SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – I-Data Structures – SCSA1203

I. ALGORITHM

An algorithm is a well-defined computational procedure having well defined steps for solving a particular problem. Algorithm is finite set oflogic or instructions, written in order for accomplishing the certain predefined task.

Each algorithm must have:

- **Specification:** Description of the computational procedure.
- **Pre-conditions:** The condition(s) on input.
- **Body of the Algorithm:** A sequence of clear and unambiguous instructions.
- **Post-conditions:** The condition(s) on output.

Characteristics of an Algorithm

An algorithm must follow the mentioned below characteristics:

- **Input:** An algorithm must have 0 or well-defined inputs.
- **Output:** An algorithm must have 1 or well-defined outputs, and should match with the desired output.
- **Feasibility:** An algorithm must be terminated after the finite number of steps.
- **Independent:** An algorithm must have step-by-step directions which is independent of any programming code.
- **Unambiguous:** An algorithm must be unambiguous and clear. Each of their steps and input/outputs must be clear and lead to only one meaning.

  The performance of algorithm is measured on the basis of following properties:
- **Time complexity:** It is a way of representing the amount of time needed by a program to run to the completion.
- **Space complexity:** It is the amount of memory space required byan algorithm, during a course of its execution. Space complexity is required in situations when limited memory is available and for the multi user system.

2. INTRODUCTION TO DATA STRUCTURES

2.1 Data

In the modern context, data stands for both singular and plural. Data means a value or set of

values. Data can be defined as an elementaryvalue or the collection of values, for example, student's name and its idare the data about the student.

2.2 Structure

A building is a structure. A bridge is structure. In general, a structureis made up of components. It has a form or shape. It is made up of parts.A structure is an arrangement of and relations between parts or elements.

2.3 Data Structures

A data structure is an arrangement of data elements. Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspectof Computer Science i.e. Operating System, Compiler Design, Artificial Intelligence, Graphics and many more.

2.3.1 Needs

As applications are getting complex and amount of data is increasing day by day, the following issues might be araised:

Processor speed: To handle very large amount of data, high speed processing is required, but as the data is growing day by day tothe billions of files per entity, processor may fail to deal with that much amount of data.

Data Search: Consider an inventory size of 106 items in a store;if our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

Multiple requests: If thousands of users are searching the data simultaneously in a web server, it fails to process the requests.

In order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items arenot required to be searched and required data can be searched instantly.

Data structures are important for the following reasons

1. Data structures are used in almost every program or software system.
2. Specific data structures are essential ingredients of many efficient algorithms, and make

possible the management of huge amounts of data, such as large integrated collection of databases.

3. Some programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design.

Advantages of Data Structures

Efficiency: Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a particular record. element. Hence, using array may not be very efficient here.

Reusability: Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place.

Abstraction: Data structure is specified by the ADT which provides alevel of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

3.Classification of Data Structure

The data structure is classified into two different types, primitive and non-primitive data structures is shown in Fig.1.

Primitive Data Structures

Simple data structure can be constructed with the help of primitivedata structure. A primitive data structure used to represent the standard data types of any one of the computer languages. Variables, pointers, structures, unions, etc. are examples of primitive data structures.

Non Primitive Data Structures

Non Primitive Data Structures are classified as linear or non-linear. Arrays, linked lists, queues and stacks are linear data structures. Trees and Graphs are non-linear data structures. Except arrays, all other data structures have many variations. Non Primitive data structure can be constructed with the help of any one of the primitive data structure andit is having a specific functionality. It can be designed by user.

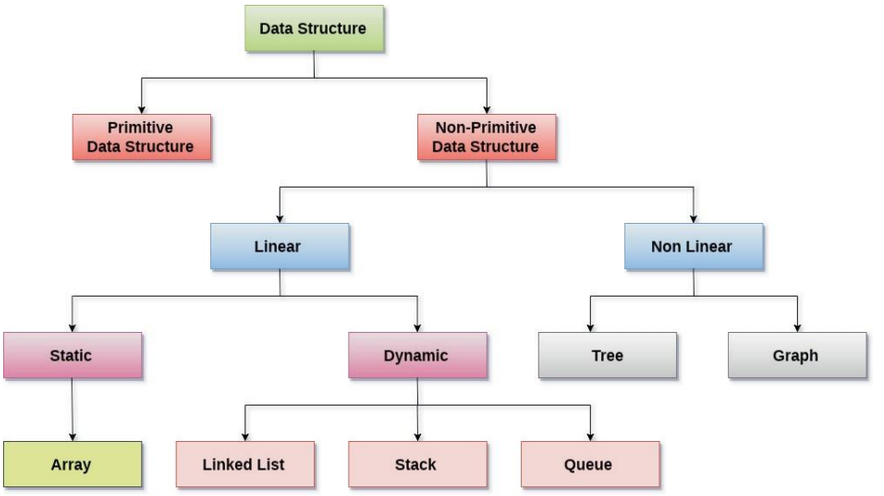


Fig. 1 Classification of Data Structure

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. In linear data structure the elements are stored in sequential order.

Types of Linear Data Structures are given below:

Arrays: An array is a collection of similar type of data items stored in consecutive memory location and is referred by common name; 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 onecarries a different index number known as subscript.

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. It is a collection of data of same data type but the data items need not be stored in consecutive.

Stack: Stack is a linear list in which insertion and deletions are allowedonly at one end, called top. It is a Last-In-First-Out linear data structure.

A stack is an abstract data type (ADT), can be implemented in mostof the programming languages. It is named as stack because it behaveslike 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 atone end called rear and deleted only at the other end called front. It isa First-In-First-Out Linear data structure.

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

Operations applied on Linear Data Structure

The following list of operations applied on linear data structures

- Add an element
- Delete an element
- Traverse
- Sort the list of elements
- Search for a data element

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. Elements are stored based on the hierarchical relationship among the data. The following are some of the Non-Linear data structures.

Trees: Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottom most nodes are called leaf node while the top most node is called root node. Each node contains pointers to point adjacent nodes. Each node in the tree can have more than one child except the leaf nodes whereas each node can have at most one parent except the root node.

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 applied on non-linear data structures

The following list of operations applied on non-linear data structures.

1. Add elements
2. Delete elements
3. Display the elements

4. Sort the list of elements

5. Search for a data element

4. ABSTRACT DATA TYPES

Abstract Data Type (ADT) is a type or class for objects whose behaviour is defined by a set of value and a set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. The process of providing only the essentials and hiding the details is known as abstraction is shown in fig,2.

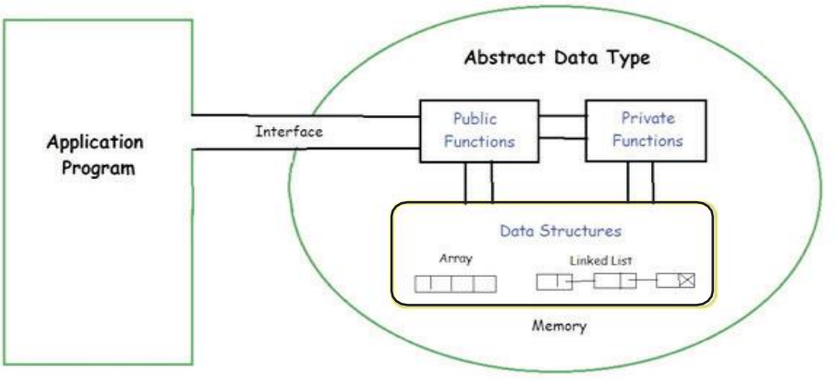


Fig. 2 Abstract Data Type

4.1 List ADT

- The data is generally stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list is shown in fig.3.

- The data node contains the pointer to a data structure and a self-referential pointer which points to the next node in the list.

```
//List ADT Type Definitions
    typedef struct node{
 void *DataPtr;
struct node *link;
    } Node;
typedef struct{
   int count;
  Node *pos;
 Node *head;
  Node *rear;
      int (*compare) (void *argument1, void *argument2);
                              } LIST;
```

Fig.3 List ADT

✱ A list contains elements of the same type arranged in sequential order and following operations can be performed on the list.

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from anon-empty list.
- removeAt() – Remove the element at a specified location from anon-empty list.
- replace() – Replace an element at any position by anotherelement.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise returnfalse.
- isFull() – Return true if the list is full, otherwise return false.

The List ADT Functions is shown in Fig.4.

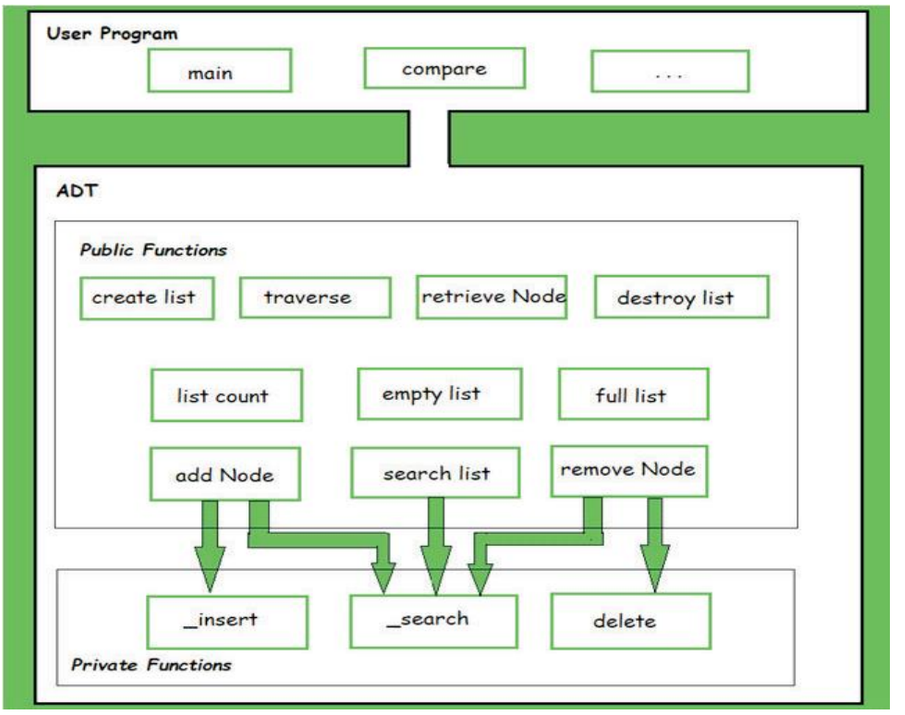


Fig. 4 List ADT Functions

4.2 Stack ADT

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- The program allocates memory for the data and address is passed to the stack ADT is shown in Fig.5.
- The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.
- The stack head structure also contains a pointer to top and countof number of entries currently in stack.
- A Stack contains elements of the same type arranged in sequential order.

```
//Stack ADT Type Definitions
typedefstruct node{
void *DataPtr;
struct node *link;
} StackNode;
typedefstruct{
int count;
StackNode *top;
} STACK;
```

Fig. 5 Stack ADT

All operations take place at a single end that is top of the stackand following operations can be performed:

- push() – Insert an element at one end of the stack called top.
- pop() – Remove and return the element at the top of the stack,if it is not empty.
- peek() – Return the element at the top of the stack withoutremoving it, if the stack is not empty.
- size() – Return the number of elements in the stack.
- isEmpty() – Return true if the stack is empty, otherwise returnfalse.
- isFull() – Return true if the stack is full, otherwise return false.

4.3 Queue ADT

✱ The queue Abstract Data Type (ADT) follows the basic design of the stack abstract data type is shown in fig.6.

```
//Queue ADT Type Definitions
typedefstructnode {
void*DataPtr;
 structnode *next;
} QueueNode;
typedefstruct {
QueueNode *front;
QueueNode *rear;
intcount;
} QUEUE;
```

Fig 6. Queue ADT

✱ A Queue contains elements of the same type arranged in sequential order. Operations take place at both ends, insertion is done at theend and deletion is done at the front. Following operations can be performed:

- enqueue() – Insert an element at the end of the queue.
- dequeue() – Remove and return the first element of the queue,if the queue is not empty.
- peek() – Return the element of the queue without removing it,if the queue is not empty.
- size() – Return the number of elements in the queue.
- isEmpty() – Return true if the queue is empty, otherwise returnfalse.
- isFull() – Return true if the queue is full, otherwise return false.

6.ARRAYS

The number of data items chunked together into a unit is known as data structure. When the data structure consists simply a sequence of data items, the data structure of this simple variety is referred as an array.

Definition: Array is a collection of homogenous (same data type) dataelements that are stored in contiguous memory locations.

Array Syntax

Syntax to declare an array:

✱ dataType [ ] arrayName;

arrayName= new dataType[n]; //keyword new performs dynamicmemory location

(or)

✱ dataType [ ] arrayName = new dataType[n];

Example:

int [ ] x; x=new int [10];(or)

int [] x=new int [10];

Array Initialization

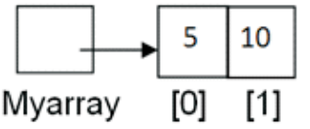
The values of an array can be initialized as follows,

Example 1:

int [] j=new int [1]; j[0] =10;**(or)**int [] j= {25};

Example 2:

int [] myarray= {5, 10};



5.1 Characteristics of an Array

The following are the characteristics of an array data structure.

(i) Array stores elements of same data type.

(ii) The elements of an array are stored in subsequent memorylocations.

(iii) The name of array represents the starting address of the elements.

(iv) When data objects are stored in array, individual objects areselected by an index.

(v) By default an array index starts from 0 and ends with (n-1). Indexis also referred as subscripts.

(vi) The size of the array is mentioned at the time of declaring array.

(vii) While declaring 2D array, number of columns should be specifiedwhereas for number of rows there is no such rule.

(viii) Size of array can't be changed at run time.

5.2 Array Types

1.One-Dimensional Array or Linear arrays

2. Multi-Dimensional Array

3.Two dimensional (2D) Arrays or Matrix Arrays

5.2.1 One-Dimensional Array

In one dimensional array each element is represented by a single subscript. The elements are stored in consecutive memory locations. E.g.A [1], A [2], ..., A [N].

5.2.2 Two dimensional (2-D) arrays or Matrix Arrays

In two dimensional arrays each element is represented by two subscripts. Thus a two dimensional m x n array has m rows and n columns and contains m * n elements. It is also called matrix array because in this case, the elements form a matrix. For example A [4] [3]has 4 rows and 3 columns and 4*3 = 12 elements.

```
int [] [] A = new int [4] [3];
```

5.2.3 Multi dimensional arrays:

In it each element is represented by three subscripts. Thus a three dimensional m x n x l array contains m * n * l elements. For exampleA [2] [4] [3] has 2 * 4 * 3 = 24 elements.

6.STORAGE REPRESENTATION

An array is a set of homogeneous elements. Every element is referredby an index. Memory storage locations of the elements are not arrangedas a rectangular array with rows and columns. Instead, they are arranged in a linear sequence beginning with location 1, 2, 3 and so on. The elements are stored either column-by-column or row-by-row. The first one is called column-major order and later is referred as row-major order.

6.1 Row Major Order

The table 1 shows the linear arrangement of data in row major order.

Example

- Rows : 3
- Columns : 4

Data (A):

1	2	3	4
5	6	7	8
9	10	11	12

Table 1 Linear Arrangement of Array A in Row Major Order

The formula is:

Location (A [j,k]) = Base Address of (A) + w [(N * (j-1)) + (k-1)]

Location (A [j, k]): Location of j^{th} row and k^{th} column

Linear Arrangement of Array A												
Row	**1**				**2**				**3**			
Index	**(1,1)**	**(1,2)**	**(1,3)**	**(1,4)**	**(2,1)**	**(2,2)**	**(2,3)**	**(2,4)**	**(3,1)**	**(3,2)**	**(3,3)**	**(3,4)**
Memory	**100**	**102**	**104**	**106**	**108**	**110**	**112**	**114**	**116**	**118**	**120**	**122**
Data	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**

Base (A) : Base Address of the Array A

w : Bytes required to represent single element of the Array A

N : Number of columns in the Array

j : Row position of the element

k : Column position of the element

Example

Suppose to find the address of (3,2) then

Base (A) = 100

w = 2 Bytes (integer type)

N = 4

j = 3

k = 2

Location (A [3, 2]) = 100 + 2 [(4 * (3-1) + (2-1)]

= 118

6.2 Column Major Order

The table 2 shows the linear arrangement of data in column major order.

- Rows : 3
- Columns : 4

Linear Arrangement of Array A												
Column	**1**			**2**			**3**			**4**		
Index	**(1,1)**	**(2,1)**	**(3,1)**	**(1,2)**	**(2,2)**	**(3,2)**	**(1,3)**	**(2,3)**	**(3,3)**	**(1,4)**	**(2,4)**	**(3,4)**
Memory	**100**	**102**	**104**	**106**	**108**	**110**	**112**	**114**	**116**	**118**	**120**	**122**
Data	**1**	**5**	**9**	**2**	**6**	**10**	**3**	**7**	**11**	**4**	**8**	**12**

Table 2 Linear Arrangement of Array A in Column Major Order

The formula for column major order is:

Location (A [j, k]) = Base Address of (A) + w [(M * (k-1)) + (j-1)]

Location (A [j, k]): Location of j^{th} row and k^{th} column

Base (A)	: Base Address of the Array A
w	: Bytes required to represent single element of the Array A
M	: Number of rows in the Array
j	: Row position of the element
k	: Column position of the element

Example

Base (A) = 100
w = 2 Bytes (integer type)
M = 3
j = 3
k = 2
Location (A [3, 2]) = 100 + 2 [(3 * (2-1) + (3-1)]
= 110

7. Array Order Reversal

Given an array (or string), the task is to reverse the array/string.

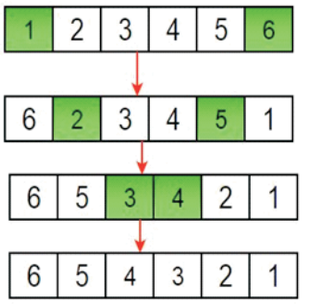
Examples:

Input : arr[] = {1, 2, 3} Output : arr[] = {3, 2, 1}

Input : arr[] = {4, 5, 1, 2} Output : arr[] = {2, 1, 5, 4}

Algorithm

1) Initialize start and end indexes as start = 0, end = n-1.
2) In a loop, swap arr[start] with arr[end] and change start and end as follows : start = start +1, end = end – 1



8.Array Counting

1. Create a function arraycounting(array, size)

2. Find largest element in array and store it in max

3. Initialize count array with all zeros

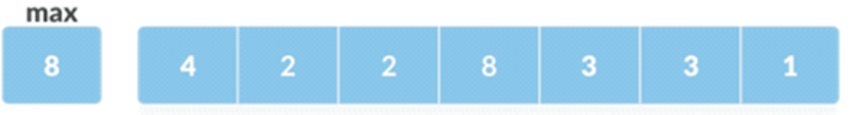
4. for j = 0 to size

5. Find the total count of each unique element and

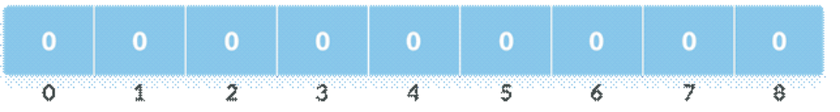
6. Store the count at jth index in count array

Example A={4,2,2,8,3,3,1}

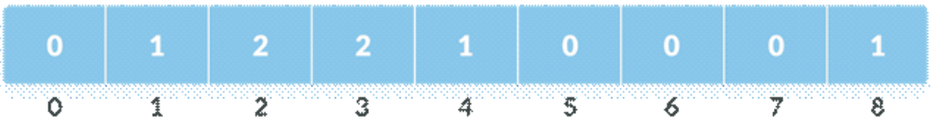
1. Find out the maximum element (let it be max) from the given array.



2. Initialize an array of length max+1 with all elements 0. This arrayis used for storing the count of the elements in the array.



3. Store the count of each element at their respective index in count array. For example: If the count of element “4” occurs 2 times then2 is stored in the 4^{th} position in the count array. If element “5” isnot present in the array, then 0 is stored in 5^{th}



8. Finding the maximum Number in a Set

Algorithm

- Read the array elements
- Initialize first element of the array as max.
- Traverse array elements from second and compare every elementwith current max
- Find the largest element in the array and assign it as max
- Print the largest element.

Example: A={56,78,34,23,70}

Step 1: Initialize max=0 , n=len(A)

Step 2: Repeat step 3 until n

Step 3: 56>0 yes, Assign max=56

78>56, yes Assign max=78

34>78, No

23>78, No

70>78, No

Step 4: print Max Output:78

9. RECURSION

Recursion is a programming technique using function or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one calledto the first.

A simple recursive algorithm:

- Solves the base cases directly.
- Recurs with a simpler sub problem or sub problems.
- May do some extra work to convert the solution to the simpler subproblem into a solution to the given problem.

Some Example Algorithms

1. Factorial
2. All permutations
3. Tree traversal

4. Binary Search
5. Quick Sort
6. Towers of Hanoi

9.1 Design Methodology and Implementation of Recursion

* The recursive solution for a problem involves a two-way journey:
* First we decompose the problem from the top to the bottom
* Then we solve the problem from the bottom to the top.

Steps for Designing Recursive Algorithms

* Each call of a recursive algorithm either solves one part of theproblem or it reduces the size of the problem.
* The general part of the solution is the recursive call. At eachrecursive call, the size of the problem is reduced.
* The statement that ôsolvesö the problem is known as the base case.
* Every recursive algorithm must have a base case.
* The rest of the algorithm is known as the general case. The generalcase contains the logic needed to reduce the size of the problem.
* Once the base case has been reached, the solution begins.We now know one part of the answer and can return that part tothe next, more general statement.
* This allows us to solve the next general case.
* As we solve each general case in turn, we are able to solve the next-higher general case until we finally solve the most general case, the original problem.

Rules for Designing a Recursive Algorithm

1. First, determine the base case.
2. Then determine the general case.
3. Combine the base case and the general cases into an algorithm.
4. Each recursive call must reduce the size of the problem and moveit toward the base case.
5. The base case, when reached, must terminate without a call to therecursive algorithm; that is, it must execute a return.

9.2 Broad Categories of Recursion

Recursion is a technique that is useful for defining relationships, andfor designing algorithms that implement those relationships. It is a natural way to express many algorithms in an optimized way. Recursive function is defined in terms of itself.

- Linear Recursion
- Binary Recursion

Linear Recursion:

Linear recursion is by far the most common form of recursion. In this style of recursion, the function calls itself repeatedly until it hits the termination condition (Base condition).

Binary Recursion

Binary recursion is another popular and powerful method. This formof recursion has the potential for calling itself twice instead of once asbefore. This is pretty useful in scenarios such as binary tree traversal,generating a Fibonacci sequence, etc.

Tail Recursion

A function call is said to be tail recursive if there is nothing to doafter the function returns except return its value. Since the current recursive instance is done executing at that point, saving its stack frameis a waste

For example the following C function print () is tail recursive.

```
// An example of tail recursive functionPrint (n) {
If (n < 0) return;

Display n;

Print (n-1);

}
```

10. FIBONACCI SERIES

Fibonacci Series generates subsequent number by adding two previous numbers. Fibonacci series starts from two numbers F0 & F1. The initial values of F0 & F1 can be

- 0 and 1

- 1 and 1 respectively.

The Fibonacci series looks like
F8 = 0 1 1 2 3 5 8 13

The algorithm for generating Fibonacci series can be drafted in 2 ways

1. Fibonacci Iterative Algorithm.

2. Fibonacci Recursive Algorithm.

Fibonacci RecursiveAlgorithm

Algorithm Fibo (n)
If n = 0
Return 0Else If n = 1
Return 1
Else
Fibo (n) = Fibo (n-1) + Fibo (n-2)Return Fibo (n)

11.FACTORIAL

The factorial of a positive number is the product of the integral values from 1 to the number: Factorial of the given number can be calculated as:

Algorithm

RecursiveFactorial (n)if (N equals 0)
Return 1
else
Return (n*recursiveFactorial (n-1))
end if
end recursiveFactorial

Calling a Recursive Factorial Algorithm with n=3

Fig.6 shows the steps for calculating the factorial using Recursion for n=3.

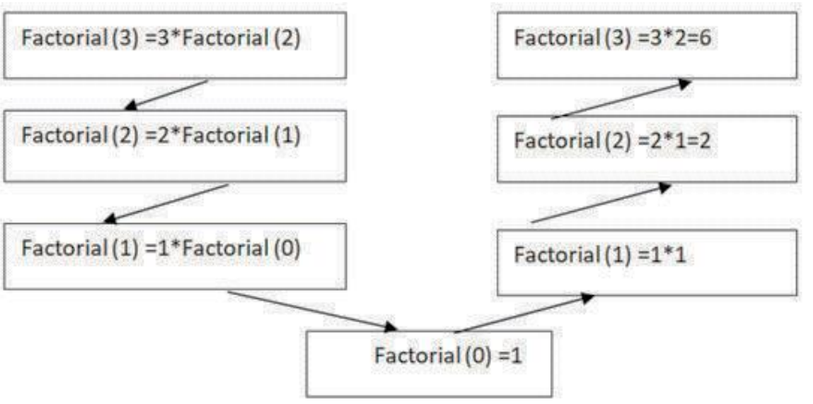


Fig 6. Factorial using Recursion Steps

Output: 6

12. TOWERS OF HANOI

Tower of Hanoi is a mathematical puzzle which consists of three tower (pegs) and more than one ring; as depicted in Fig.7.

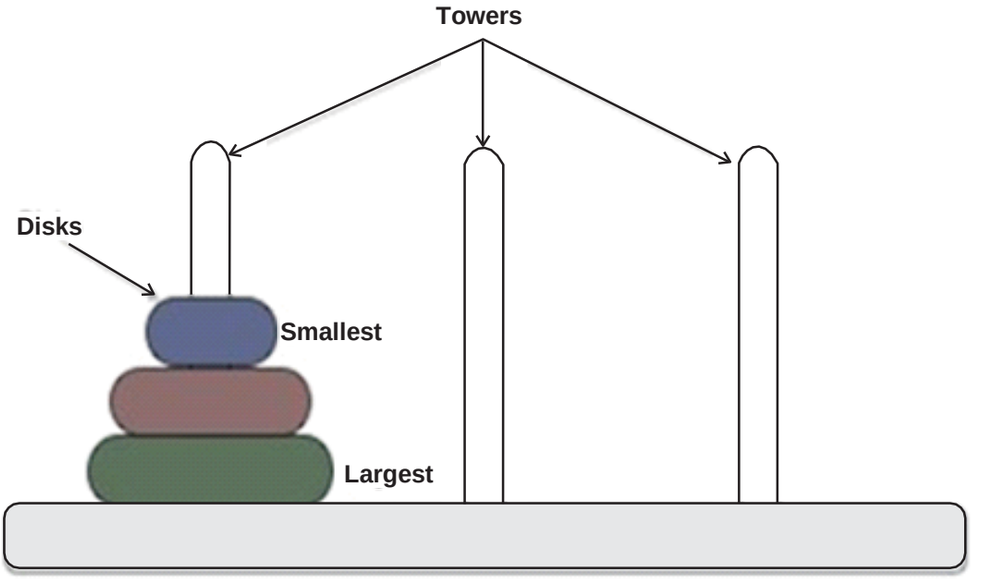


Fig.7 Tower of Hanoi

These rings are of different sizes and stacked upon in ascending order i.e. the smaller

one sits over the larger one. There are other variationsof puzzle where the number of disks increases, but the tower count remains the same.

7.1 Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. The below mentioned are few rules which are to be followed for Tower of Hanoi

- Only one disk can be moved among the towers at any given time.
- Only the “top” disk can be removed.
- No large disk can sit over a small disk.

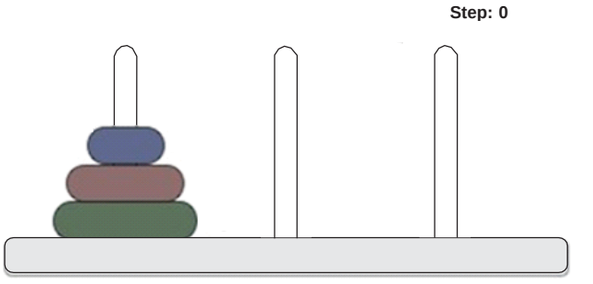
Steps for solving the Towers of Hanoi problem

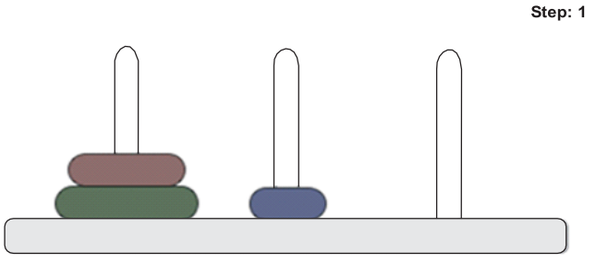
The following steps are to be followed.

Step 1: Move n-1 disks from source to aux.

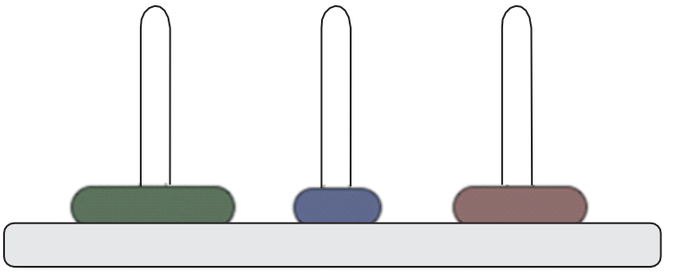
Step 2: Move nth disk from source to destinationStep 3: Move n-1 disks from aux to destination.

Following Fig.8 illustrates the step by step movement of the disks to implement Tower of Hanoi.

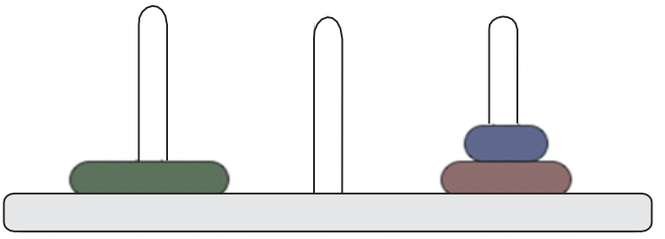




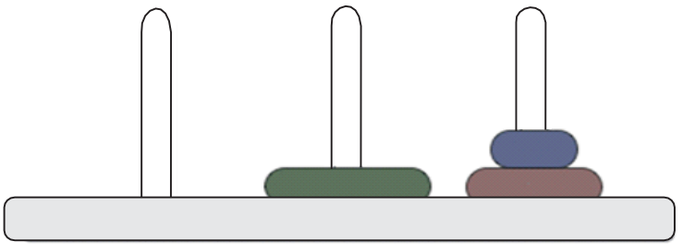
Step: 2



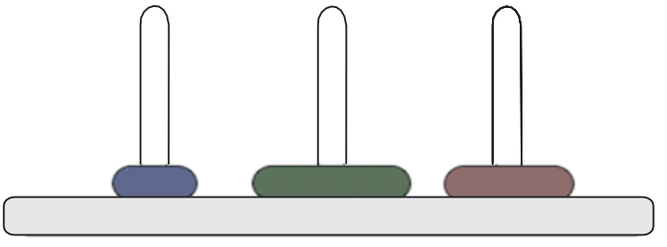
Step: 3



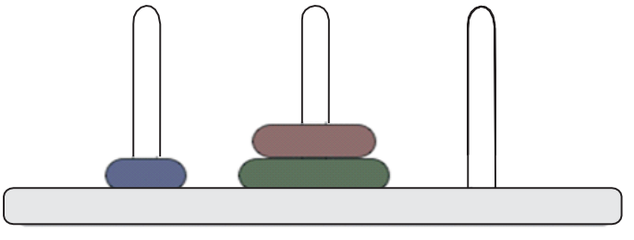
Step: 4



Step: 5



Step: 6



Done!

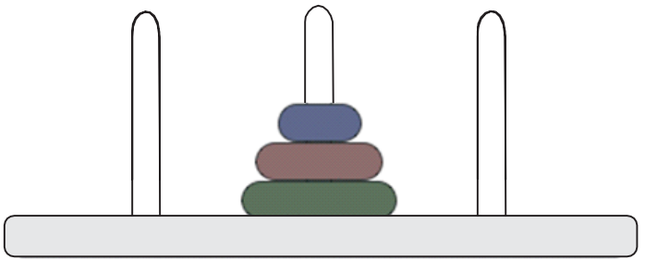


Fig.8 Tower of Hanoi

A recursive algorithm for Tower of Hanoi can be driven as followsSTART

Procedure **Hanoi** (disk, source, dest, aux)

IF disk = 0, THEN

Move disk from source to destELSE

Hanoi (disk-1, source, aux, dest) //Step1Move disk from source to dest //Step2 Hanoi (disk-1, aux, dest, source) //Step3 ENDIF

END