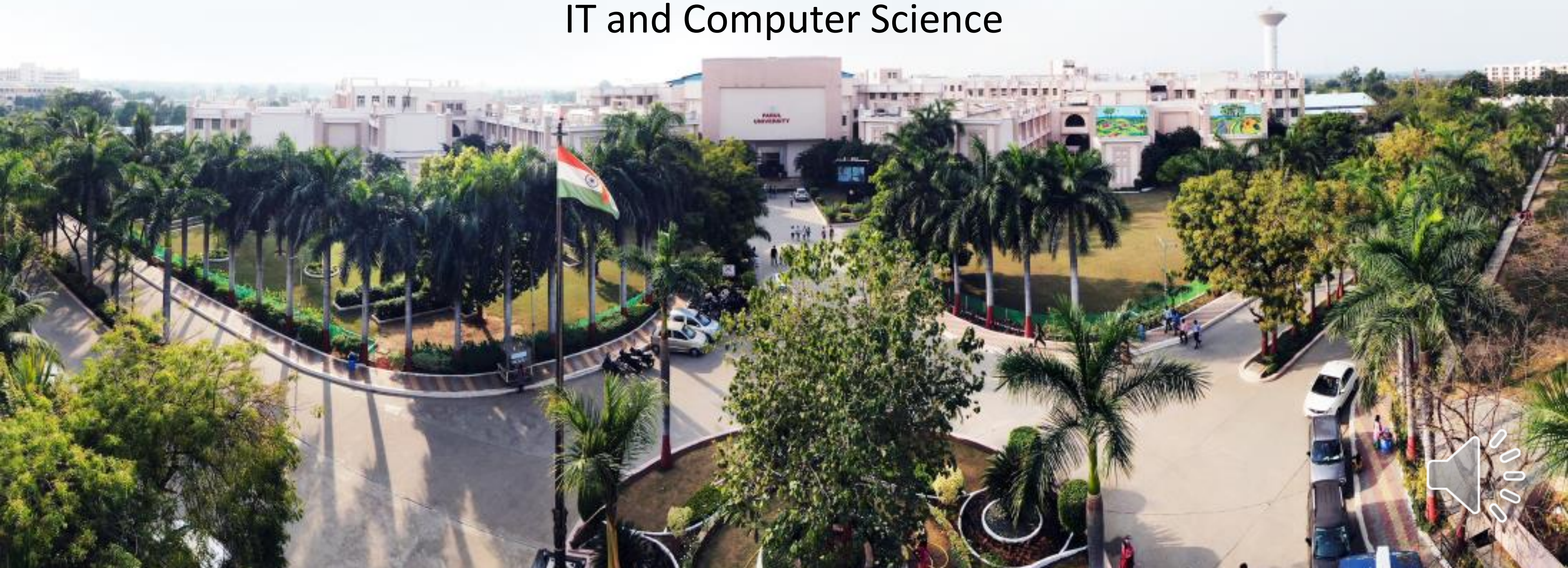
Data Structure

Prof. Vijya Tulsani,
Prof.Dharmendrasinh Rathod
Assistant Professor
IT and Computer Science



Parul® University

DIGITAL LEARNING CONTENT

CHAPTER-1

Introduction of Data Structure

Terms

Data: facts and statistics collected together for reference or analysis.

Collection of values, for example, student's name and its id are the data about the student.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

Structure : Way of organising information , so that it is easier to use.

Data organization, in broad terms, refers to the method of classifying and organizing data sets to make them more useful.

Parul® University

DIGITAL LEARNING CONTENT

What is Data Structure ?

- Data Structures are the programmatic way of storing and organizing data.
- Is a data organization , management and storage format that enables efficient access and modifications.
- Allows handling data in an efficient way.
- Plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible

- "**Way we organise our data**"

Image source : Google

What is Data Structure ?

Image source : Google

Need of Data Structure

- As applications are getting complex and amount of data is increasing day by day, there may arise the following problems:

1. Processor speed
2. Data Search
3. Multiple requests

Advantages of Data Structures

- Efficiency – access and store data
- Reusability
- Used to manage large amounts of data
- Specific data structure used for specific tasks

Image source : Google

Applications of DS

- Compiler Design
- Operating System
- DBMS
- Simulation
- Network Analysis
- AI
- Graphs
- Numerical Analysis
- Statistical Analysis Package

Algorithm-Introduction

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

Categories of Algorithm

- **Search** – Algorithm to search an item in a data structure.
- **Sort** – Algorithm to sort items in a certain order.
- **Insert** – Algorithm to insert item in a data structure.
- **Update** – Algorithm to update an existing item in a data structure.
- **Delete** – Algorithm to delete an existing item from a data structure.

Criteria for efficiency of the algorithm

- Correctness
- Implementation
- Simplicity
- Execution time
- Memory space required
- Alternative way of doing the same task.

Image source : Google

Characteristics of Algorithm

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.
- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

Image source : Google

Algorithm Complexity

- Suppose X is an algorithm and n is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

1. **Time Factor** – Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
2. **Space Factor** – Space is measured by counting the maximum memory space required by the algorithm.

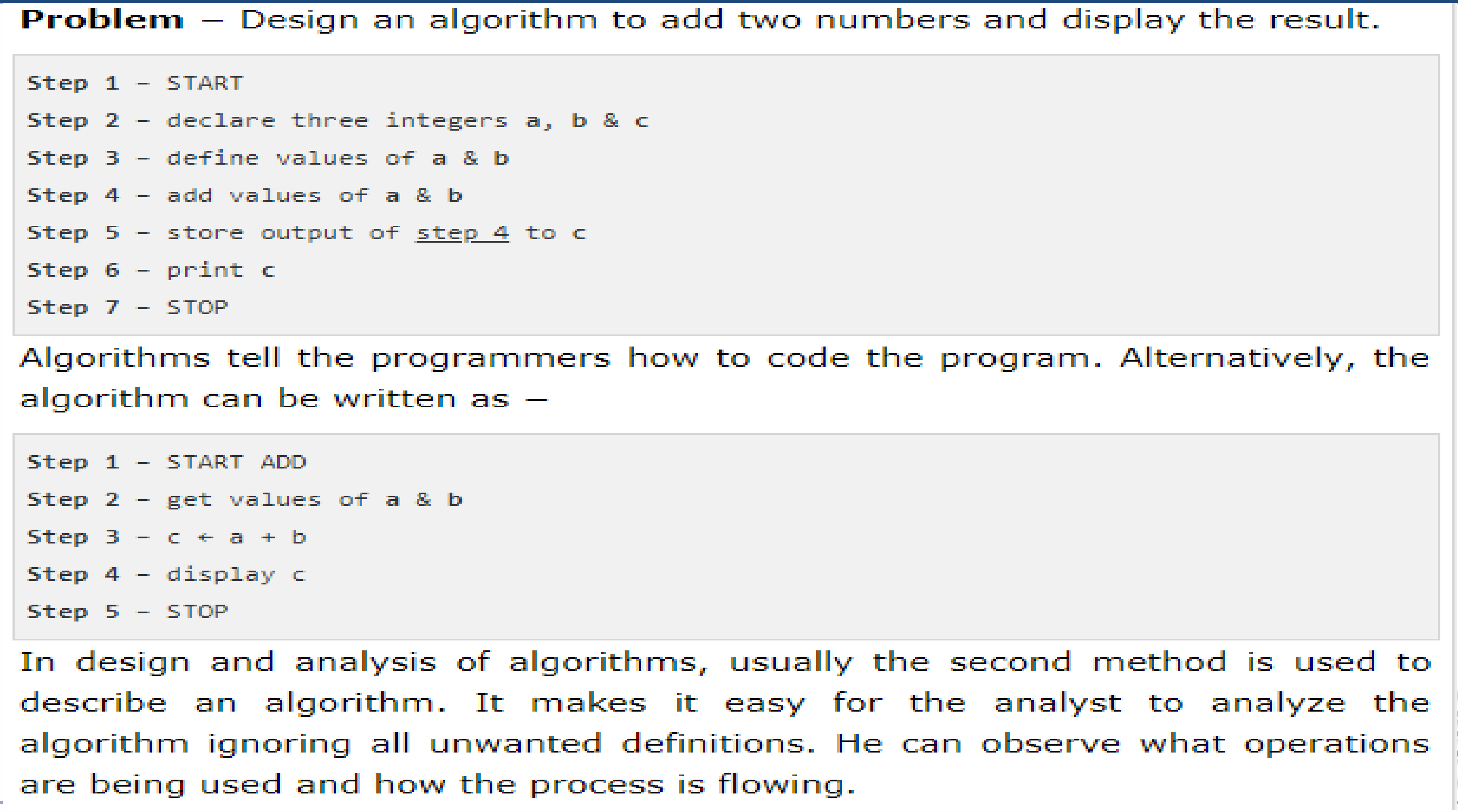
- The complexity of an algorithm f(n) gives the running time and/or the storage space required by the algorithm in terms of n as the size of input data.

Image source : Google

Parul® University

DIGITAL LEARNING CONTENT

Example



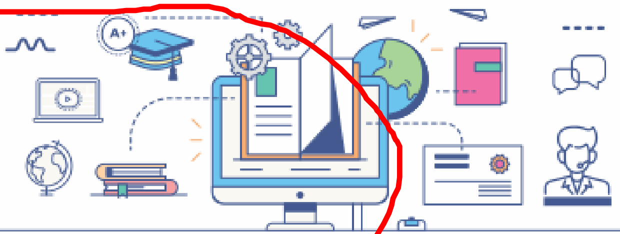
Problem − Design an algorithm to add two numbers and display the result.

```
Step 1 − START
Step 2 − declare three integers a, b & c
Step 3 − define values of a & b
Step 4 − add values of a & b
Step 5 − store output of step 4 to c
Step 6 − print c
Step 7 − STOP
```

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

```
Step 1 − START ADD
Step 2 − get values of a & b
Step 3 − c ← a + b
Step 4 − display c
Step 5 − STOP
```

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.



Not Teaches in Class..
So no need to Study

Asymptotic Analysis

- In mathematical analysis, asymptotic analysis of algorithm is a method of defining the mathematical boundation of its run-time performance.
- Using the asymptotic analysis, we can easily conclude about the average case, best case and worst case scenario of an algorithm.
- It is used to mathematically calculate the running time of any operation inside an algorithm.
- Usually the time required by an algorithm comes under three types:

1. **Worst case:** It defines the input for which the algorithm takes the huge time.
2. **Average case:** It takes average time for the program execution.
3. **Best case:** It defines the input for which the algorithm takes the lowest time

Image source : Google

Asymptotic Notations - Big oh Notation (O)

- It express the upper boundary of an algorithm running time.
- It measures the worst case of or upper bound of running time complexity or the longest amount of time, algorithm takes to complete their operation.
- It is represented as shown below:

For example:
If f(n) and g(n) are the two functions defined for positive integers, then
f(n)=O(g(n))
IFF
F(n)≤cg(n) for all n≥no, c=0 ,n0>=1
This implies that f(n) does not grow faster than g(n), or g(n) is an upper bound on the function f(n).

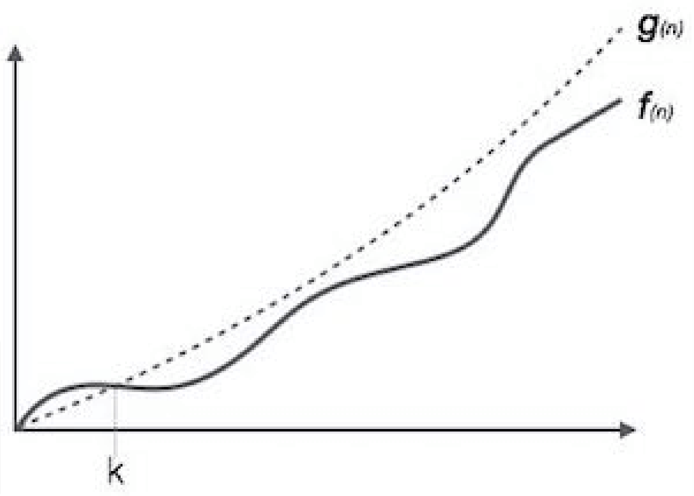


Image source : Google

Asymptotic Notations -Omega Notation (Ω)

- It is the formal way to represent the lower bound of an algorithm's running time. It measures the best amount of time an algorithm can possibly take to complete or the best case time complexity.

If we required that an algorithm takes at least certain amount of time without using an upper bound, we use big- Ω notation
For example:
If f(n) and g(n) are the two functions defined for positive integers, then
f(n)= Ω(g(n))
IFF
F(n)>=c*g(n) for all n≥no,c=0 ,n0>=1

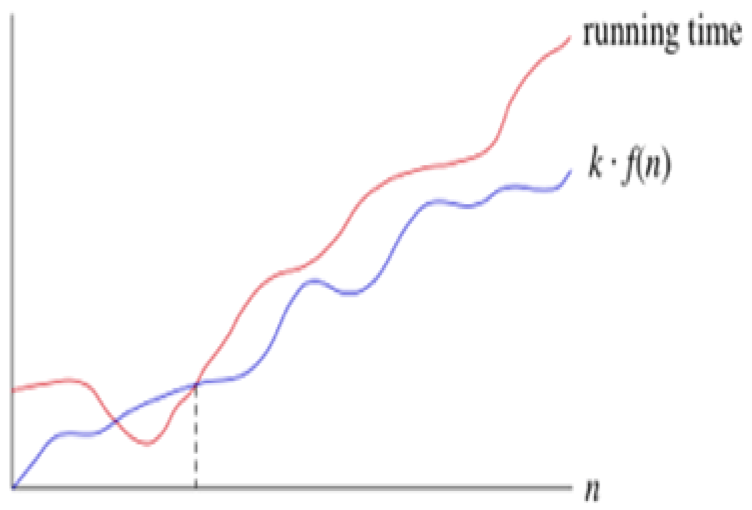


Image source : Google

Asymptotic Notations -Theta Notation (?)

- It is the formal way to express both the upper bound and lower bound of an algorithm running time.
- It is represented as shown below:
- Expressed average case scenario

For example:

If f(n) and g(n) are the two functions defined for positive integers, then

f(n)= ?(g(n))

IFF

C1*g(n)<=F(n)<=c2*g(n) for all n≥no,c=0 ,n0>=1

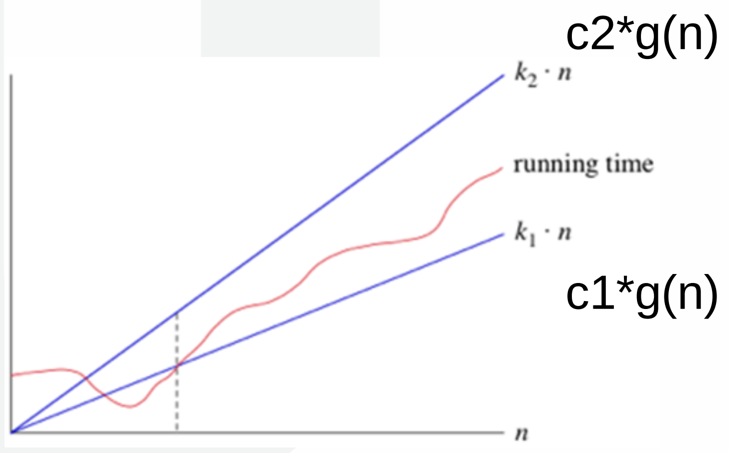


Image source : Google

Parul® University

DIGITAL LEARNING CONTENT

Greedy Algorithms

- An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain.
- As being greedy, the closest solution (localized) that seems to provide an optimum solution is chosen.
- Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions.
- However, generally greedy algorithms do not provide globally optimized solutions.

Image source : Google

Parul® University

Not Teaches in Class..
So no need to Study

DIGITAL LEARNING CONTENT

Greedy Algorithms-Examples

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

Remeber Minimum 5

Image source : Google

Divide and conquer

- In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently.
- When we keep on dividing the sub problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible.
- Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

Image source : Google

Steps of Divide and conquer Algorithm

1. Divide/Break
2. Conquer/Solve
3. Merge/Combine

Image source : Google

Dynamic Programming

- Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used.
- Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems.
- The solutions of sub-problems are combined in order to achieve the best solution.

So we can say that –

- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use Memoization.

Data type

- Data type is a way to classify various types of data such as integer, string, etc. which determines the values that can be used with the corresponding type of data, the type of operations that can be performed on the corresponding type of data.
- **There are two data types –**

1. Built-in Data Type
2. Derived Data Type

Image source : Google

Built-in Data Type

- Those data types for which a language has built-in support are known as Built-in Data types.
- For example, most of the languages provide the following built-in data types.

1. Integers
2. Boolean (true, false)
3. Floating (Decimal numbers)
4. Character and Strings

Image source : Google

Derived Data Type

- Those data types which are implementation independent as they can be implemented in one or the other way are known as derived data types.
- These data types are normally built by the combination of primary or built-in data types and associated operations on them.
- For example –

1. List
2. Array
3. Stack
4. Queue

Image source : Google

Classification of Data Structure

Remember this Diagram

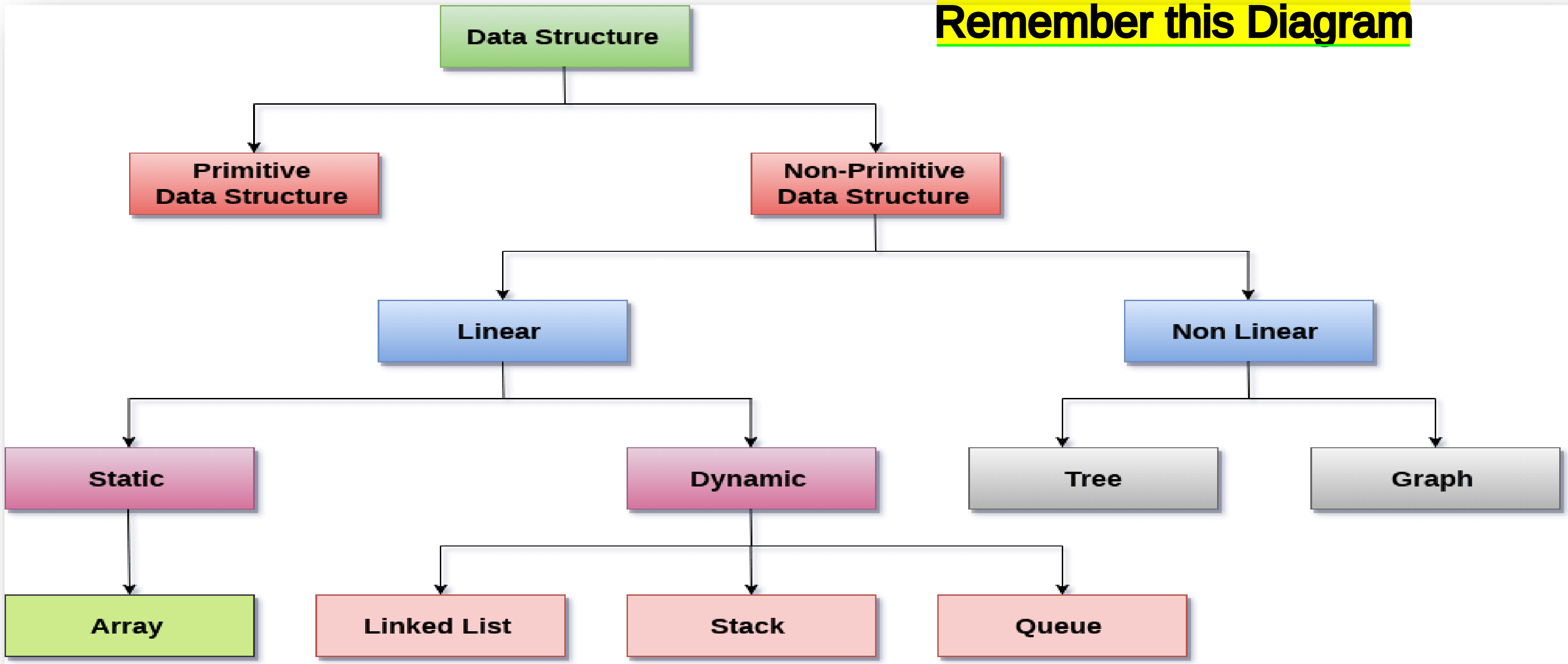


Image source : Google

Primitive Data Structure

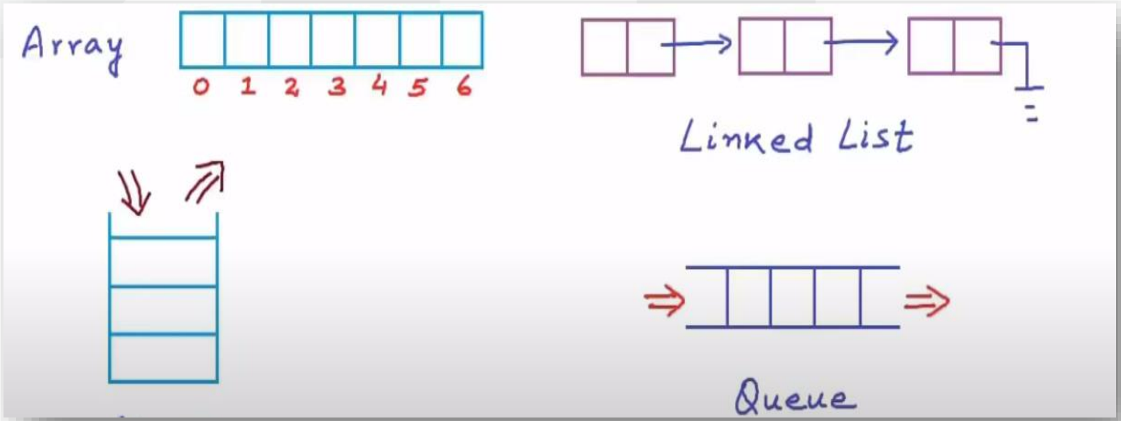
- There are basic structures and directly operated upon by the machine level instructions.
- In general, there are different representation on different computers.
- It is a fundamental data type.
- Integer, Floating-point number, Character constants, string constants, pointers etc, fall in this category.

Non-Primitive Data Structure

- These are derived from the primitive data structures.
- The non primitive data structures are not directly operated by machine level instruction.
- The non primitive data structure is also known as complex or composite data structure.
- The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.
- Linked Lists, Stack, Queue, Tree, Graph are example of non-primitive data structures.

Linear Data Structure

- Data structure where data elements are arranged sequentially or linearly where the elements are attached to its previous and next adjacent in what is called a linear data structure.
- In linear data structure, single level is involved. Therefore, we can traverse all the elements in single run only.
- Linear data structures are easy to implement because computer memory is arranged in a linear way. Its examples are array, stack, queue, linked list, etc.



Parul® University

DIGITAL LEARNING CONTENT

Non-Linear Data Structure

- Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures.
- In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only.
- Non-linear data structures are not easy to implement in comparison to linear data structure.
- It utilizes computer memory efficiently in comparison to a linear data structure. Its examples are trees and graphs

Image source : Google

Operations on DS

- Creation
- Selection
- Updation
- Deletion

Image source : Google

Creation

- This operation is used to create a data structure.
- e.g. In ‘C’ language using declaration statement

  int n = 45;
- Once any data is declared, it is associated with memory space, address and the value.

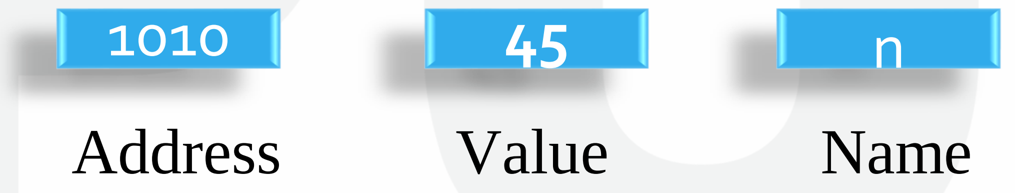


Image source : Google

Selection

- This operation is used to access data within data structure
- Selection is used to refer the value stored in data structure
- e.g. created data structure is int a = 10;
- So value 10 is referred from created data structure.

Image source : Google

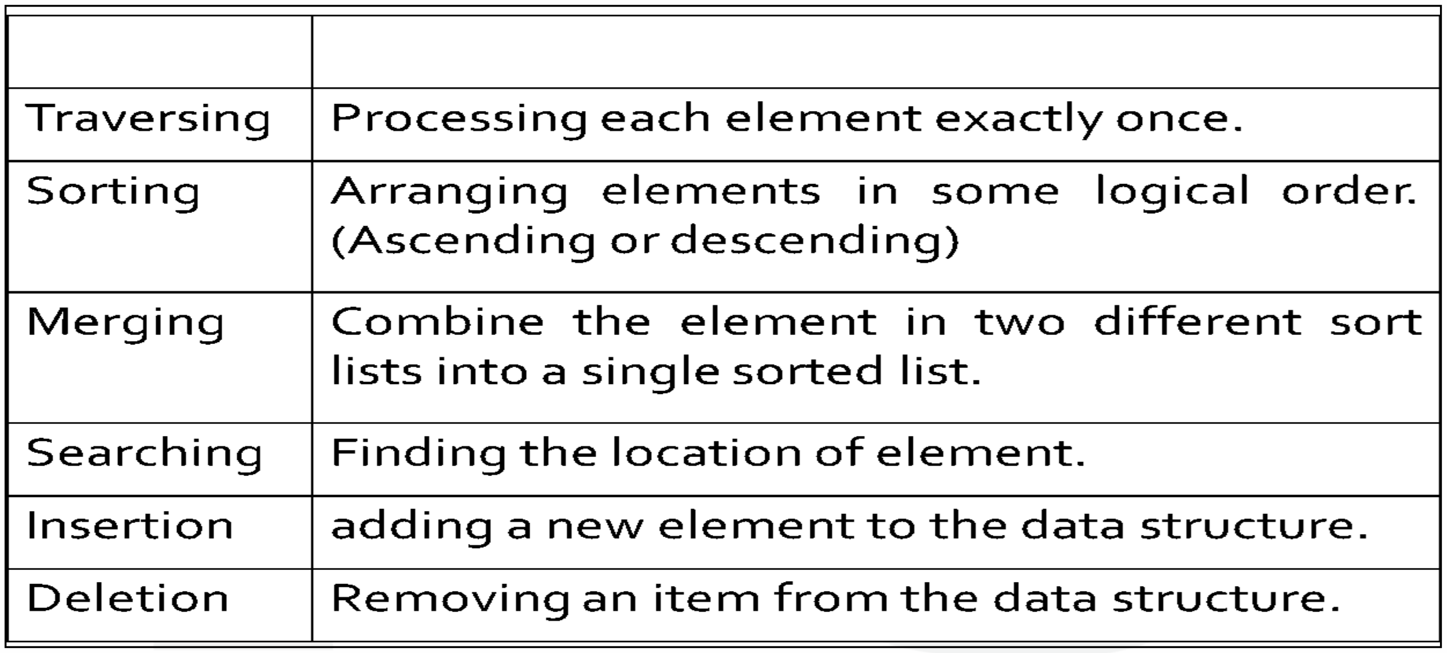
Deletion

- Once the structure is used and its purpose is over, it is to be destroyed.

Image source : Google

Operations on Non-Primitive Data Structure

Remember this



Traversing	Processing each element exactly once.
Sorting	Arranging elements in some logical order. (Ascending or descending)
Merging	Combine the element in two different sort lists into a single sorted list.
Searching	Finding the location of element.
Insertion	adding a new element to the data structure.
Deletion	Removing an item from the data structure.

Image source : Google

Parul® University

DIGITAL LEARNING CONTENT

Pointers

- Pointer is used to points the address of the value stored anywhere in the computer memory.
- To obtain the value stored at the location is known as dereferencing the pointer.

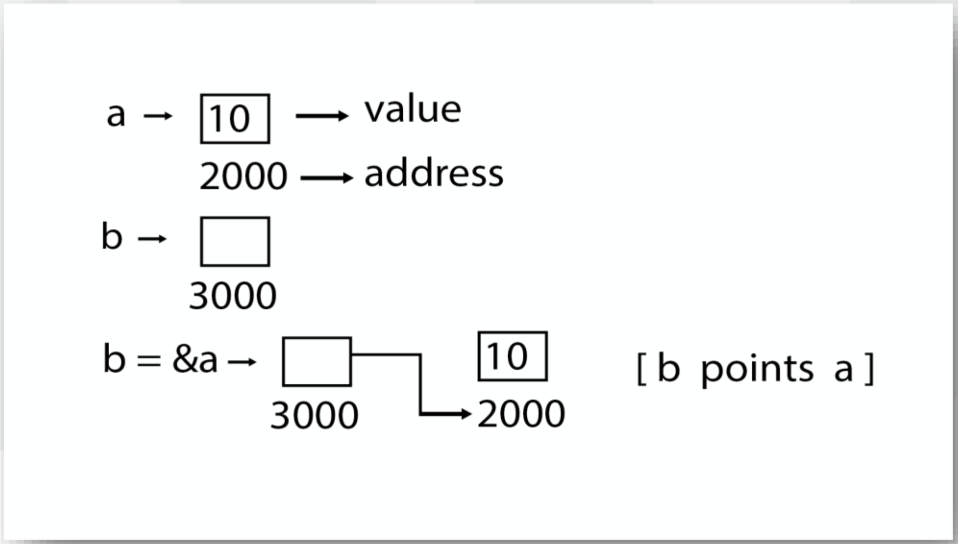
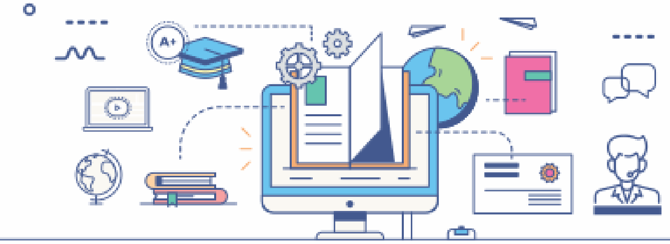


Image source : Google



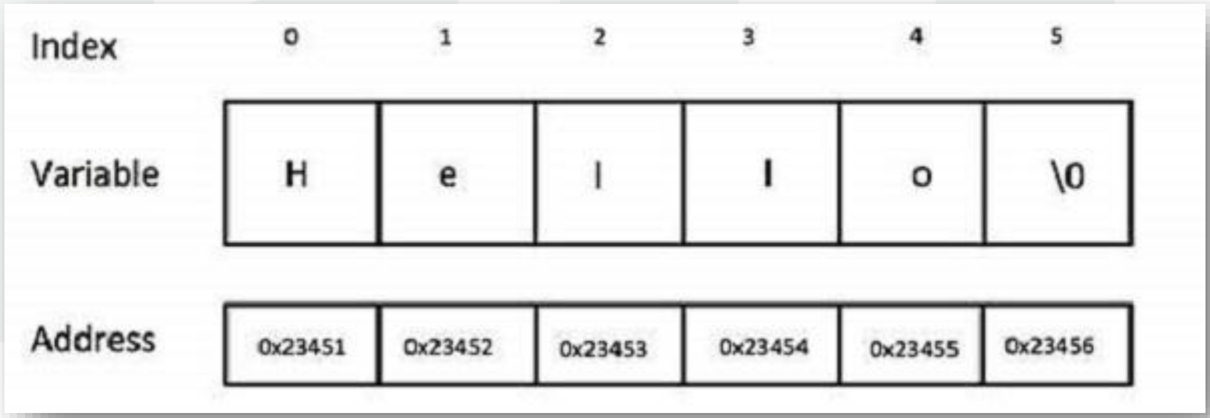
Strings

- In computer programming, a string is traditionally a sequence of characters, either as a literal constant or as some kind of variable.
- A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding.
- String may also denote more general arrays or other sequence (or list) data types and structures.

Image source : Google

Strings

- in C string is stored in an array of characters. Each character in a string occupies one location in an array.
- The null character '\0' is put after the last character.
- This is done so that program can tell when the end of the string has been reached.
- For example, the string "Hello" is stored as



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

Image source : Google

Strings

- Since the string contains 5 characters. it requires a character array of size 6 to store it. the last character in a string is always a NULL('\0') character.
- Always keep in mind that the '\0' is not included in the length if the string, so here the length of the string is 5 only. Notice above that the indexes of the string starts from 0 not one so don't confuse yourself with index and length of string.

Parul® University

DIGITAL LEARNING CONTENT

String functions

- #include<string.h>
- include this library to your program to use some of the inbuilt string manipulation functions present in the library.
- There are about 20-22 inbuilt string manipulating functions present in this library.

1.strlen("name of string")
2. strcpy(dest, source)
3. strcmp(string1, string2)
4. strstr(str1, str2) ➔ **Searches for one string inside another string.**

Image source : Google

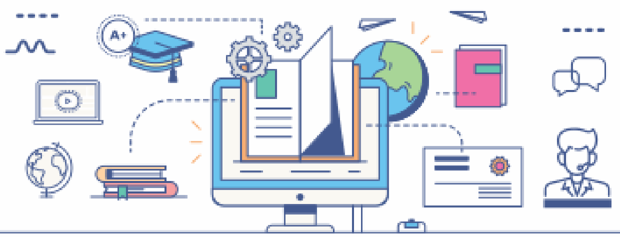
strlen(string)

- Declaration and usage of strlen() function
- It is one of the most useful functions used from this library, whenever we need the length of the string say "str1"
- we write it as

```
int len;
len = strlen(str1);
```

- len will be the length of the string "str1".(it will include all the spaces and characters in the string except the NULL(‘\0') character.

Image source : Google



strcpy(dest, source)

- This function copies the string "source" to string "dest".
- Declaration

  strcpy(str2, str1);
- Return value

  This returns a pointer to the destination string dest.

Image source : Google

strcmp(str1, str2)

- This function is used to compare two strings "str1" and "str2". this function returns zero("0") if the two compared strings are equal, else some none zero integer.
- Declaration and usage of strcmp() function

  strcmp(str1 , str2); //declaration //using this function to check given two strings are equal or not.

```
if( strcmp( str1, str2 )==0)
{
        printf("string %s and string %s are equal\n",str1,str2);
}
else
        printf("string %s and string %s are not equal\n",str1,str2);
```

Now Chapter 1 is End

Image source : Google

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in