Chapter-5
Introduction to Algorithm (2 hour)

Content

1.0 Algorithm

An algorithm can be defined as a finite numbers of step-by-step procedures for solving a problem. Multiple algorithms can be designed to solve a particular problem. However, the algorithms may differ in how efficiently they can solve the problem. In such situation, an algorithm that provides the maximum efficiency should be used for solving the problem. Efficiency means that the algorithm should work in minimal time and use minimal memory.
An algorithm has five important properties.

- **Finiteness:** an algorithm terminates after a finite number of steps.
- **Definiteness:** Each step in an algorithm is unambiguous. This means that the action specified by the step can't be interpreted in multiple ways and can be performed without any confusion.
- **Input:** An algorithm accepts zero or more inputs.
- **Output:** An algorithm produces at least one output.
- **Effectiveness:** An algorithm consists of basic instructions that are realizable. This means that the instruction can be performed by using the given input in a finite amount of time.

1.1 Deterministic and Non-Deterministic Algorithm

Deterministic algorithms solve the problem with exact decision at every step of the algorithms. Given an input, it will always produce same output and the underlying machine will always pass through the same sequence of states. Formally, a deterministic algorithm computes a mathematical function; a function has a unique value for any given input, and the algorithm is a process that produces this particular value as output.

Deterministic algorithms can be defined in terms of a state machine: a state describes what a machine is doing at a particular instant in time. State machines pass in a discrete manner from one state to another. Just after we enter the input, the machine is in its initial state or start state. If the machine is deterministic, this means that from this point onwards, its current state determines what its next state will be; its course through the set of states is predetermined. Note that a machine can be deterministic and still never stop or finish, and therefore fail to deliver a result.

Non-Deterministic algorithm is an algorithm that can exhibit different behaviors on different run. It is the algorithm with one or more choice points where multiple different routes are possible, without any specification of which one will be taken. There are several ways an algorithm may behave different from run to run. A concurrent algorithm can perform differently on different runs due to a race condition. A probabilistic algorithm's behavior depends on random number generator.

Non deterministic algorithms are used when the problem solved by the algorithm inherently allows multiple outcomes or when there is a single outcome with multiple paths by which the outcome may be discovered.

1.2 Divide and Conquer Algorithm

The **divide and conquer** algorithm approach is an algorithm design technique that involves breaking down the problem recursively into sub problems until the sub problems become so small that they can be directly solved. The solutions to the problems are then combined to give a solution to the original problem.
Divide and conquer is a powerful approach for solving conceptually difficult problems. Divide and conquer paradigm consists of following major phases.

1. Breaking the problems into sub problems.
2. Solving the trivial cases.
3. Combining the solutions of the sub problems to solve the original problem

Divide and conquer often provides a natural way to design efficient algorithm. One of the example of algorithm based on divide and conquer approach is **merge sorting.**
A simple variant of divide and conquer is called decrease and conquer algorithm, that solve an identical sub problem and use the solution to solve the bigger problem. An example of decrease and conquer algorithms is **binary search algorithm.**

Example of Divide and Conquer Algorithm:

Consider we have to find the minimum value from the list of numbers as given below

7	3	4	2	1	9	8	10

To find the minimum value, we can divide the list into two halves as shown below

7	3	4	2

1	9	8	10

Again dividing each of two list into two halves as shown below

7	3

4	2

1	9

8	10

Now, there are only two elements in each list. Comparing the two elements and finding minimum of the two we get four elements as shown below.

3

2

1

8

Again, comparing first two and last two minimum values to determine their minimum. The two minimum values thus obtained are shown below.

2

1

Finally, comparing the two minimum values to obtain the overall minimum value which is a 1 as shown below.

1

1.3 Series and Parallel Algorithm

In computer science, a **sequential algorithm or serial algorithm** is an algorithm that is executed sequentially i.e. instructions of an algorithm execute one at a time. Most standard computer algorithms are sequential algorithms. Serial algorithms are used to solve serial problem. Serial problems are those which can't be divided into pieces because they require the result from preceding step to operate on next step. Examples include iterative numerical methods such as NR method to find the roots of quadratic equations.

In computer science, **a parallel algorithm,** as opposed to a traditional serial algorithm, is an algorithm which can be executed a piece at a time on many different processing devices, and then combined together again at the end to get the correct result. Parallel algorithms are used to solve parallel problem. Parallel problems are those problems which can be easily divided into pieces. Example includes a algorithm to find all the prime numbers ranging from 1 to 500. In this problem, we can assign a subset of the numbers to each available processor, and putting the list of positive result back together.

Parallel algorithms are valuable because it is faster to perform large computing task via parallel algorithm than via serial (non-parallel) algorithm, because it is far more difficult to construct a computer with a single fast processor than one with many slow processors with the same throughput.

1.4 Approximate algorithms and Heuristics Algorithms

Approximation algorithms are algorithms used to find approximate solutions to optimization problems i.e. instead of searching for an optimal solution, search for an "almost" optimal solution. The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time. Approximation algorithms are generally associated with NP (Non-Deterministic polynomial) hard problem which is a class of decision problem.

A **heuristic algorithm** is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. The objective of a heuristic is to produce a solution in a reasonable time. This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because of its ability to find solution in a reasonable time. In heuristics algorithms, information about the problem such as nature of the states, cost of transferring from one state to another, the consequence of taking certain path etc can be used to find out an optimum solution. This information is called heuristic information.

Example: An example of decision problem whose solution can be obtained based on approximation is Travelling Salesman problem (TSP). One of the types of heuristic algorithm is **Greedy approach (** *The greedy approach is an algorithm design technique that select the best possible option any given time so as to minimize costs under a given set of conditions.***)** which can be used to give good solution i.e. approximation to optimal solution.

1.5 Big O Notation

The greatest difficulty in solving programming problem is now how to solve the problem but how to solve the problem efficiently. There are numbers of algorithm to solve a particular problem, so it is difficult to decide which algorithm to use. The efficiency of algorithm can be computed by determining the amount of resources it consumes. The resources include.

1. Time: The CPU time required to execute algorithm
2. Space: The amount of memory used by the algorithm for execution.

Since memory is extensible but not time so the efficiency of algorithm is determined on how efficiently it uses time. The running time of a program is a function of n, where n is the size of input data. The rate at which the running time of an algorithm increases as a result of an increase in the volume of input data is called the **order of growth** of the algorithm. The order of growth of an algorithm is defined by using big O notation. The big O notation has been accepted as a fundamental technique for describing the efficiency of an algorithm.

The following table lists some possible order of growths and their corresponding big O notation.

Order of growth	Big O Notation
Constant	O(1)
Logarithmic	O(Log n)
Linear	O(n)
Loglinear	O(n Logn)
Quadratic	$O(n^2)$
Cubic	$O(n^3)$

if an algorithm has linear order of growth, the algorithm is said to be of the order O(n). Similarly, if an algorithm has a quadratic order of growth, the algorithm is said to be order O (n^2). According to their order of growth, the big O notation can be arranged in an increasing order ass

$$O(1) < O(\text{Log } n) < O(n) < O(n \text{ Log} n) < O(n^2) < O(n^3)$$

So , if a problem can be solved by using algorithms having preceding orders of growth, then an algorithm of order O(1) is considered as best, and an algorithm of the order $O(n^3)$ will be considered worst.

Example:

Consider the following tow algorithms to find the sum of the first n natural numbers.

Algorithm1

1. Set sum = 0 // 1 assignments
2. Set I=0 // 1 assignments
3. While (I < =n) // n+2comparisons
   a. Set sum = sum + i // n+1 arithmetic operations and n+1 assignments
   b. i ++ // n+1 increments
4. Display sum // 1 write

Algorithm 2

1. Set sum = (n * (n + 1)) / 2 // 3 arithmetic operation and 1 assignments
2. Display sum // 1 Write

Both the above algorithm perform same task, now to determine which one is efficient we have to find the order of growth.

Now, let us consider that assignments, comparisons, write and increment statement take a, b, c, d time units to execute respectively. Suppose arithmetic operations require e time to execute.

Then the execution time required for algorithm1 is given by

T1= (1 * a) + (1 * a) + ((n+2) * b) + ((n+1) * e) +((n+1) * a) + ((n+1) *d) + (1 * c)
= a + a + nb +2b + ne + e + na +a + nd +d + c
= n(b + e + a + d) +(3a + c+d+e)

As T1 is linear function of n i.e. execution time increases linearly with value of user inputs, therefore the algorithm is of order O (n).

Again, the execution time required for algorithm2 is given by

T2 = (3 * e) + (3 * a) + (1 * c)
= 3e + 3a + c

As T2 is a constant, algorithm2 doesn't depend on the value of user input. Therefore the algorithm is of the order O(1). Since an algorithm with big O notation O (1) is more efficient than algorithm with big O notation O (n), hence algorithm1 is more efficient.

Assignment: Writes short notes on Space-Time Tradeoff.