**Unit 2: Algorithm**

**(2 hrs and contains 3 marks)**

1. **Concept and Definition:**

Informally, an algorithm is any well defined computational procedure that takes some value or set of values, as **input** and produces some value, or set of values; as **output.** An algorithm is is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a **tool** for solving a well defined-specified computational problem.

There are many definition of algorithms:

* An algorithm is a procedure, a finite set of well defined instructions, for solving a problem which, given an initial state, will terminate in a defined end state. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.
* An algorithm is said to be correct if, for every input instance, it halts with the correct output. We say that a **correct** algorithm solves the given computational problem.

1. **Design of Algorithm**

Algorithm design is a specific method to create a mathematical process in solving problems. Applied algorithm design is [algorithm engineering](http://en.wikipedia.org/wiki/Algorithm_engineering).

Algorithm design is identified and incorporated into many solution theories of [operation research](http://en.wikipedia.org/wiki/Operation_research), such as [dynamic programming](http://en.wikipedia.org/wiki/Dynamic_programming) and [divide-and-conquer](http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm). Techniques for designing and implementing algorithm designs are algorithm design patterns,[[1]](http://en.wikipedia.org/wiki/Algorithm_design#cite_note-1) such as [template method pattern](http://en.wikipedia.org/wiki/Template_method_pattern) and [decorator pattern](http://en.wikipedia.org/wiki/Decorator_pattern), and uses of data structures, and name and sort lists. Some current day uses of algorithm design can be found in internet retrieval processes of web crawling, packet routing and caching.

Mainframe programming languages such as [ALGOL](http://en.wikipedia.org/wiki/ALGOL) (for *Algo*rithmic *l*anguage), [FORTRAN](http://en.wikipedia.org/wiki/FORTRAN), [COBOL](http://en.wikipedia.org/wiki/COBOL), PL/I, [SAIL](http://en.wikipedia.org/wiki/SAIL_programming_language), and SNOBOL are computing tools to implement an "algorithm design"... but, an "algorithm design" (a/d) is not a language. An a/d can be a handwritten process, e.g. set of equations, a series of mechanical processes done by hand, an analog piece of equipment, or a digital process and/or processor.

One of the most important aspects of algorithm design is creating an algorithm that has an efficient run time, also known as its [big Oh](http://en.wikipedia.org/wiki/Big_Oh).

Steps in development of Algorithms

1. Problem definition
2. Development of a model
3. Specification of Algorithm
4. Designing an Algorithm
5. Checking the correctness of Algorithm
6. Analysis of Algorithm
7. Implementation of Algorithm
8. Program testing
9. Documentation Preparation

Basically there are three types of algorithm design technique.

* **Divide and conquer**

Divide and Conquer algorithms break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

There are three steps to applying Divide and Conquer algorithm in practice:

* **Divide** the problem into one or more subproblems.
* **Conquer** subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
* **Combine** the solutions to the subproblems into the solution for the original problem
* **Dynamic programming**

Dynamic programming is a design technique similar to divide and conquer. Divide-and-conquer algorithms partition the problem into independent subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. Dynamic programming is applicable when the subproblems are not independent, that is, when subproblems share subsubproblems. A dynamic-programming algorithm solves every subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the subsubproblem is encountered.

There are two ways of doing this.

**1.) Top-Down :** Start solving the given problem by breaking it down. If you see that the problem has been solved already, then just return the saved answer. If it has not been solved, solve it and save the answer. This is usually easy to think of and very intuitive. This is referred to as ***Memoization***.

**2.) Bottom-Up** **:** Analyze the problem and see the order in which the sub-problems are solved and start solving from the trivial subproblem, up towards the given problem. In this process, it is guaranteed that the subproblems are solved before solving the problem. This is referred to as Dynamic Programming.

Note that divide and conquer is slightly a different technique. In that, we divide the problem in to non-overlapping subproblems and solve them independently, like in mergesort and quick sort.

Dynamic programming is typically applied to optimization problems. In such problems there can be many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution *an optimal solution*, as opposed to *the optimal solution*, since there may be several solutions that achieve the optimal value.

Dynamic programming can be effectively applied to solve the longest common subsequence (LCS) problem. The problem is stated as following: given two sequences (or strings) x and y find a maximum-length common subsequence (substring) of x and y.

For example, given two sequences x = "ABCBDAB" and y = "BDCABA", the LCS(x, y) = { "BCAB", "BCAB", "BCAB" }. As you can see there are several optimal solutions.

* **Greedy Paradigm**

A greedy algorithm repeatedly executes a procedure which tries to maximize the return based on examining local conditions, with the hope that the outcome will lead to a desired outcome for the global problem. In some cases such a strategy is guaranteed to offer optimal solutions, and in some other cases it may provide a compromise that produces acceptable approximations.

1. **Characteristics of algorithm**

While designing an algorithm as a solution to a given problem, we must take care of the following five important characteristics of an algorithm.

**Finiteness:**

An algorithm must terminate after a finite number of steps and further each step must be executable in finite amount of time. In order to establish a sequence of steps as an algorithm, it should be established

that it terminates (in finite number of steps) on all allowed inputs.

**Definiteness (no ambiguity):**

Each steps of an algorithm must be precisely defined; the action to be carried out must be rigorously and unambiguously specified for each case.

**Inputs:**

An algorithm has zero or more but only finite, number of inputs.

**Output:**

An algorithm has one or more outputs. The requirement of at least one output is obviously essential, because, otherwise we cannot know the answer/ solution provided by the algorithm. The outputs have specific relation to the inputs, where the relation is defined by the algorithm.

**Effectiveness:**

An algorithm should be effective. This means that each of the operation to be performed in an algorithm must be sufficiently basic that it can, in principle, be done exactly and in a finite length of time, by person using pencil and paper. It may be noted that the ‘FINITENESS’ condition is a special case of ‘EFFECTIVENESS’. If a sequence of steps is not finite, then it cannot be effective also.

1. **“Big - Oh” Notation**

**Time Complexity And Space Complexity**

Time complexity is an analysis of the amount of time required to solve a problem of a particular size,while the Space complexity is an analysis of the amount of memory required.

**Analysis Of Algorithm**

The theoretical study of computer program performance and resource usage

**What more important than Performance?**

Correctness, stability, features, functionality, security, user friendliness, modularity, Simplicity, reliability

**Why study algorithm and performance?**

Scalability

Feasible vs Infeasible

performance is like money

JAVA and c

**Kinds of analysis**

**Worst Case (usually)**

T(n) = max time on any input of size n

**Average Case(sometimes)**

T(n)=expected time over all input of size n

**(Need assumption of statistical distribution)**

**Best Case Analysis:(bogus: not good)**

Cheat with a slow algorithm that works fast on some input

**What is insertion sorts W-c time?**

It depends on the speed of our computer.

* absolute speed (on diff machine)
* relative speed(on same machine)

**BIG IDEA: Asymptotic analysis**

1. Ignore machine dependent constant
2. Look at growth of T(n) as n->

**Asymptotic notation**

**ʘ notation:**

1. Drop low order terms
2. Ignore Leading constants

**EX** 3n³ + 90n² + 10n = ʘ(n³)

**As** n->**ʘ(n²) always beats ʘ(n³)**

**what do you mean by Asymptotic :**

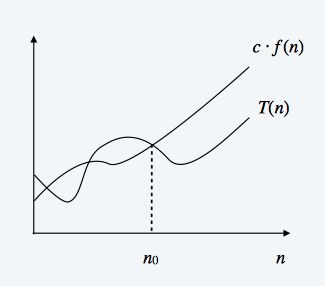
A line that continually approaches a given curve but does not meet at any finite distance

**Asymptotic Notation:**

Asymptotic Notation primarily deal with running time of algorithm.

***O* notation (Upper Bound):**

f(n)=*O*(g(n)) means there are constants C>0, no>0 such that 0 < = f(n) < c \* g(n) for all n = n0



**Ex. 2n2 = *O* (n3) means 2n2 *O* (n3)**

**Set definition**

For a given function g(n) , we denote by(g(n)) the set of function

*O*(g(n))={ f(n): there exist positive constants c>, no such that 0 < = f(n) <= c \* g(n) for all n=n0

}