Unit-1 Introduction to Data Structures & Algorithm

1.1. Data type and Abstract data types.
1.2. Data structures: linear and non-linear, static and dynamic.
1.3. Algorithms.
    1.3.1. Greedy algorithm.
    1.3.2. Divide and Conquer.
    1.3.3. Backtracking.
    1.3.4. Randomized algorithms.
1.4. Analysis of Algorithms.
    1.4.1. Big O notation and its rule.

1.1 Data type and Abstract data types

- Data type

A data type is defined as a set of values that a variable can store along with the set of operations that can be performed on that variable.

In many programming languages, each and every variable and constant that we are using in our program must be declared before it's use.

There are two categories of data types:

a) Fundamental or basic data types (int, char, float, void) &

b) Derived data types (array, string, structure, union)

- Abstract data types:

An abstract data type is a specification of a set of data and the set of operations that can be performed on the data. It is a useful tool for specifying the logical properties of a data type. The term ADT refers to the basic mathematical concept that defines data type.

When we use abstract data types, our program is divided into two pieces.

1. Application: The part that uses the abstract data type.
2. Implementation:

The part that implements the abstract data type.

These two pieces are completely independent. It should be possible to take the implementation developed for one application and use it for a completely different application with no changes.

1.2 Data structures: linear and non-linear, static and dynamic.

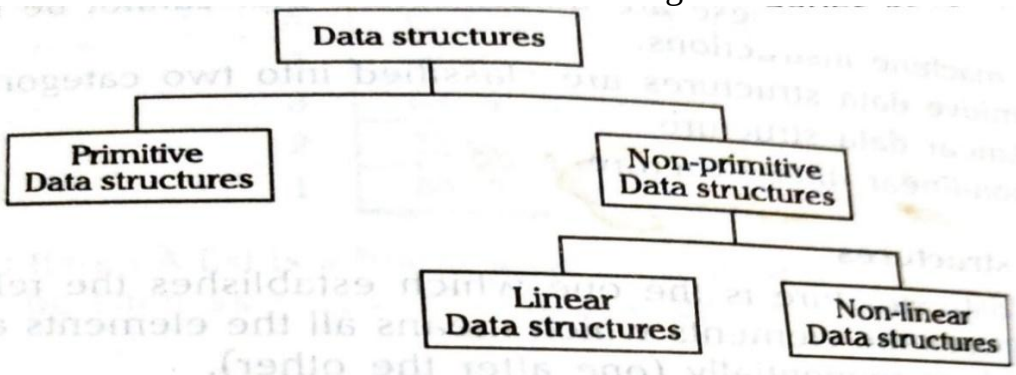
- Data structure

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. The word data means content and structure refers to the place where data of content is stored.

A data structure is the way of organizing all data items and establishing relationships among those data items. Data structures are the building blocks of a program.

- Classification of Data Structure

Data Structure can be classified into two categories as shown below.



1. Primitive Data Structure.

The data structure which can be directly operated by machine level instructions is called primitive data structure. It is a kind of data structure that stores the data of only one type.

Examples: char, int, float, double etc.

2. Non-primitive Data Structure.

The data structure which cannot be directly operated by machine level instructions is called non-primitive data structure. It is a kind of data structure that stores the data of more than one type.

There are two types of non-primitive data structure.

a) Linear data structure.

In linear data structure, the data elements are stored in a consecutive memory location or data are stored in a sequential manner.

Examples: array, stack, queue etc.

b) Non-linear data structure

In non-linear data structure, the data elements are not stored in a consecutive memory location or stored in non-sequential manner. These data structures are used to represent the hierarchical relationship between data elements.

Examples: tree, graph etc.

According to the nature of size, data structure can be classified into following types.

a) Static data structure

The size of data items (data elements) is fixed in static data structure.

Examples: array.

b) Dynamic data structure

The size of data items (data elements) is not fixed in dynamic data structure. It can be changed during the execution of the program in dynamic data structure.

Examples: linked list, tree, graph etc.

1.3 Algorithms.

An algorithm is composed of a finite set of steps each of which may require one or more operations. Each operation may be characterized as either a simple or complex. Operations performed on scalar quantities are termed as simple, while operations performed on vector data are normally termed as complex.

- Properties of algorithm:

1. The number of instructions should be finite.
2. The problem should be divided into small modules.
3. The steps written in the algorithm should be simple and unambiguous.
4. It should have an input, process and desire output after executing the algorithm.
5. It should not depend on any particular language or computer.

- **Algorithm Design Approach**

1.3.1 Greedy algorithm

A greedy algorithm works by taking a decision that appears best at the moment, without thinking about the future. The decision once taken is never reconsidered. This means that a local optimum is chosen at every step in hope of getting a global optimum at the end. It is not necessary that a greedy algorithm will always produce an optimal solution. Some examples where greedy approach produces optimal solutions are Dijkstra's algorithm for single source shortest paths, Prim's and Kruskal's algorithm for minimum spanning tree and Huffman algorithm.

1.3.2 Divide and Conquer algorithm

A divide and conquer algorithm solves a problem by dividing it into smaller and similar sub problems. The solutions of these smaller problems are then combined to get the solution of the given problem. The examples of divide and conquer algorithms are merge sort, quick sort, binary search.

1.3.3 Backtracking algorithm

In some problems we have several options, where any one might lead to the solution. We will take an option and try and if we do not reach the solution, we will undo our action and select another one. The steps are retraced if we do not reach the solution. It is a trial and error process. An example of backtracking is the Eight Queens problem.

1.3.4 Randomized algorithms

In a randomized algorithm, random numbers are used to make some decisions. The running time of such algorithms depends on the input as well as the random numbers that are generated. The running time may vary for the same input data. An example of randomized algorithm is a version of quick sort where a random number is chosen as the pivot.

1.4 Analysis of Algorithms

- Asymptotic Analysis

Asymptotic notation is the simplest and easiest way of describing the running time of an algorithm. It represents the efficiency and performance of an algorithm in a systematic and meaningful manner. It describes the time complexity in terms of three common measures, best case (fastest possible), worst case (slowest possible) and average case (average possible time).

Asymptotic analysis is the theoretical analysis of an algorithm. The following notations are used for asymptotic analysis.

1. Big-Oh Notation (O)

This notation (O) gives an upper bound and a function *f(n).* The upper bound of *f(n)* indicates that the function *f(n)* will be the worst case that it does not consume more than this computing time. In other words, we can compute the maximum possible amount of time that an algorithm will take for its completion.

Definition:

Consider *f(n)* and *g(n)* be two positive functions of *n*, where '*n*' is the size of the input data. Then *f(n)* is Big-Oh of *g(n)*, if and only if there exists a positive constant C and an integer n_o, such that $f(n) \leq C.g(n)$ and $n > n_0$.

Here, *f(n)=O(g(n))*

Based on Big-Oh Notation (O), the algorithms can be categorized as follows:

- Constant time *(O(1))* algorithms.
- Linear time *(O(n))* algorithms.
- Logarithmic time *(O(log n))* algorithms.
- Polynomial time $(O(n^k)$, for $k>1)$ algorithms.
- Exponential time $(O(k^{n)}$, for $k>1)$ algorithms.