**Chapter 1 : Role of Algorithms in Computing**

An **algorithm** is a well defined computational procedure that takes a set of values as input, performs some operations on that input and then produces a set of values as an output.

Kinds of problems solved by Algorithms:

1. Identifying all the 100000 genes in Human DNA, determining the sequences of the 3 billion chemical base pairs that make up the Human DNA. Developing tools to analyze this data.

2. Finding good routes for packets to travel in a network.

3. Searching for pages in the world wide web by matching the keywords

4. Public key cryptography and Digital signatures to ensure security of sensitive information being transferred across the network

5. Using linear programming to ensure that scarce resources are allocated in the most beneficial way/ Maximize profits by choosing the placement of investment etc

Examples of Algorithms:

1. Shortest path finding algorithms

2. Longest subsequence in strings

3. Topological sort to order items in terms of hierarchy

4. Finding the convex hull of n points on a graph

Not every problem solved by algorithms has an easily identified set of candidate solutions. Eg: Computing the Fourier Transform of few samples of a signal.

Data Structure is a way to store and organize data in order to facilitate access and modifications. No single DS works for all problems. We have to pick and choose whichever works the best for the problem in hand.

Hard Problems : These are the problems for which there is no efficient solution available.

A subset of these hard problems called NP-Complete problems have interesting properties. If an efficient solution exists for one instance of these problems, then there should be an efficient solution for all of them.

We use approximation algorithms to get the best possible output for these Hard problems.

Parallelism:

For many years, we could improve the performance of a processor by increasing its clock speed. But, this is not feasible now because chips run the risk of melting once their clock speeds become high enough. Hence, in order to perform more operations per second, chips are being designed to contain multiple cores.

In order to elicit the best performance from multi-core computers, we need to design algorithms with parallelism in mind.

Exercises:

1.1-1) Give a real-world example that requires sorting or a real-world example that requires computing a convex hull

A) Sorting - Displaying the names of students according to their rank.

Convex Hull - Finding outliers , Getting perimeter length etc

1.1-2) Other than speed, what other measures of efficiency might one use in a real-world setting?

A) Memory usage, Accuracy

1.1-3) Select a data structure that you have seen previously and discuss its strengths and limitations?

A) Tree

Strengths: Can arrange data in a hierarchical structure so that we can identify which nodes arrived before the others, can form a topology etc

Limitations: It is difficult to insert nodes into the right position in the tree

1.1-4) How are the shortest-path and travelling salesman problems given above similar? How are they different?

A) Similar : Both these problems try to find the shortest possible path in the graph

Different: In the shortest-path algorithm, we need not traverse all the points in the graph before reaching the destination node. Also, the destination node should be a node other than the origin node.

In the travelling-salesman problem, we need to traverse all the nodes in the graph in the shortest possible way before returning to the same node.

1.1-5) Come up with a real-world problem in which only the best-solution will do. Then come up with one in which a solution that is "approximately" the best is good enough.

A) Querying a database should always provide the result based on the clause in the query, and not an approximate result.

Hard problems like finding the Hamiltonian circuit in a graph or the graph coloring problem can have approximate solutions.

Algorithms as a technology:

We need algorithms even if the memory and computational resources are unlimited, so that we have a way of solving the problem with a successful program termination.

Efficiency:

Efficiency is always a concern in real-world scenarios.

Eg: Insertion sort takes C1n^2 with a small constant C1 , where as merge sort takes C2 n log n with a larger C2 value. But since log n function grows slowly, for a sufficiently large value of n, merge sort always outperforms insertion sort no matter how faster the system running insertion sort is.

Total system performance depends on choosing efficient algorithms as much as choosing fast hardware.

1.2-1) Give an example of an application that requires algorithmic content at the application level, and discuss the function of the algorithms invoked.

A) Google flights is an application that suggests the best/cost-effective flights available between the given source and destination for a range of dates. The suggestions can be filtered based on the layover time, number of flights to change, cost, time taken to reach the destination etc.

This is a graph problem, where the weights can be calculated based on the most important criteria, and then, the best path can be chosen between the source and destination using the shortest-path algorithm.

1.2-2) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in 8n^2 steps, while merge sort runs in 64n log n steps. For which values of n does insertion sort beat merge sort?

A) Insertion sort beats merge sort if 8n^2 <= 64 n log n

i.e., n <= 8 log n

implies for values n <= 42 , insertion sort will be faster than merge sort.

1.2-3) What is the smallest value of n such that an algorithm whose running time is 100n^2 runs faster than an algorithm whose running time is 2^n on the same machine.

A) We need to find the least n for which 100n^2 <= 2^n

The minimum value for it to be possible is n=15