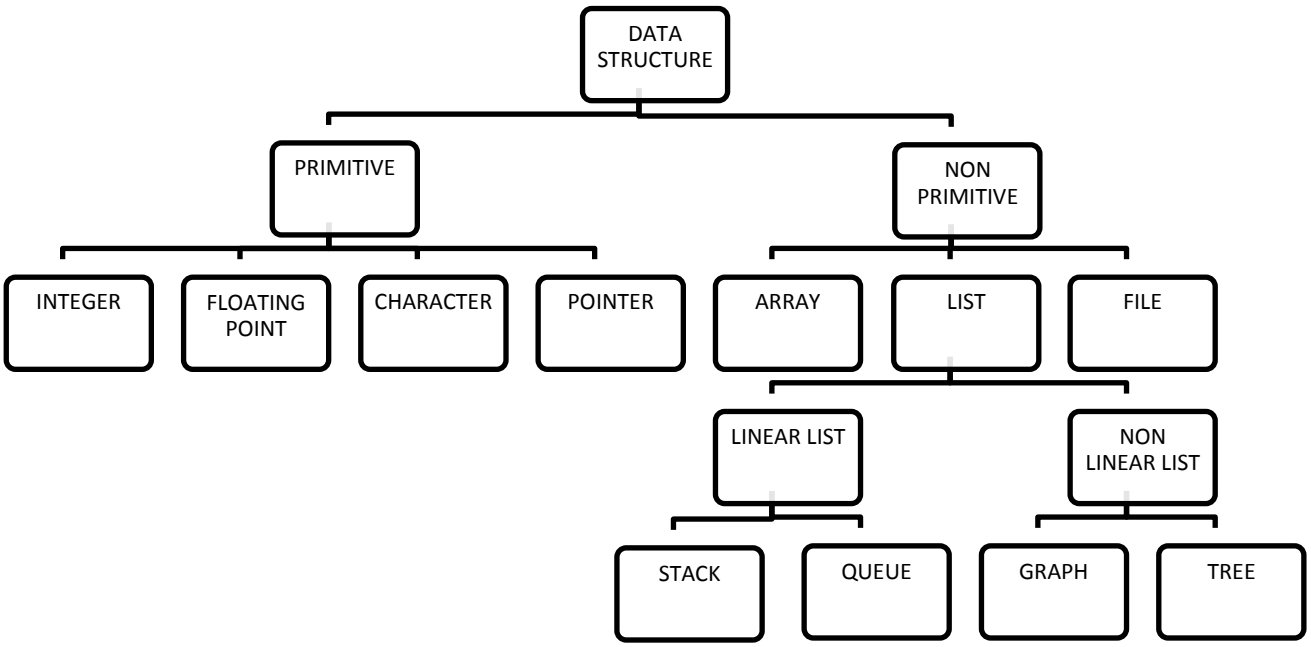
Introduction to Data Structure

- Computer is an electronic machine which is used for data processing and manipulation.
- When programmer collects such type of data for processing, he would require to store all of them in computer's main memory.
- In order to make computer work we need to know
  - Representation of data in computer.
  - Accessing of data.
  - How to solve problem step by step.
- For doing this task we use data structure.

What is Data Structure?

- **Data structure** is a representation of the logical relationship existing between individual elements of data.
- Data Structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
- We can also define data structure as a mathematical or logical model of a particular organization of data items.
- The representation of particular data structure in the main memory of a computer is called as **storage structure.**
- The storage structure representation in auxiliary memory is called as **file structure**.
- It is defined as the way of storing and manipulating data in organized form so that it can be used efficiently.
- Data Structure mainly specifies the following four things
  - Organization of Data
  - Accessing methods
  - Degree of associativity
  - Processing alternatives for information
- Algorithm + Data Structure = Program
- Data structure study covers the following points
  - Amount of memory require to store.
  - Amount of time require to process.
  - Representation of data in memory.
  - Operations performed on that data.

Classification of Data Structure



Data Structures are normally classified into two broad categories

1. Primitive Data Structure
2. Non-primitive data Structure

Data types

A particular kind of data item, as defined by the values it can take, the programming language used, or the operations that can be performed on it.

Primitive Data Structure

- Primitive data structures are basic structures and are directly operated upon by machine instructions.
- Primitive data structures have different representations on different computers.
- Integers, floats, character and pointers are examples of primitive data structures.
- These data types are available in most programming languages as built in type.
  - Integer: It is a data type which allows all values without fraction part. We can use it for whole numbers.
  - Float: It is a data type which use for storing fractional numbers.
  - Character: It is a data type which is used for character values.

Pointer: A variable that holds memory address of another variable are called pointer.

Non primitive Data Type

- These are more sophisticated data structures.
- These are derived from primitive data structures.
- The non-primitive data structures emphasize on structuring of a group of homogeneous or heterogeneous data items.
- Examples of Non-primitive data type are Array, List, and File etc.
- A Non-primitive data type is further divided into Linear and Non-Linear data structure
  - **Array:** An array is a fixed-size sequenced collection of elements of the same data type.
  - **List:** An ordered set containing variable number of elements is called as Lists.
  - **File:** A file is a collection of logically related information. It can be viewed as a large list of records consisting of various fields.

Linear data structures

- A data structure is said to be Linear, if its elements are connected in linear fashion by means of logically or in sequence memory locations.
- There are two ways to represent a linear data structure in memory,
  - Static memory allocation
  - Dynamic memory allocation
- The possible operations on the linear data structure are: Traversal, Insertion, Deletion, Searching, Sorting and Merging.
- Examples of Linear Data Structure are Stack and Queue.
- Stack: Stack is a data structure in which insertion and deletion operations are performed at one end only.
  - The insertion operation is referred to as 'PUSH' and deletion operation is referred to as 'POP' operation.
  - Stack is also called as Last in First out (LIFO) data structure.
- Queue: The data structure which permits the insertion at one end and Deletion at another end, known as Queue.
  - End at which deletion is occurs is known as FRONT end and another end at which insertion occurs is known as REAR end.
  - Queue is also called as First in First out (FIFO) data structure.

Nonlinear data structures

- Nonlinear data structures are those data structure in which data items are not arranged in a sequence.
- Examples of Non-linear Data Structure are Tree and Graph.
- **Tree:** A tree can be defined as finite set of data items (nodes) in which data items are arranged in branches and sub branches according to requirement.
  - Trees represent the hierarchical relationship between various elements.
  - Tree consist of nodes connected by edge, the node represented by circle and edge lives connecting to circle.

- **Graph:** Graph is a collection of nodes (Information) and connecting edges (Logical relation) between nodes.
  - A tree can be viewed as restricted graph.
  - Graphs have many types:
    - Un-directed Graph
    - Directed Graph
    - Mixed Graph
    - Multi Graph
    - Simple Graph
    - Null Graph
    - Weighted Graph

Difference between Linear and Non Linear Data Structure

Linear Data Structure	Non-Linear Data Structure
Every item is related to its previous and next time.	Every item is attached with many other items.
Data is arranged in linear sequence.	Data is not arranged in sequence.
Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
Implementation is easy.	Implementation is difficult.

Operation on Data Structures

Design of efficient data structure must take operations to be performed on the data structures into account. The most commonly used operations on data structure are broadly categorized into following types

1. **Create**
   The create operation results in reserving memory for program elements. This can be done by declaration statement. Creation of data structure may take place either during compile-time or run-time. malloc() function of C language is used for creation.

2. **Destroy**
   Destroy operation destroys memory space allocated for specified data structure. free() function of C language is used to destroy data structure.

3. **Selection**
   Selection operation deals with accessing a particular data within a data structure.

4. **Updation**
   It updates or modifies the data in the data structure.

5. **Searching**
   It finds the presence of desired data item in the list of data items, it may also find the locations of all elements that satisfy certain conditions.

6. **Sorting**
   Sorting is a process of arranging all data items in a data structure in a particular order, say for example, either in ascending order or in descending order.

7. **Merging**
   Merging is a process of combining the data items of two different sorted list into a single sorted list.

8. **Splitting**
   Splitting is a process of partitioning single list to multiple list.

9. **Traversal**
   Traversal is a process of visiting each and every node of a list in systematic manner.

Time and space analysis of algorithms

Algorithm

- An essential aspect to data structures is algorithms.
- Data structures are implemented using algorithms.
- An algorithm is a procedure that you can write as a C function or program, or any other language.
- An algorithm states explicitly how the data will be manipulated.

Algorithm Efficiency

- Some algorithms are more efficient than others. We would prefer to choose an efficient algorithm, so it would be nice to have metrics for comparing algorithm efficiency.
- The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process.
- Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm
- **Time complexity**
  - **Time Complexity** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm.

- "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.

- **Space complexity**
  - **Space complexity** is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.
  - We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this.
  - We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit.
  - Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

Worst Case Analysis

In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched is not present in the array. When x is not present, the search () functions compares it with all the elements of array [] one by one. Therefore, the worst case time complexity of linear search would be.

Average Case Analysis

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed. So we sum all the cases and divide the sum by (n+1).

Best Case Analysis

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in worst case is constant (not dependent on n). So time complexity in the best case would be.
