses *shortest path algorithm*. Online shopping uses cryptography which uses RSA algorithm.

Characteristics of an algorithm:-

- Must take an input.
- Must give some output (yes/no, value etc.)
- Definiteness – each instruction is clear and unambiguous.
- Finiteness – algorithm terminates after a finite number of steps.
- Effectiveness – every instruction must be basic i.e. simple instruction.

Expectation from an algorithm

- Correctness:-
 - Correct: Algorithms must produce correct result.
 - Produce an incorrect answer: Even if it fails to give correct results all the time still there is a control on how often it gives wrong result. Eg. Rabin-Miller Primality Test (Used in RSA algorithm): It doesn't give correct answer all the time. 1 out of 2^{50} times it gives incorrect result.
 - Approximation algorithm: Exact solution is not found, but near optimal solution can be found out. (Applied to optimization problem.)
- Less resource usage:

Algorithms should use less resources (time and space).

Resource usage:

Here, the time is considered to be the primary measure of efficiency. We are also concerned with how much the respective algorithm involves the computer memory. But mostly time is the resource that is dealt with. And the actual running time depends on a variety of backgrounds: like the speed of the Computer, the language in which the algorithm is implemented, the compiler/interpreter, skill of the programmers etc.

So, mainly the resource usage can be divided into: 1. Memory (space) 2. Time

Time taken by an algorithm?

- performance measurement or Aposteriori Analysis: Implementing the algorithm in a machine and then calculating the time taken by the system to execute the program successfully.
- Performance Evaluation or Apriori Analysis. Before implementing the algorithm in a system. This is done as follows

1. How long the algorithm takes :-will be represented as a function of the size of the input.

$f(n)$ →how long it takes if 'n' is the size of input.

2. How fast the function that characterizes the running time grows with the input size.

“Rate of growth of running time”.

The algorithm with less rate of growth of running time is considered better.

How algorithm is a technology ?

Algorithms are just like a technology. We all use latest and greatest processors but we need to run implementations of good algorithms on that computer in order to properly take benefits of our money that we spent to have the latest processor. Let's make this example more concrete by pitting a faster computer (computer A) running a sorting algorithm whose running time on n values grows like n^2 against a slower computer (computer B) running a sorting algorithm whose running time grows like $n \lg n$. They each must sort an array of 10 million numbers. Suppose that computer A executes 10 billion instructions per second (faster than any single sequential computer at the time of this writing) and computer B executes only 10 million instructions per second, so that computer A is 1000 times faster than computer B in raw computing power. To make the difference even more dramatic, suppose that the world's craftiest programmer codes in machine language for computer A, and the resulting code requires $2n^2$ instructions to sort n numbers. Suppose further that just an average programmer writes for computer B, using a high-level language with an inefficient compiler, with the resulting code taking $50n \lg n$ instructions.

Computer A (Faster)

Running time grows like n^2 .

10 billion instructions per sec.

$2n^2$ instruction.

Computer B (Slower)

Grows in $n \lg n$.

10 million instruction per sec

$50n \lg n$ instruction.