2 DATA STRUCTURES

INTRODUCTION TO DATA STRUCTURE:

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artificial intelligence, Graphics and many more.

Basic Terminology

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned.

Data: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.**00:28/00:37**

Group Items: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

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.

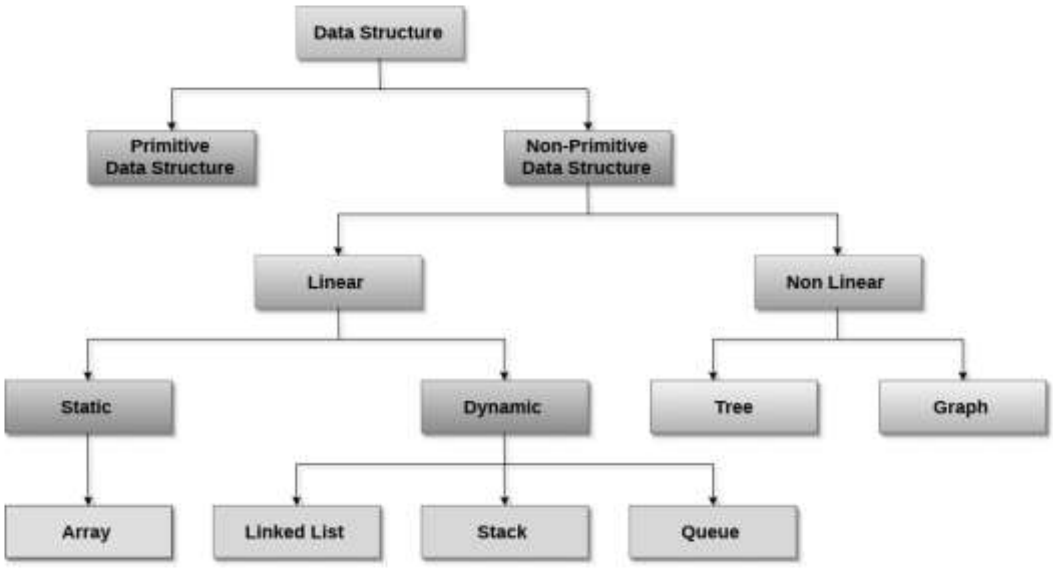
File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

Field: Field is a single elementary unit of information representing the attribute of an entity.

Elementary items: A data item that does not have subordinate data items is called an elementary item.

Data Structure Classification



Linear Data Structures: A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element.

Types of Linear Data Structures are given below:

Arrays: An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

The elements of array share the same variable name but each one carries a different index number known as subscript. The array can be one dimensional, two dimensional or multidimensional.

The individual elements of the array age are:

age[0], age[1], age[2], age[3],......... age[98], age[99].

Linked List: Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.

Stack: Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**.

A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

Queue: Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**.

It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.

Non Linear Data Structures: This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure.

Types of Non Linear Data Structures are given below:

Trees: Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the hierarchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.

Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have almost one parent except the root node. Trees can be classified into many categories which will be discussed later in this tutorial.

Graphs: Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree cannot have the one.

OPERATIONS ON DATA STRUCTURE

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

   Example: If we need to calculate the average of the marks obtained by a student in 6 different subject, we need to traverse the complete array of marks and calculate the total sum, then we will divide that sum by the number of subjects i.e. 6, in order to find the average.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.
   If the size of data structure is **n** then we can only insert **n-1** data elements into it.

3) **Deletion:** The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.
   If we try to delete an element from an empty data structure then **underflow** occurs.

4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** Combining the records in two different sorted files into a single sorted file. When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging

ALGORITHMIC NOTATION

An algorithm is a finite step –by–step list of well–defined instructions for solving a particular problem. The form for formal representation of an algorithm consists of two parts,

(1) It tells the purpose of algorithm, identification variables, which occur in algorithm and lists, input data.
(2) It contains list of steps that is to be executed.
There are certain conventions, which has to be followed in algorithms.

(i) Step, control, exit
The steps of algorithm are executed one after another beginnings with step 1.
Sometimes control may be transferred to step 'n' of algorithm by statement 'goto step n' or using control structures.
The algorithm is completed with the statement 'Exit'.

(ii) **Comments**
It is given in brackets either at beginning or end of step. It indicates the main purpose of step.

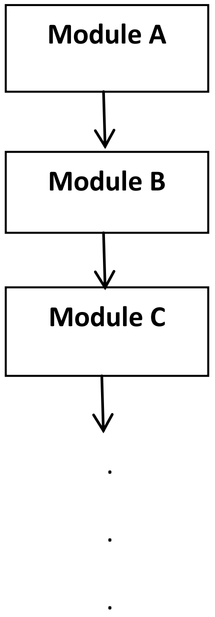
(iii) **Variables**
Variable names are generally given in capital letters. Also variables used as counters or subscripts are in capitals.
Assignment Statement
It uses: = notation
e.g. MAXIMUM: = 5
where MAXIMUM is a variable

(iv) **Input/Output**
For input read statement is used
Read: Variable name
For output write/print statement is used.
Write: Messages/Variable names

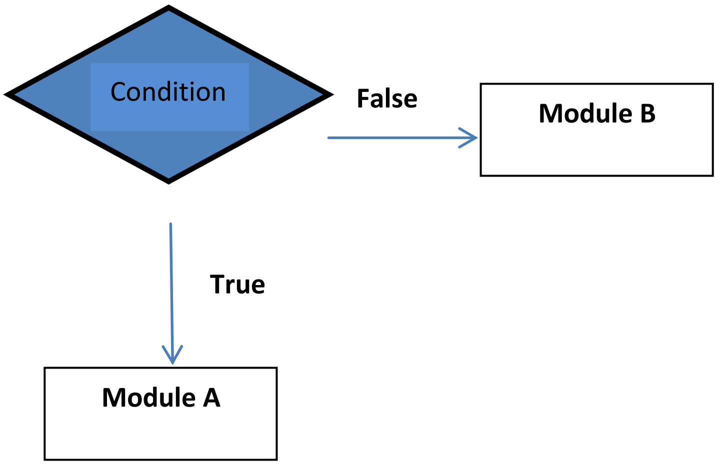
CONTROL STRUCTURES

Control structures are a way to specify flow of control in programs. Control structure analyses and chooses the direction a program flows. It is based on following parameters:

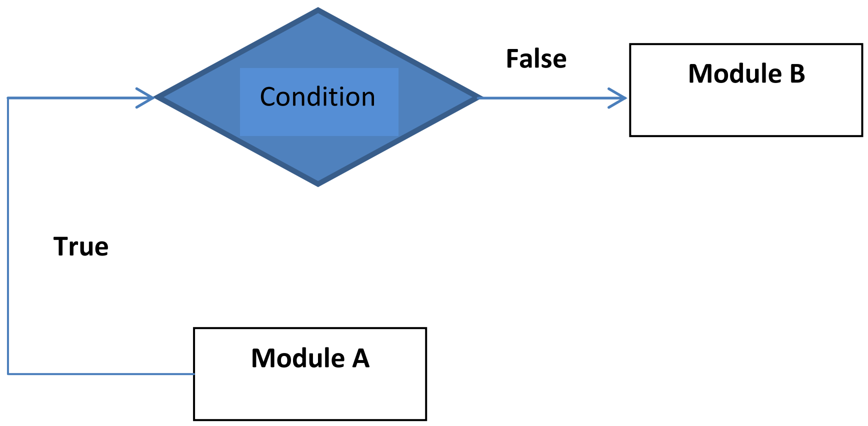
1) **Sequence control structure:** In sequence control structure the execution of the program statements happen line by line in a sequential manner following exactly the way they are written in the computer program. Only one statement gets executed at a time and it follows the top-down approach until or unless they are directed by other control structures.



2) **Selection control structure:** Sequential control structure works on the basis of true or false conditions. If the condition mentioned in the program becomes true then the program statements under the true block will be executed. If the condition mentioned in the program becomes false then the program statements under the false block will be executed.



3) **Iterative control structure:** Iterative control structure is a set of statements which get repeatedly executed until a condition becomes false.



<u>ARRAYS IN DATA STRUCTURE</u>

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

OR

Array is a container which holds a fixed number of items and these items should be of the same type stored at continuous memory location. Most of the data structures make use of arrays to implement their algorithms.

In C programming, they are the derived data types that can store the primitive type of data such as int, char, double, float, etc. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define a different variable for the marks in different subjects. Instead, we can define an array that can store the marks in each subject at the contiguous memory locations.

Properties of array

There are some of the properties of an array that are listed as follows -

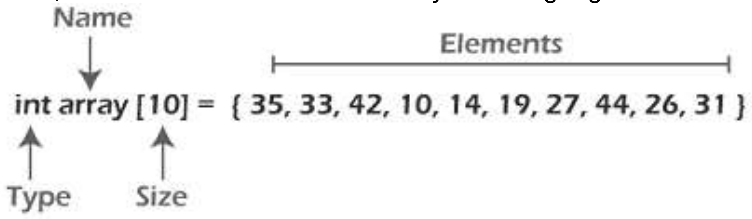
- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an array in memory

We can represent an array in various ways in different programming languages.
Following are the terms related to array:-

1) Element- Each item stored in the array are called element
2) Index- Each location of an element in an array has a numerical index which is used to identify the element.

As an illustration, let's see the declaration of array in C language -



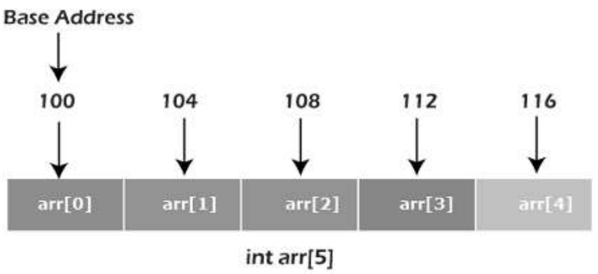
As per the above illustration, there are some of the following important points -

- Index starts with 0.
- The array's length is 10, which means we can store 10 elements i.e array size-1. So the index numbers will range from 0 to 9.
- Each element in the array can be accessed via its index.

Memory allocation of an array

We can define the indexing of an array in the below ways -

1. 0 (zero-based indexing): The first element of the array will be arr[0].
2. 1 (one-based indexing): The first element of the array will be arr[1].
3. n (n - based indexing): The first element of the array can reside at any random index number.



In the above image, we have shown the memory allocation of an array arr of size 5. The array follows a 0-based indexing approach. The base address of the array is 100 bytes. It is the address of arr[0]. Here, the size of the data type used is 4 bytes; therefore, each element will take 4 bytes in the memory.

BASIC ARRAY OPERATIONS

Following are the operations performed on an array.

- Traversal - This operation is used to print the elements of the array.
- Insertion - It is used to add an element at a particular index.
- Deletion - It is used to delete an element from a particular index.
- Search - It is used to search an element using the given index or by the value.
- Update - It updates an element at a particular index.

The three most basic operations are

1) Insert: Adds an element at the given index number.

Algorithm to insert element in array

STEP 1:Get the element value which needs to be inserted

Step 2: Get the position value.

Step 3: Check whether the position value is valid or not.

Step 4: If it is valid

Shift all the elements from the last index to position index by 1.

Insert the new element in array position

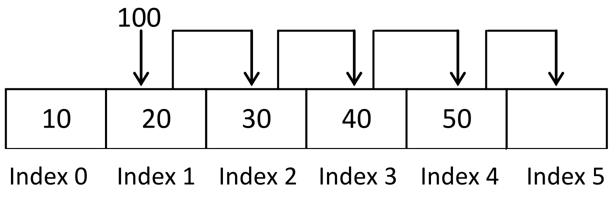
arr[position]

arr[1] : here arr is the name of the array and 1 is the index number

Step 5: Otherwise invalid position

Diagrammatic Representation:-

<u>Before inserting new element 100 at index 1.</u>



<u>After inserting new element 100 at index 1.</u>

10	100	20	30	40	50
Index 0	Index 1	Index 2	Index 3	Index 4	Index 5

Algorithm

Step 1: We need to insert element 100 at position 1

Step 2: Move all the elements from the last index to the position where the elements has to be inserted by 1.

Element at arr[4] will be placed in arr[5].

Element at arr[3] will be placed in arr[4].

Element at arr[2] will be placed in arr[3].

Element at arr[1] will be placed in arr[2].

Step 3: finally element 100 is placed at position 1.

2) Delete Operation: Deletes an element from the given index number.
Algorithm to delete an element from the array

Step 1: Find the given element in the array and note the index number.

Step 2: If element is found

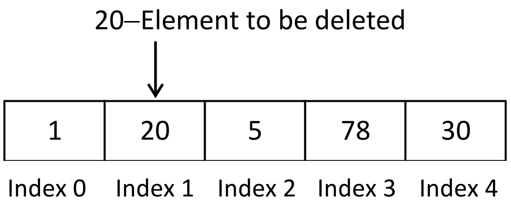
Shift all the elements from index+1 by 1 position to the left

Reduce the size of the array.

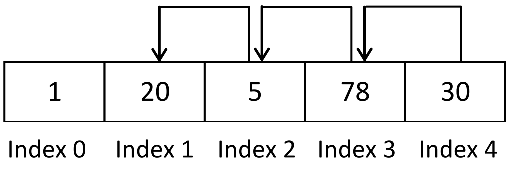
Step 3: Otherwise print element not found.

Diagrammatic representation

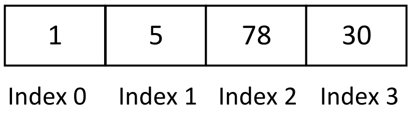
Step 1:



Step 2:



Step 3:



Algorithm:
Step 1: Take an array of 5 integers {1,20,5,78,30}. We need to remove element 20(index value=1) from the array.
Step 2: Shift all the elements from index+1(i.e. from 2 to 3) by 1 position to the left.
Element at arr[2] will be placed in arr[1].
Element at arr[3] will be placed in arr[2].
Element at arr[4] will be placed in arr[3].
Step 3: Finally a new array is generated
Elements are: {1,5,78,30}

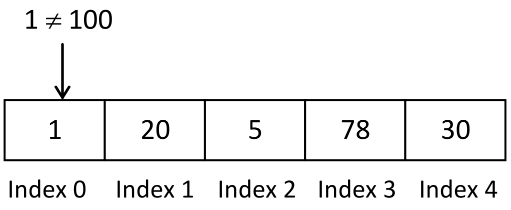
3) Search: Searches an element using the given index number.
Algorithm to search an element from the array
Step 1: Iterate the array using the loop.
arr[5]={34,2,23,100,60}
search key=100

Step 2: Check whether the given key is present in the array or not i.e. arr[i]= = key.

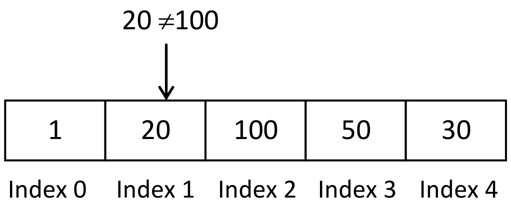
Step3: If Yes
    Print search found or print element found

Step 4: Else
    Print search not found

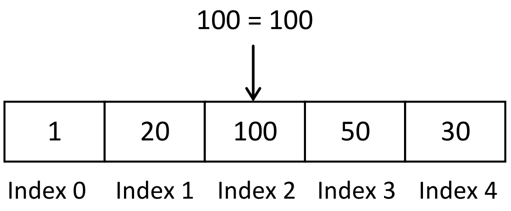
Diagrammatic representation:
CASE 1: Search found
Step 1:



Step 2:



Step 3:



Step 4: Print search found

Algorithm:

Case 1: Consider the scenario where the element to be searched is present in the array.

The array elements are-1,20,100,50,30.

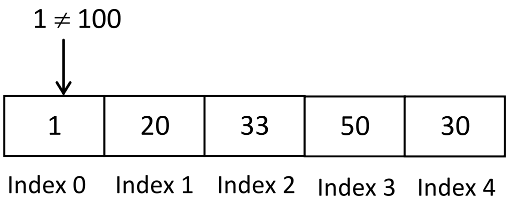
Step 1: Move the pointer to next element since 1 is not equal to 100.

Step 2: Move the pointer to next element since 20 is not equal to 100.

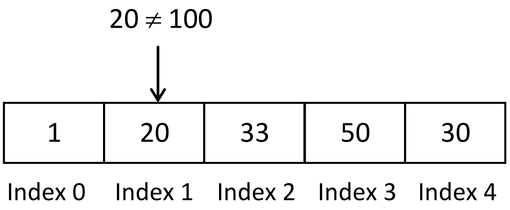
Step 3:Element 100 matches the search key value=100. Print element found or search found.

CASE 2: Search not found

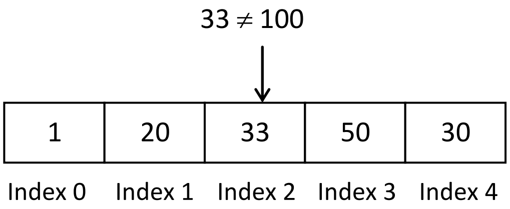
Step 1:



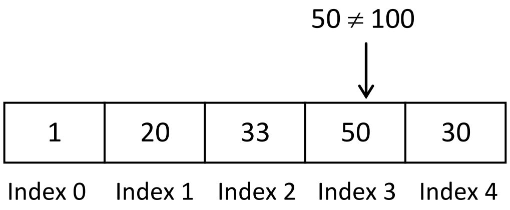
Step 2:



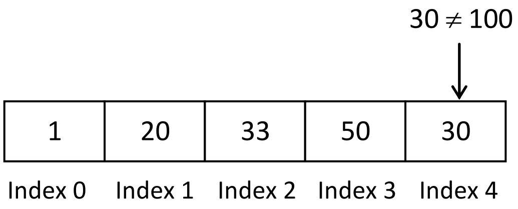
Step 3:



Step 4:



Step 5:



Step 6: Print search not found

Algorithm:

Case 2: Consider the scenario where the element to be searched is not present in the array.

The array elements are-1,20,33,50,30.

Step 1: Move the pointer to next element since 1 is not equal to 100.

Step 2: Move the pointer to next element since 20 is not equal to 100.

Step 3: Move the pointer to next element since 33 is not equal to 100.

Step 4: Move the pointer to next element since 50 is not equal to 100.

Step 5: Move the pointer to next element since 30 is not equal to 100.

End of array reached.

Print search not found

SORTING ALGORITHMS

1) BUBBLE SORT

Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.

Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world. It is not suitable for large data sets.

Algorithm

In the algorithm given below, suppose **arr** is an array of **n** elements. The assumed **swap** function in the algorithm will swap the values of given array elements.

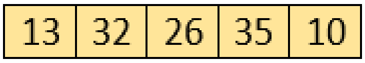
```
1. begin BubbleSort(arr)
2.  for all array elements
3.  if arr[i] > arr[i+1]
4.  swap(arr[i], arr[i+1])
5. end if
6. end for
7. return arr
8. end BubbleSort
```

Working of Bubble sort Algorithm

Now, let's see the working of Bubble sort Algorithm.

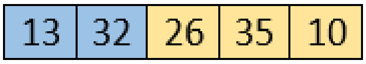
To understand the working of bubble sort algorithm, let's take an unsorted array. We are taking a short and accurate array, as we know the complexity of bubble sort is **$O(n^2)$.**

Let the elements of array are -

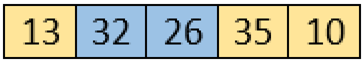


First Pass

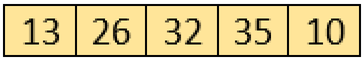
Sorting will start from the initial two elements. Let compare them to check which is greater.



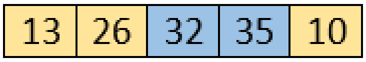
Here, 32 is greater than 13 (32 > 13), so it is already sorted. Now, compare 32 with 26.



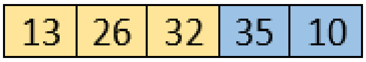
Here, 26 is smaller than 36. So, swapping is required. After swapping new array will look like -



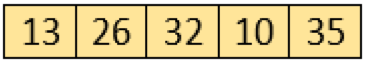
Now, compare 32 and 35.



Here, 35 is greater than 32. So, there is no swapping required as they are already sorted. Now, the comparison will be in between 35 and 10.



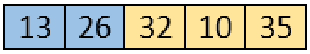
Here, 10 is smaller than 35 that are not sorted. So, swapping is required. Now, we reach at the end of the array. After first pass, the array will be -

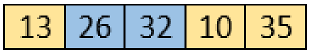


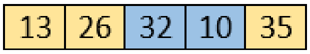
Now, move to the second iteration.

Second Pass

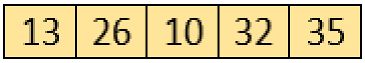
The same process will be followed for second iteration.

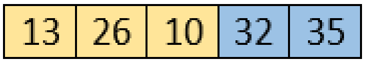






Here, 10 is smaller than 32. So, swapping is required. After swapping, the array will be -

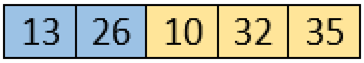


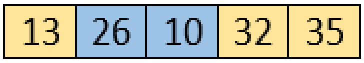


Now, move to the third iteration.

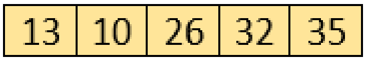
Third Pass

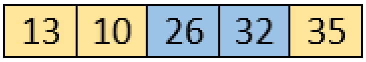
The same process will be followed for third iteration.

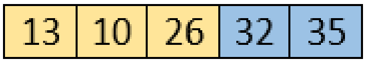




Here, 10 is smaller than 26. So, swapping is required. After swapping, the array will be -



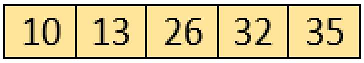




Now, move to the fourth iteration.

Fourth pass

Similarly, after the fourth iteration, the array will be -



Hence, there is no swapping required, so the array is completely sorted.

<u>SEARCHING ALGORITHM</u>

1) LINEAR SEARCH ALGORITHM

LINEAR [DTA, N, ITEM, LOC]

Here DATA is a linear array with N elements and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA or sets LOC: = 0, if search is unsuccessful.

```
1.  [Insert ITEM at the end of DATA] set DATA [N + 1]: = ITEM
2.  [Initialize counter] set LOC: = 1
3.  [Search for ITEM]
    Repeat while DATA [LOC] ≠ ITEM
        Set LOC: = LOC + 1
    [End of loop]
4.  [Successful?] IF LOC = N + 1 then set LOC: = 0
5.  Exit
```

2) BINARY SEARCH ALGORITHM

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
Binary search follows the divide and conquer approach in which the list is dividedinto two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

Algorithm

```
1.  Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search
2.  Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
3.  Step 2: repeat steps 3 and 4 while beg <=end
4.  Step 3: set mid = (beg + end)/2
5.  Step 4: if a[mid] = val
6.  set pos = mid
7.  print pos
8.  go to step 6
9.  else if a[mid] > val
10. set end = mid - 1
11. else
12. set beg = mid + 1
13. [end of if]
14. [end of loop]
15. Step 5: if pos = -1
16. print "value is not present in the array"
17. [end of if]
18. Step 6: exit
```

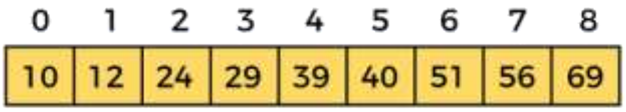
Working of Binary search

Now, let's see the working of the Binary Search Algorithm.

To understand the working of the Binary search algorithm, let's take a sorted array. Itwill be easy to understand the working of Binary search with an example.
There are two methods to implement the binary search algorithm -

- Iterative method
- Recursive method

The recursive method of binary search follows the divide and conquer approach.Let the elements of array are -

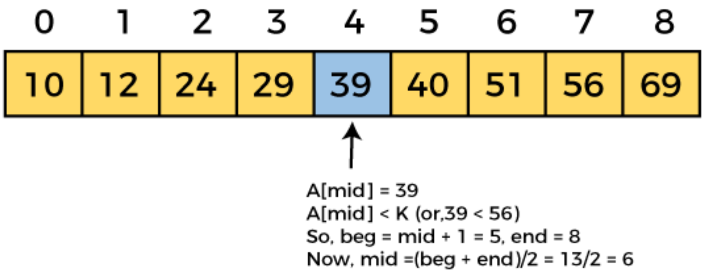


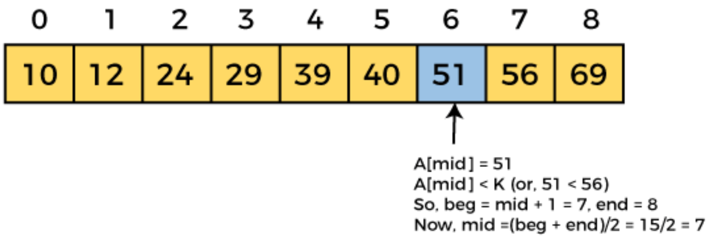
Let the element to search is, **K = 56**

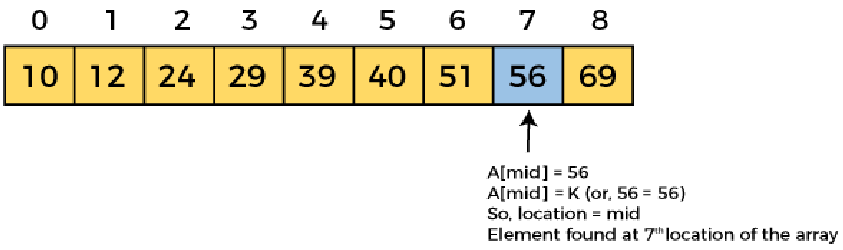
We have to use the below formula to calculate the **mid** of the array -

1. mid = (beg + end)/2 So, in the given
   array -**beg** = 0
   end = 8

   mid = (0 + 8)/2 = 4. So, 4 is the mid of the array.







Now, the element to search is found. So algorithm will return the index of theelement matched.

POINTERS ARRAY

An array whose each element is a pointer is called pointer array.

e.g. int * p[4] – pointer array

int a [4] – array of integers

p[0] = & a[0] p[1] = & a[1]

p[2] = &a[2] p[3] = &a[3]

This pointer array p holds the address of each element in array a so it is called pointer array.

Detailed Example of Pointer Array

Suppose TEAM in an array, which contains the locations of different terms or more specifically locations of first elements in different teams.

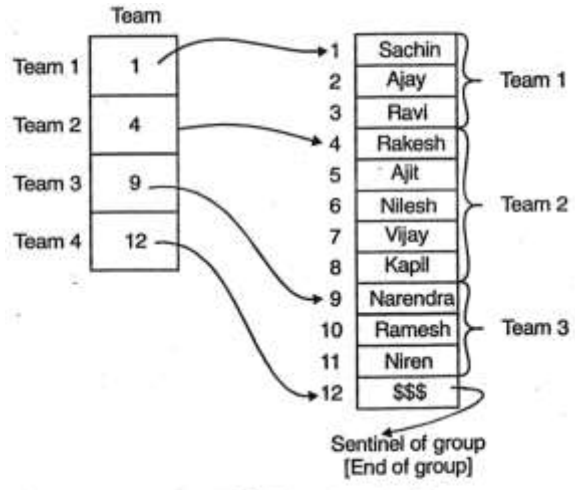


Fig. Example of an Array

Team [L] and Team [L + 1]–1 contains respectively the first and last elements in Team I.

e.g. Team 1 $\longrightarrow$ 1 $\longrightarrow$ first element of Team 1

Team [1 + 1] – 1 $\longrightarrow$ 4 $\longrightarrow$ last element of Team 1

Team 4 points to the sentinel of list.

RECORD

A record is collection of related data items. Each data items is termed as field. File is collection of similar records. Each data item may be a group item composed of sub items.

Comparison

	Record	**Linear Array**
(1)	A record may be a collection of non homogeneous data; i.e. data items in a record may have different data types.	The data items in an array may have same data tyres.
(2)	The data items in a record are indexed by attributes. So there may not be a natural ordering if its elements	There may be natural ordering of its elements.

Example of Record

Suppose a college keeps a record of each student, which contains the following data items.

ITEM	SUBITEM
Name	Last, First, MI (Middle Initials)
Address	
Stream division	
Phone	

The structure of above record is described as follows:

```
1. Student
        2. Name
                3. Last
                        3. First
                        3. MI (Middle Initial)
        2. Address
        2. Phone
        2. Stream
                3. Division
```

The number to the left of item name (i.e. identifying) represents a level number. Sub items follow each group item and the level of sub items is 1 more than the level of group item.

How to access item in record

Suppose we want to know the last name of student then it is accessed using dot operator as follows.

Student.name.last

Generally for separating sub items from group items dot is used.

Representation of records in memory

Since records contain nonhomogeneous data it can't stored in array. Some higher–level languages are having built in record structure (for e.g. PL/1, PASCAL, COBOL). Suppose these types of structures are not available in language then record has to be stored in individual variables. But if we want to keep entire file of records. Then all data elements belonging to the same identifier will be of same type. So a file may be stored in memory as collection of arrays.

e.g. Student file

Name	**Address**	**Phone**	**Stream**
Kulkarni Smita L.	19, Shaniwar Peth	4456184	Arts
Patil Ajit D.	24, M.G. Road	6361621	Commerce
–	–	–	–
–	–	–	–
–	–	–	–
	–		

Here elements in arrays name, address, phone, and stream, with same subscript belong to the same record.

LINKED LIST

Linked list or one–way list is a linear collection of data elements called nodes here the linear order is given by means of pointers. Each node is divided into two parts:

1. First part contains the information of the element.
2. Second part contains the address of next node in list (it is called link field).

START is a pointer variable, which contains the address of first node.

Left part represents the information part of the node while right part represents the next pointer field of node. The pointer of last node contains a special value called NULL, which is an invalid address.

The schematic diagram of a linked list with 4 nodes is shown below,

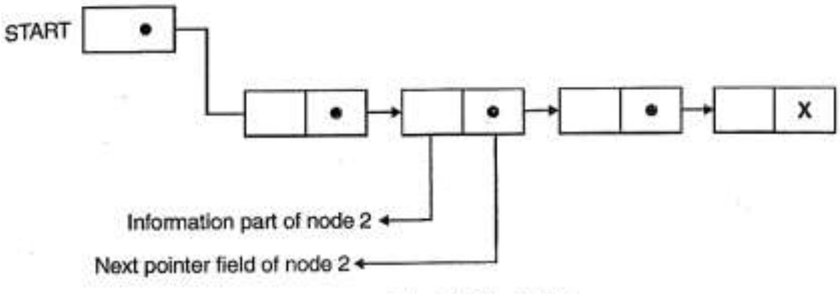
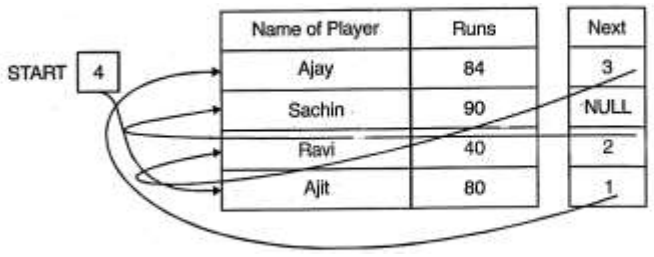


Fig. A Linked List

The following diagram shows a linked list

(1) Name of player and no. of runs from an information part while next pointer field gives the next node's address in linked list.

(1) Start $\longrightarrow$ 4
(2) 4 $\longrightarrow$ 1
(3) 1 $\longrightarrow$ 3
(4) 3 $\longrightarrow$ 2
(5) 2 $\longrightarrow$ last node



Name of Player	Runs	Next
Ajay	84	3
Sachin	90	NULL
Ravi	40	2
Ajit	80	1

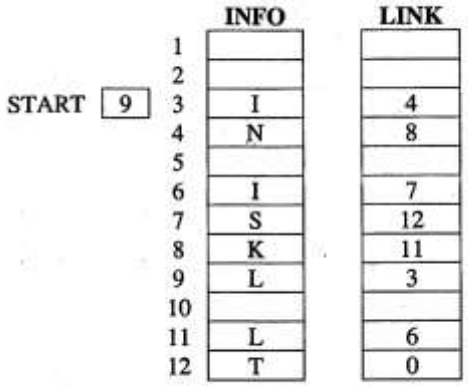
Representation of Linked List in memory

Let list be a linked list. Then List will be maintained in memory as follows,

(1) List requires 2 arrays; we will call them here INFO and LINK such that INFO [K] and LINK [K] contain respectively the information part and next pointer field of a node K of list.

(2) LIST also requires a variable name such as START, which contains the location of beginning of list and next pointer sentinel denoted by NULL, which indicates end of list.

The following e.g. shows how linked list is stored in memory. The nodes of the list need not occupy adjacent elements in the arrays INFO and LINK and more than one list may be maintained in the same linear arrays INFO and LINK. But each list must have its pointer variable giving the location of its first node.



	INFO	LINK
1		
2		
3	I	4
4	N	8
5		
6	I	7
7	S	12
8	K	11
9	L	3
10		
11	L	6
12	T	0

START 9

Linked List can be picturized in following manner:

1) START = 9, INFO [9] = L, LINK [9] = 3
2) INFO [3] = 1, LINK [3] = 4
3) INFO [4] = N, LINK [4] = 8
4) INFO [8] = K, LINK [8] = 11
5) INFO [11] = L, LINK [11] = 6
6) INFO [6] = I, LINK [6] = 7
7) INFO [7] = S, LINK [7] = 12
8) INFO [12] = T, LINK [12] = 0… last node

   i.e. “LINK LIST” string is stored in form of Linked List.

Advantages of Linked List

Generally list are stored in form of arrays. But in arrays insertion and deletion is not easy. Also array size can’t be easily increased. Using linked list this problem can be solved.

<u>TREE DATA STRUCTURE</u>

In case of Link data structure, each node has link information of next node. Thus sequentially, we can access all the nodes. It will be time consuming for searching long list; because each node has information about next node only.

In order to reduce time consumption, binary tree concept was developed. Before going to binary search tree, let us learn basics of tree structure.

Tree is an hierarchical structure of collected data items. It is nonlinear structure.

Node: It is as element of list. It may be a character, string or number.

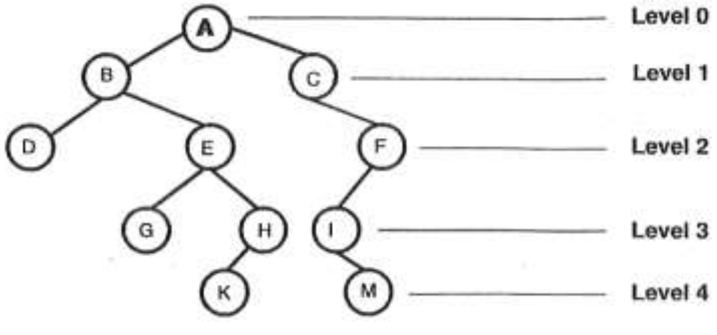
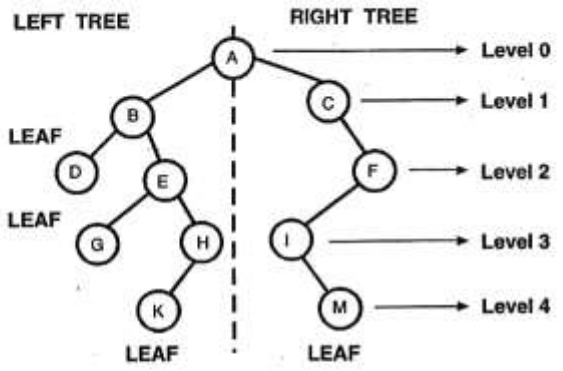


Fig. A Tree data structure

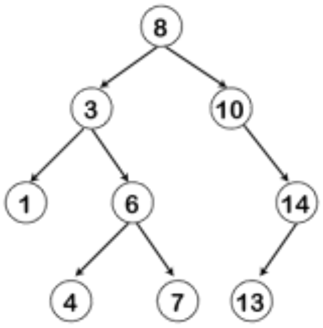
Node	**:**	**Node’s description**
ROOT	:	A node which has no parent. The node containing ‘A’ is the root of the tree.
CHILD	:	Nodes of root.
(a) LEFT CHILD	:	The left node of root.
(b) RIGHT CHILD	:	The right node of root. The node containing A has left child containing node B and right (child containing node C.)
PARENT	:	Node which has child (or children). A is parent of B and C
LEAF	:	The node which has no child (or children) D, G, K, M are leaves.
SIBLINGS	:	Two nodes have the same parent. The nodes containing D and E are both children of the node containing B. These nodes are siblings.
ANCESTOR	:	The node is an ancestor of another node, if it is a parent of that node. The ancestors of the node containing G are the nodes containing E, B, A. ROOT is always an ancestor of every node.
DESCENDANT	:	The node is a descendant of another node, if it is a child of that node. The descendant of the node containing E are the nodes containing G, H, K. All nodes in the tree are descendants of the ROOT.
LEFT SUBTREE	:	The descendant of root’s left child which acts a root of subtree.
RIGHT SUBTREE	:	The descendants of root’s right child which acts a root of subtree.
LEVEL	**:**	**The distance from the ROOT. The ROOT is always at zero (0) level.**



TYPES OF TREE

Descendant: The immediate successor of the given node is known as a descendant of a node.

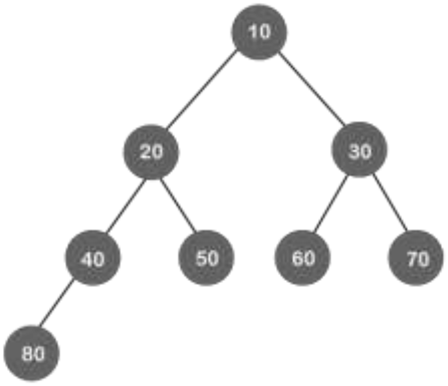
Binary tree**:** Here, binary name itself suggests two numbers, i.e., 0 and 1. In a binary tree, each node in a tree can have utmost two child nodes. Here, utmost means whether the node has 0 nodes, 1 node or 2 nodes.



Complete Binary Tree

The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.

Let's create a complete binary tree.



The above tree is a complete binary tree because all the nodes are completely filled, and all the nodes in the last level are added at the left first.

Extended Binary Tree

Extended binary tree consists of replacing every null subtree of the original tree with special nodes.

Empty circle represents internal node and filled circle represents external node.

The nodes from the original tree are internal nodes and the special nodes are external nodes.

Every internal node in the extended binary tree has exactly two children and every external node is a leaf. It displays the result which is a **complete binary tree**.

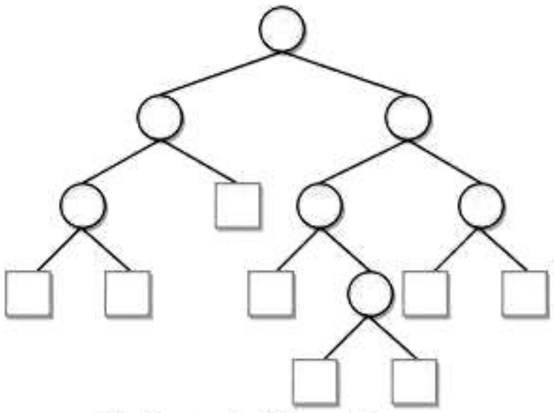
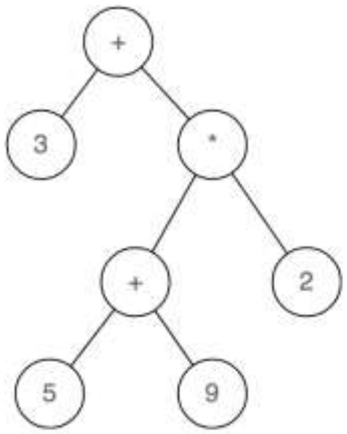


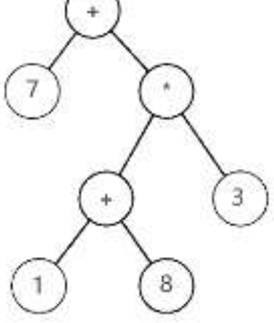
Fig. Extended Binary Tree

Expression Tree

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)*2) would be:



The expression tree is a binary tree in which each external or leaf node corresponds to the operand and each internal or parent node corresponds to the operators so for example expression tree for 7 + ((1+8)*3) would be:



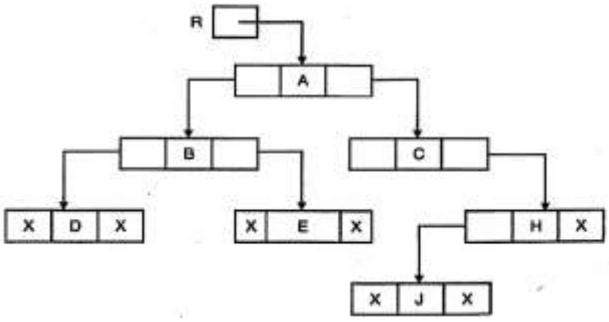
REPRESENTING BINARY TREES IN MEMORY

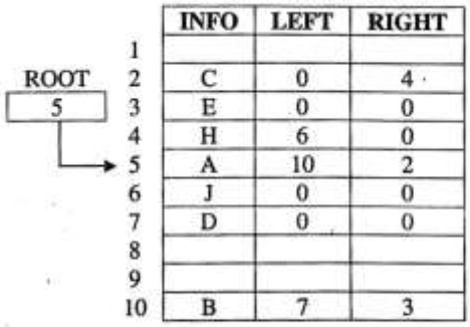
Let T be a binary tree. There are two way of representing binary trees in memory. (1) Linked representation (2) Sequential representation

1. Linked representation

   In this method 3 parallel arrays are used INFO, LEFT and RIGHT.

   (1) INFO [K] contains the data at node N.
   (2) LEFT [K] contains the location of left child of node N.
   (3) RIGHT [K] contains the location of right child of node N.
   (4) ROOT will contain the location of root R of T. It is a pointer variable.
   (5) If any subtree is empty then corresponding pointer will contain a null value.



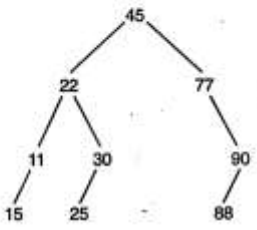


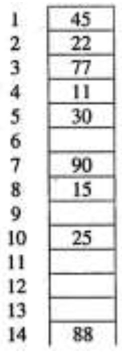
ROOT: 5

	INFO	LEFT	RIGHT
1			
2	C	0	4
3	E	0	0
4	H	6	0
5	A	10	2
6	J	0	0
7	D	0	0
8			
9			
10	B	7	3

Sequential Representation

Suppose T is complete binary tree then only single linear array TREE is used as follows:



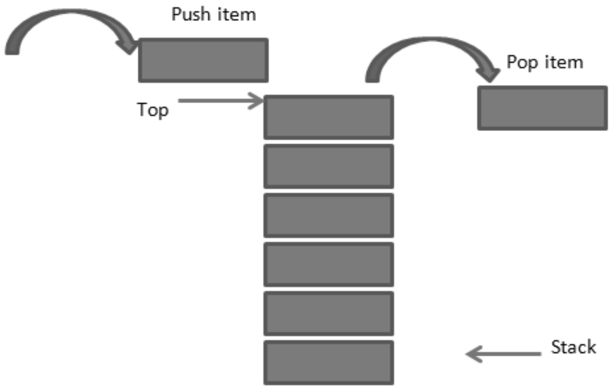


1	45
2	22
3	77
4	11
5	30
6	
7	90
8	15
9	
10	25
11	
12	
13	
14	88

1) The root R is stored in TREE [0]
1) If node N occupies TREE [K] then left child is stored in TREE [2 * K] and its right child is stored in TREE [2 * K +1]
2) If TREE [1] = NULL then it is empty.

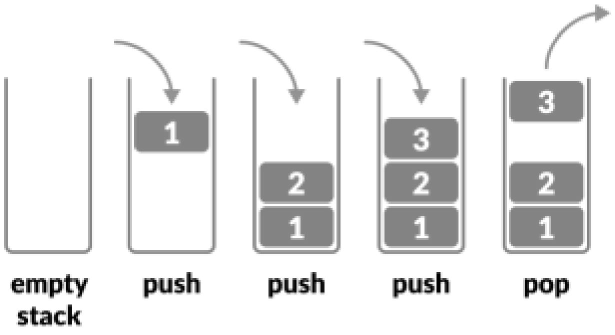
STACK DATA STRUCTURE

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

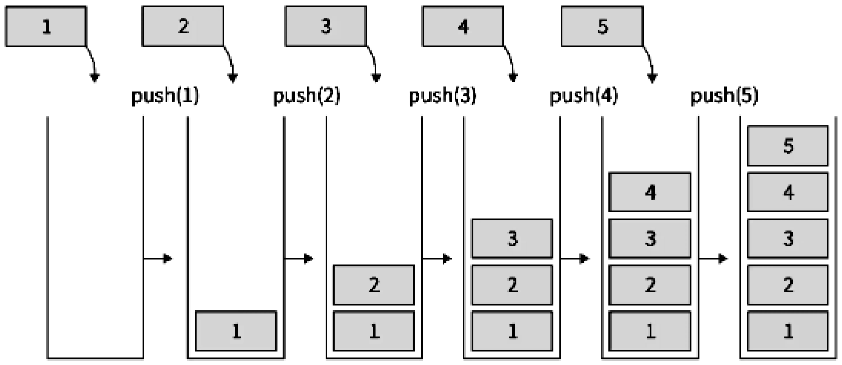


There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

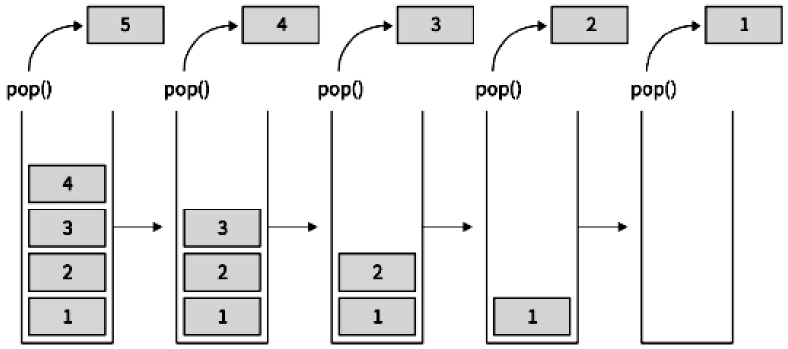
Stack Operations



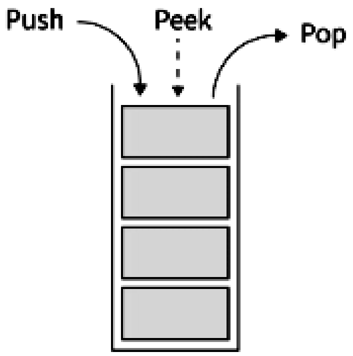
1. **Push()**
   Push is a function in stack definition which is used to insert data at the stack's top.



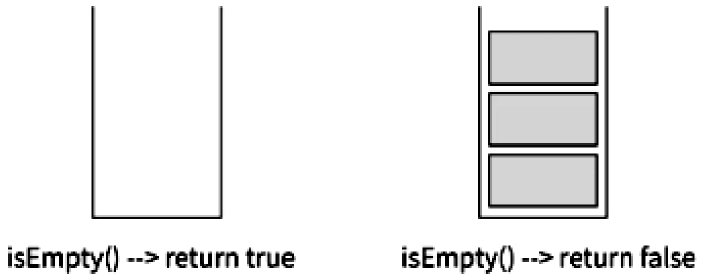
2. **pop()**
   Pop is a function in the stack definition which is used to remove data from the stack's top.



3. **topElement() / peek()**
   TopElement / Peek is a function in the stack which is used to extract the element present at the stack top.



4. **isEmpty()**
   isEmpty is a boolean function in stack definition which is used to check whether the stack is empty or not. It returns true if the stack is empty. Otherwise, it returns false.

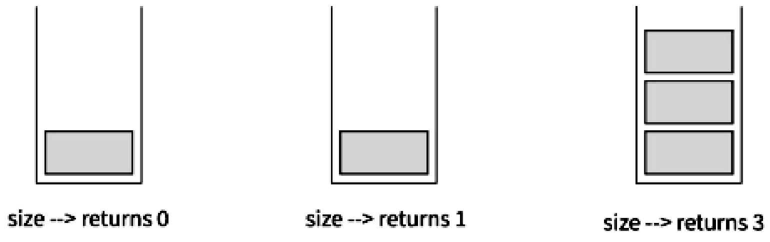


5. isFull()

isFull is a function which is used to check whether the stack has reached its maximum limit of insertion of data or not i.e. if 'maxLimit' is the maximum number of elements that can be stored in the stack and if there are exactly maxLimit number of elements present in the stack currently, then the function isFull() returns true. Otherwise, if the number of elements present in the stack currently are less than 'maxLimit', then isFull() returns false.

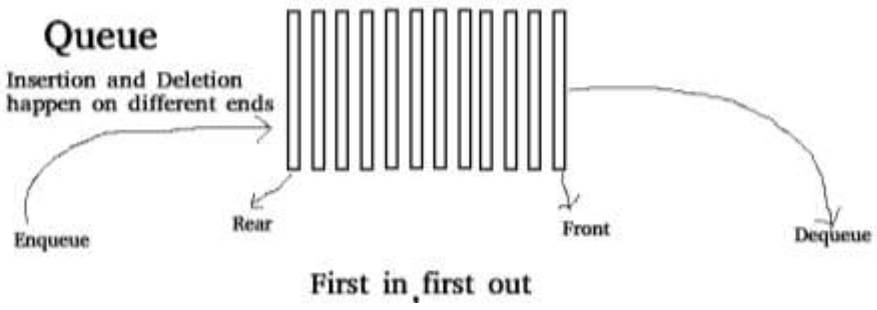
6. size()

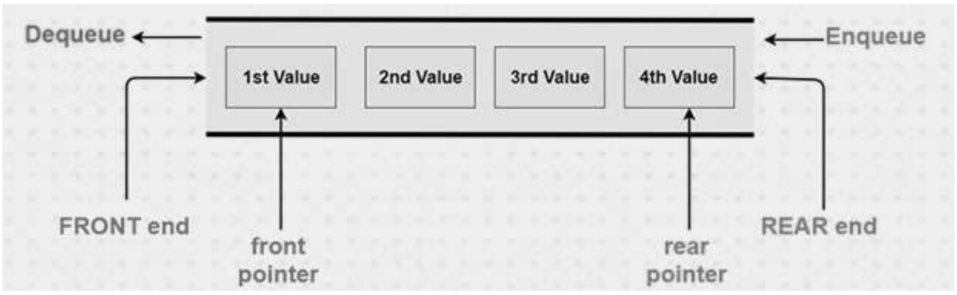
Size is a function in stack definition which is used to find out the number of elements that are present inside the stack.



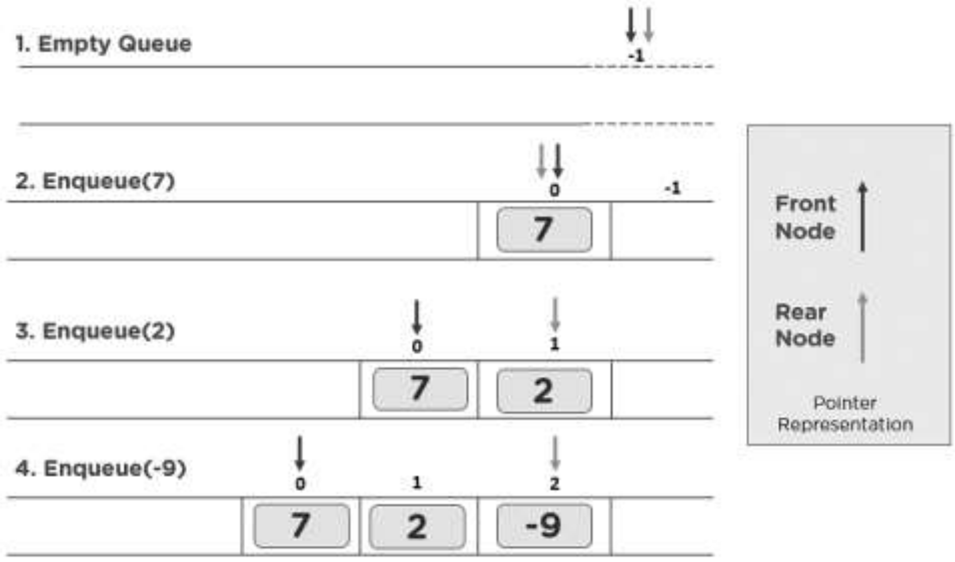
QUEUE DATA STRUCTURE

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

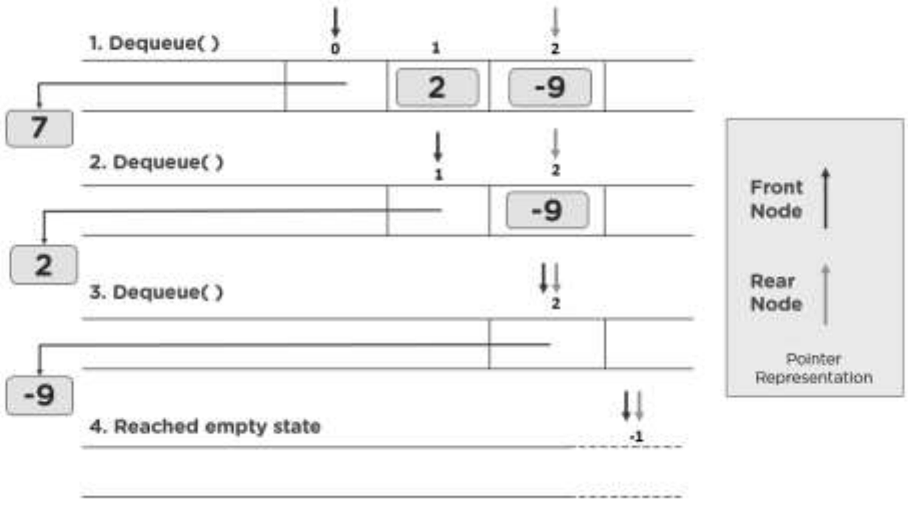




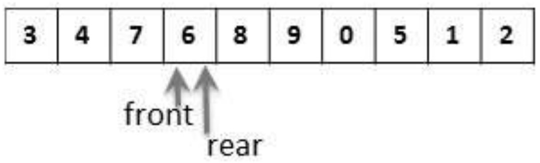
Queue Operations –
1) Enqueue() – Add item to the queue from the REAR.



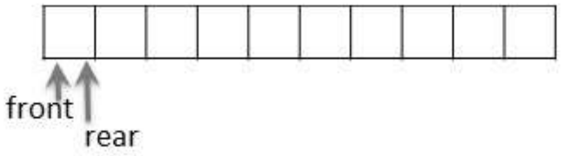
2) Dequeue() – Remove item from the queue from the FRONT.



3) **isFull()** – Check if queue is full or not.



4) **isEmpty()** – Check if queue empty or not.



5) **count()** – Get number of items in the queue.

Exercise

Select the correct alternative and rewrite the following statements

1. Finding location of given element is called as ___________.
   (1) Traversing (2) Insertion (3) Searching (4) None of the above
1. (3) Searching

2. Data items that are divided into subitems are called as _______
   (1) Group items (2) Elementary items
   (3) Nodes (4) Arrays
2. (1) Group items

3. In LINKED LIST, Link field contains __________.
   (1) Value of next node (2) Address of next node
   (3) Value of previous node (4) None of these
3. (2) Address of next node

4. A record is a collection of _________.
   (1) Files (2) Arrays (3) Fields (4) Maps
4. (3) Fields

5. The time required to execute bubble sort algorithm having 'n' input items is directly proportional to ________.
   (1) n^2 (2) n (3) $Log_2 n$ (4) $Log_e n^2$
5. (1) n^2

6. Maximum number of nodes of symmetric binary tree with depth n are _________.
   (1) 2^n (2) $Log_2 n$ (3) n^2 (4) $2^n - 1$
6. (4) $2^n - 1$

7. Maximum number of nodes of symmetric binary tree with depth 5 are _______.
   (1) 5 (2) 25 (3) 31 (4) 32
7. (3) 31

8. Accessing each element in an array only once is called ________.
(1) Searching (2) Inserting (3) Deleting (4) Traversing

8. (4) Traversing

9. The elements of record are ________.
(1) Homogeneous (2) Similar
(3) Non–homogeneous (4) Identical

9. (3) Non–homogeneous

10. The most efficient search algorithm is ________.
(1) Binary search (2) Reverse search
(3) Linear search (4) Pointer search

10. (1) Binary search

11. The number of comparisons required for bubble sorting of an array of n elements is __________.
(1) $n(n-1)/2$ (2) $n/2$ (3) $\log_2 n$ (4) $\log_{10} n$

11. (1) $n(n-1)/2$

12. Finding the location of record with a give key value is known as ________.
(1) Traversing (2) Searching (3) Sorting (4) Inserting

12. (2) Searching

13. Maximum number of nodes in a symmetric binary tree with depth four are _______.
(1) 4 (2) 15 (3) 16 (4) 5

13. (2) 15

14. In _______ data structure, an element may be inserted or deleted only at one end called Top.
(1) Queue (2) Array (3) Stack (4) Tree

14. (3) Stack

15. Maximum number of nodes of symmetric binary tree with depth of 6 is _____.
(1) 64 (2) 6 (3) 63 (4) 36

15. (3) 63

16. ______ is the only non–linear data structure from the following list.
(1) Array (2) Stack (3) Tree (4) Linked List

16. (3) Tree

17. ______ is the operation of rearranging the elements of an array either in increasing and decreasing order.
(1) Sorting (2) Searching (3) DMS (4) DBMS

17. (1) Sorting

18. The complete binary tree (Tn) has n = 15 nodes then its depth (dn) is _________.
(1) 2 (2) 3 (3) 4 (4) 5

18. (4) 5

19. Maximum number of nodes of symmetric binary tree with depth of 7 is ________.
(1) 125 (2) 127 (3) 128 (4) 124
19. (2) 127

20. Elements of Array are always ________.
(1) Homogenous (2) Hetrogenous
(3) Non–homogenous (4) None of these
20. (1) Homogenous

21. Record contains ______ Data.
(1) Homogeneous (2) Non–homogeneous
(3) Same (4) None of these
21. (2) Non–homogeneous

22. Sorted List is essential requirement for _____ process of an array.
(1) Linear Search (2) Binary Search (3) Traversing (4) Insertion
22. (2) Binary Search

23. Maximum number of nodes of symmetric binary tree with depth 6 are ______.
(1) 31 (2) 127 (3) 63 (4) 64
23. (3) 63

24. Tree is _________ Data Structure.
(1) Linear (2) Non–linear (3) Homogeneous (4) Non–homogeneous
24. (4) Non–homogeneous

25. The elements of the binary tree are ______.
(1) Homogenous (2) Non–homogeneous
(3) Similar (4) Identical
25. (2) Non–homogeneous

26. Complete Binary Tree (T_n) has n = 31 nodes, then its depth is _________.
(1) 2 (2) 3 (3) 4 (4) 5
26. (4) 5

27. Most efficient search algorithm is __________.
(1) Binary (2) Reverse (3) Linear (4) Pointer
27. (1) Binary

28. Finding location of given element in array is called _______.
(1) Sorting (2) Searching (3) Traversing (4) Merging
28. (2) Searching

29. _______ data structure does not require contiguous memory allocation.
(1) Array (2) String (3) Pointer array (4) Linked list
29. (4) Linked list

30. Tree is a _____ collection of Nodes.
(1) Hierarchical (2) Linear (3) Relational (4) Graphical
30. (1) Hierarchical

31. If a complete binary tree (Tn) has n = 1000 nodes, then its depth (Dn) is _________.
(1) 10 (2) 20 (3) 50 (4) 100
31. (1) 10

32. __________ is the only non–linear data structure from the following list.
(1) Array (2) Stack (3) Tree (4) Linked List
32. (3) Tree

33. If lower bound = 0 and upper bound = 15, then midterm is ______ in binary search method.
(1) 6 (2) 7 (3) 8 (4) 9
33. (3) 8

34. _______ is very useful in situation when data is to be stored and retrieved in reverse order.
(1) Stack (2) Queue (3) Linked List (4) Tree
34. (1) Stack

35. Record contains _________ data.
(1) Homogenous (2) Non–homogenous
(3) Same (4) None of these
35. (2) Non–homogenous

36. ________ is collection of fields.
(1) File (2) Record (3) Array (4) Queue
36. (2) Record

37. In binary search method, the condition is data must be ______.
(1) Sorted (2) Unsorted (3) Random (4) Discrete
37. (1) Sorted

❑ ❑ ❑ ❑ ❑ ❑