UNIT – I

DATA STRUCTURES

DEFINITION

A *data structure* is a group of data elements that are put together under one name and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

or

It is the way of organizing, storing, retrieving data and maintain their relationship with each other.

or

It is the logical and mathematical model to organize and store data in computer memory so that we can use it efficiently.

Characteristics of Data Structures

- ✓ It represents the logical representation of data in computer memory.
- ✓ It represents the logical relationship between the various data elements.
- ✓ It helps in efficient manipulation of stored data elements.
- ✓ It allows the programs to process the data in an efficient manner.

Applications of Data Structures

Data structures are widely applied in the following areas:

- ✓ Compiler design
- ✓ Operating system
- ✓ Statistical analysis package
- ✓ DBMS
- ✓ Numerical analysis
- ✓ Simulation
- ✓ Artificial intelligence
- ✓ Graphics

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CLASSIFICATION OF DATA STRUCTURES

The data structures have been classified into two types:

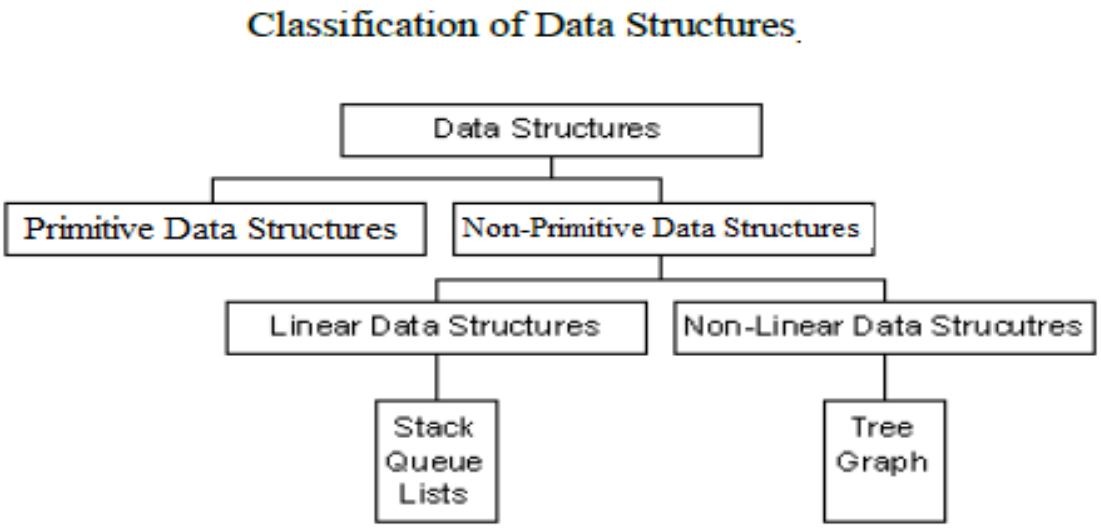
- ✓ **Primitive Data Structures**
- ✓ **Non – Primitive Data Structures**

Primitive Data Structures

- ✓ It is also known as primary data structures.
- ✓ Primitive data structures are the fundamental data types which are supported by a programming language.
- ✓ They can be directly manipulated by machine instructions.
- ✓ For example integer, real, character, and boolean.

Non – Primitive Data Structures

- ✓ It is also known as secondary data structures.
- ✓ They are created by using primitive data structures.
- ✓ Its main objective is to form a set of homogeneous or heterogeneous data elements.
- ✓ For example linked lists, stacks, trees, and graphs.
- ✓ They are classified into two types:
  - ✓ **Linear data structures**
  - ✓ **Non – Linear data structures**



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Linear Data Structure

- ✓ A data structure is said to linear data structure if all the data elements are arranged in some linear or sequential order.
- ✓ Examples for linear data structure are:
  - ✓ **Arrays**
  - ✓ **Stacks**
  - ✓ **Queues**
  - ✓ **Linked Lists, etc.**
- ✓ They can represented in memory in two ways:
  - ✓ **Using sequential memory locations**
  - ✓ **Using links**

Non – Linear Data Structure

- ✓ A data structure is said to be non – linear data structure if all the data elements are arranged not in sequential order.
- ✓ The relationship of side by side storing is not maintained.
- ✓ The hierarchical relationship among the data elements is seen in this type of data structures.
- ✓ Examples for non-linear data structure are:
  - ✓ **Trees**
  - ✓ **Graphs**

Static DS Versus Dynamic DS

- ✓ If a data structure is created using static memory allocation then it is known as static data structure.
- ✓ If a data structure is created using dynamic memory allocation then it is known as dynamic data structure.

OPERATIONS ON DATA STRUCTURES

- ✓ The basic operations that can be performed on a data structure are:
  - ✓ **Insertion**
  - ✓ **Deletion**

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

- ✓ **Search**
- ✓ **Traverse**
- ✓ **Sorting**
- ✓ **Merging**

- ✓ ***Insertion:*** - It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.
- ✓ ***Deletion:*** - It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.
- ✓ ***Searching:*** - It is used to find the location of one or more data items that satisfy the given constraint. Such a data item may or may not be present in the given collection of data items. For example, to find the names of all the students who secured 100 marks in mathematics.
- ✓ ***Traversing:*** - It means to access each data item exactly once so that it can be processed. For example, to print the names of all the students in a class.
- ✓ ***Sorting:*** - Data items can be arranged in some order like ascending order or descending order. For example, arranging the names of students in a class in an alphabetical order.
- ✓ ***Merging:*** - Lists of two sorted data items can be combined to form a single list of sorted data items.

ABSTRACT DATA TYPE (ADT)

It is the logical description of how we can view the data and operations that are allowed without implementation.

or

It is defined as mathematical model representing data and operations only but no implementation.

or

It is defined as a collections various data items and its operations while hiding implementation.

It is defined as collection of instances and operations rather than implementation.

or

It is defined as data together with functions that operate on that data

Benefits / Advantages of ADT

- ✓ Modularity
- ✓ Reuse
- ✓ Code is easier to understand
- ✓ Implementation of ADT can be changed without requiring changes to the program that uses ADT.

PRELIMINARIES OF ALGORITHMS

An algorithm is defined as a finite sequence of instructions each of which has a clear meaning and can be performed with a finite amount of effort in a finite amount of time.

- ✓ An algorithm is a method of finding solution in solving a problem.
- ✓ **An algorithm is step by step procedure to solve a problem.**
- ✓ Algorithms are used to find right solution to variety classification problems.
- ✓ The algorithm word originated from the **Arabic word "Algorism"** which is linked to the name of the Arabic mathematician **AI Khwarizmi**.
- ✓ He is considered to be the first algorithm designer for adding numbers.
- ✓ In algorithm all steps must unambiguous.

Structure of Algorithm

The structure contains the following steps:

- ✓ Input Step
- ✓ Assignment Step

- ✓ Decision Step
- ✓ Repetitive Step
- ✓ Output Step

Algorithm for adding two numbers

Step1: Start

Step2: Read two numbers a, b

Step3: Compute result=a + b

Step4: Display the result

Step5: Stop

Algorithm for to check even or odd

Step1: Start

Step2: Read the value n

Step3: Compute if n modulo division by 2 is equal to 0 then goto step 4 & then to step step 6 otherwise goto step 5

Step4: Display even number

Step5: Display odd number

Step6: Stop

Algorithm for to sum of n numbers

Step1: Start

Step2: Read the value n

Step3: Assign the value 1 to i and 0 to sum

Step4: Compute while i less than n then perform step5 and step6 then goto Step4 otherwise goto Step7

Step5: compute sum=sum plus i

Step6: increment i

Step7: Display sum value

Step8: Stop

Properties of Algorithms

- ✓ **Finiteness:** An algorithm must terminate after a finite number of steps.
- ✓ **Definiteness:** The steps of the algorithm must be precisely defined or unambiguously specified.
- ✓ **Generality:** An algorithm must be generic enough to solve all problems of a particular class.
- ✓ **Effectiveness:** The operations of the algorithm must be effective such that it must easily converted into machine code in a finite amount of time.
- ✓ **Input-Output:** The algorithm must have zero, one or more inputs and one or more outputs.

TIME AND SPACE COMPLEXITIES

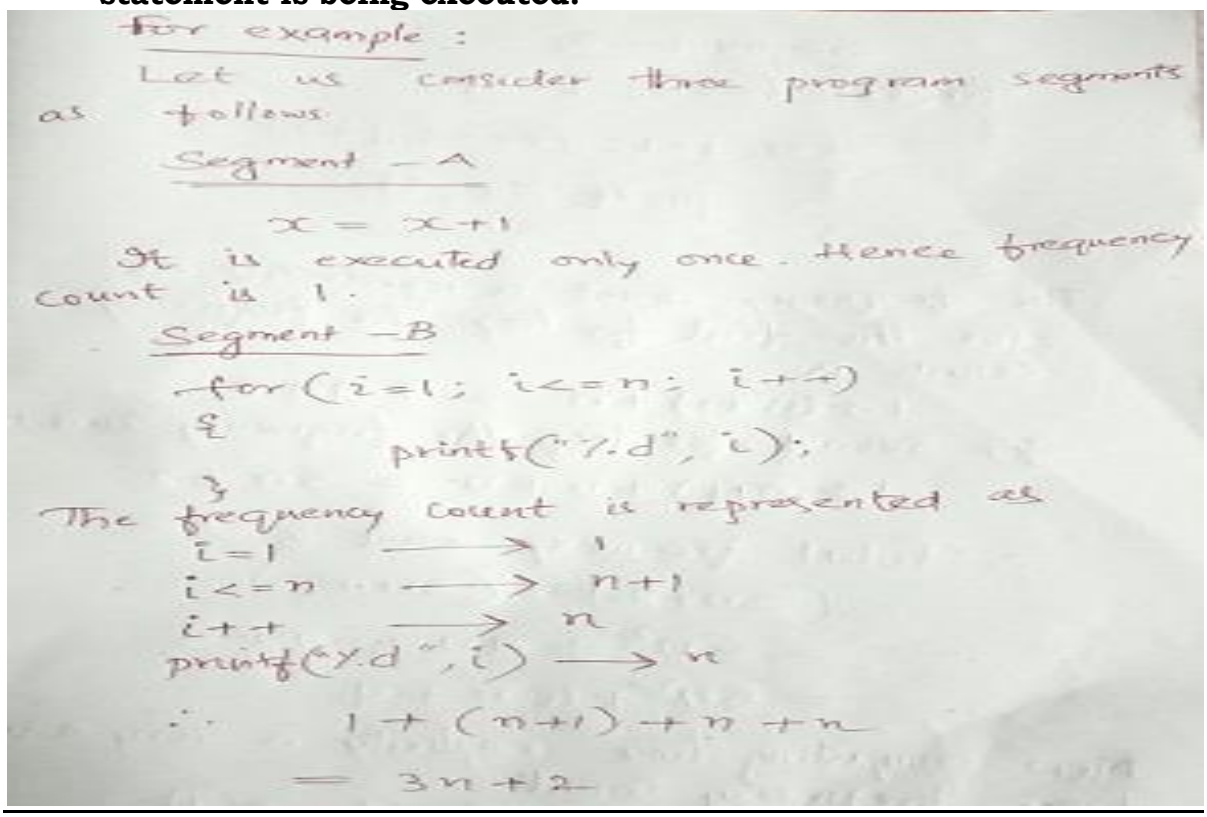
- ✓ The performance of algorithm can be measured in terms of:
  - ✓ **Time**
  - ✓ **Space**
- ✓ The performance is the amount of memory needed and time required to run it.
- ✓ We have two methods to determine the performance of the program.
  - ✓ **Analytical – in analysis**
  - ✓ **Experimental – in measurement**
- ✓ **Time Complexity:** The time complexity of an algorithm or a program is the running time of the program as a function of input size.
- ✓ **Space Complexity:** The space complexity of an algorithm or program is the amount of computer memory required for program execution as a function of input size.

Analysis of Algorithms

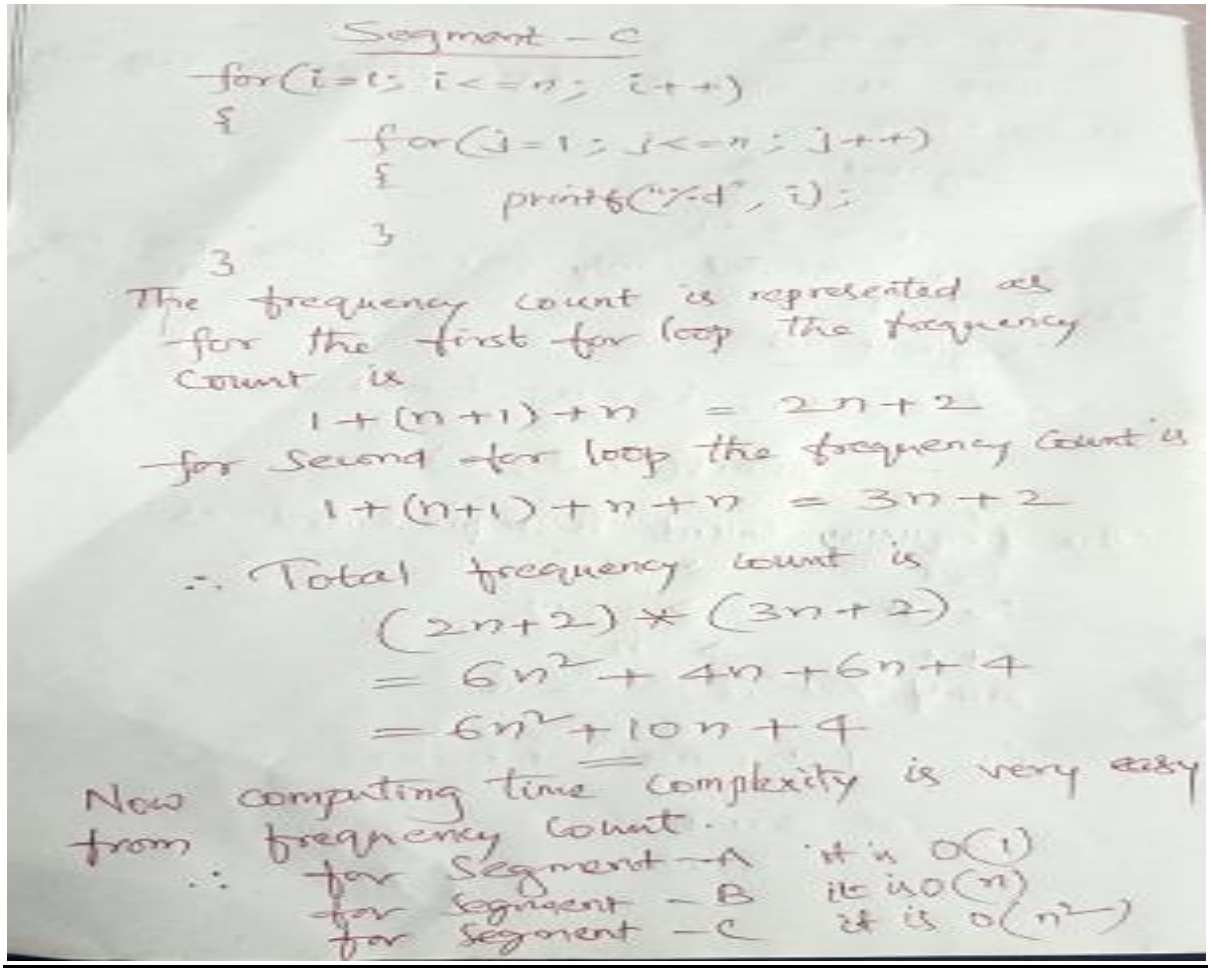
- ✓ It is a technique to compare efficiency of different algorithms
- ✓ The speed of an algorithm can be different on different computers(time taken will be different)

Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

- ✓ For solving a problem, the time is expressed in terms of **mathematical function** of input size.
- ✓ Two algorithms are compared based on **rate of growth** of that function
- ✓ If **rate of growth** is higher then the algorithm takes more time as input size increases.
- ✓ The mathematical functions are represented by using **asymptotic notations**
- ✓ It depends on how the program works efficiently.
- ✓ Efficiency means **less space** and **less time** required for execution**.**
- ✓ Hence **time** and **space** are the factors that determine the efficiency of the program.
- ✓ We cannot compute **time** in terms of seconds for the execution of the program because of several factors.
- ✓ Therefore we consider **frequency count** as time taken for execution of the program.
- ✓ **The frequency count is defined as the total number of times each statement is being executed.**



Dr. Ratna Raju Mukiri M.Tech(CSE)., S.E.T., Ph.D.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Segment – C

```
for(i=1; i<=n; i++)
{
    for(j=1; j<=n; j++)
    {
        printf("%d", i);
    }
}
```

The frequency count is represented as
for the first for loop the frequency count is

$1+(n+1)+n = 2n+2$

for second for loop the frequency count is

$1+(n+1)+n+n = 3n+2$

∴ Total frequency count is

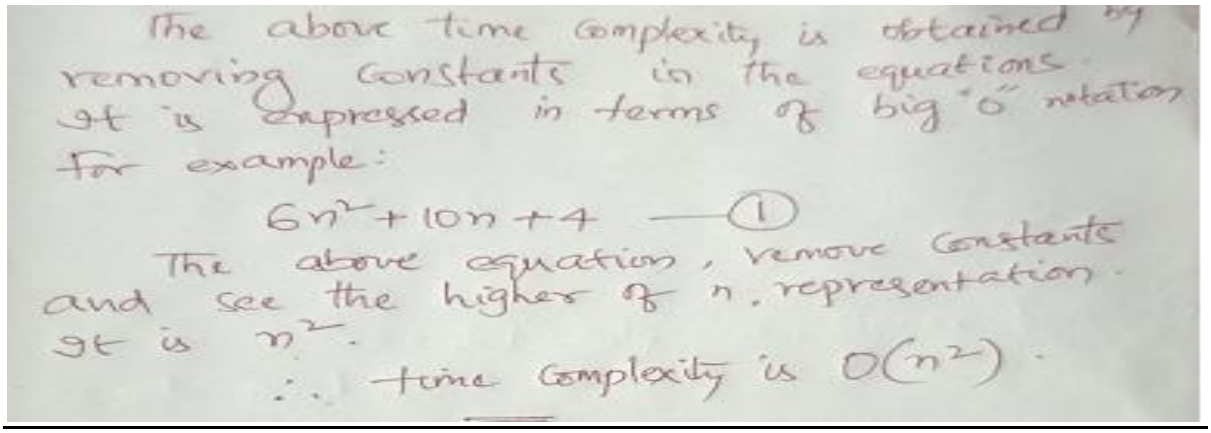
$(2n+2)*(3n+2)$

$= 6n^2+4n+6n+4$

$= 6n^2+10n+4$

Now computing time complexity is very easy from frequency count.

∴ for Segment – A it is $O(1)$
for Segment – B it is $O(n)$
for Segment – C it is $O(n^2)$



The above time complexity is obtained by removing constants in the equations. It is expressed in terms of big "O" notation for example:

$6n^2+10n+4$ — (1)

The above equation, remove constants and see the higher of n, representation. It is n^2.

∴ time complexity is $O(n^2)$.

Analyzing Algorithms

- ✓ Suppose **"M"** is an algorithm, and suppose **"n"** is the size of the input data. Clearly the complexity **f(n) of M** increases as **n** increases.
- ✓ It is usually the rate of increase of **f(n)** with some standard functions.
- ✓ The most common computing times are

$O(1)$, $O(\log_2 n)$, $O(n)$, $O(n \log_2 n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$

- ✓ When we have two algorithms to perform the same task and if first one time complexity is **O(n)** and second one is **O(n^2)** then we prefer first one since as n increases the time required for execution of second one is taking more time than the first one.
- ✓ Here we discuss about three cases for the efficiency of the algorithm.
  - ✓ **Best case – 1**
  - ✓ **Worst case – n**
  - ✓ **Average case – (n+1)/2**
- ✓ It the algorithm takes minimum amount of time to run for its completion then it is called **best case** time complexity
- ✓ It the algorithm takes maximum amount of time to run for its completion then it is called **worst case** time complexity
- ✓ It the algorithm takes average amount of time to run for its completion then it is called **average case** time complexity