Data Structure and Algorithms.

Th → 80 Int → 20
Pr → 50

1. Concept of data structure – 2 hrs. –(4)
2. The stack and Queue – 6 hrs. – (10)
3. List – 3 hrs – (6)
4. Linked Lists → 5 hrs – (10)
5. Recursion → 4 hrs. – (8)
6. Trees → 7 hrs. – (12)
7. Sorting → 5 hrs. – (8)
8. Searching → 5 hrs. – (8)
9. Growth Functions → 2 hrs. – (4)
10. Graphs → 6 hrs. – (10)

Data types:

A data type is a collection of values and set of operations on those values.

Most programming languages have data types such as integers, floating point numbers, characters etc. All these data types define set of values and operations on those values.

Eg: In 'c' programming 'int' is a data type which takes integer values and operations like addition, multiplication, division etc. are performed on those values.

Data Structure. (Used on Operating System, compiler design, AI, Search engines, software dev., graphics etc.)

→ Data structure is a technique of organizing and storing data elements (different types of data items) in computer memory.

It is considered as not only the storing of data elements but also the maintaining of the logical relationship existing between individual data elements.

→ It is also an mathematical or logical model of particular organization of data items.

Data structure mainly specifies:

→ Organization of datas
→ Accessing methods
→ Degree of associativity
→ Processing alternatives for ~~pro~~ information.
→ Building blocks of programs.

To develop a program of an algorithm, we should or
select an appropriate data structure for that algorithm do

Algorithm + Data Structure = Programs

Data structure classification.

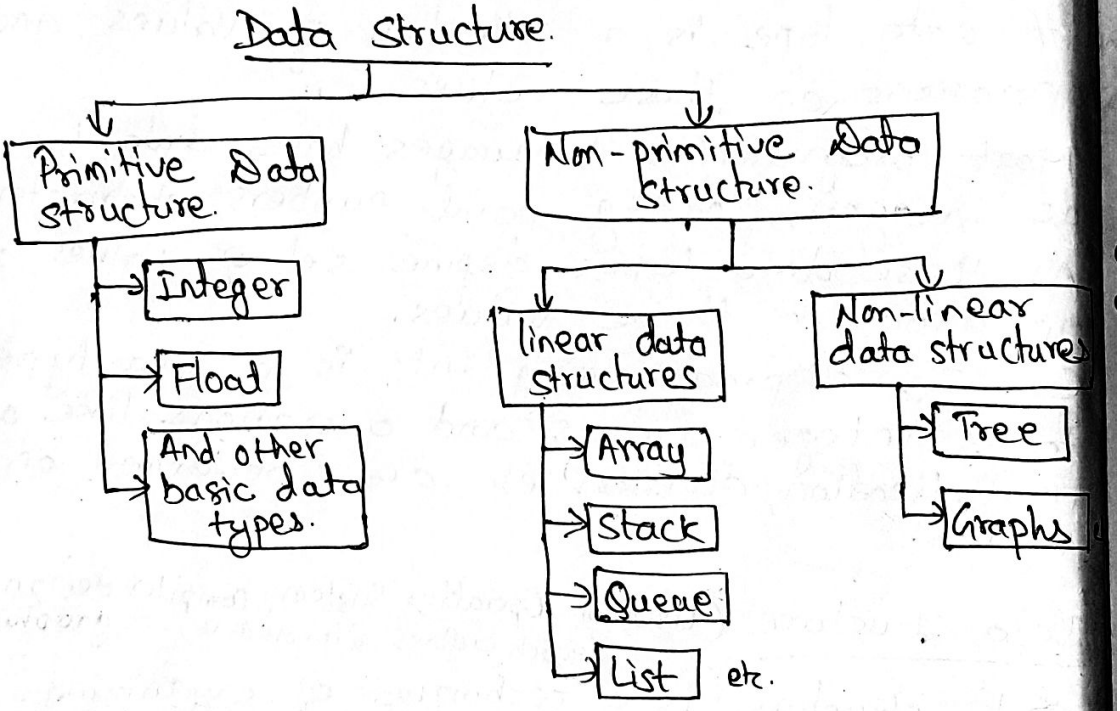


fig: classification of data structures.

Primitive data structures: (Basic data structures)

Those data structures that are directly operated upon by the machine instructions are called primitive data structures.

Eg: Integer
float
character
pointer etc.

Normally, primitive data structures have different represen
on ~~differ~~ different computers.

Non-primitive Data Structure.

→ Complex data structures
→ Derived from primitive data structures.

The non-primitive data structure is responsible for organizing the group of homogeneous and heterogeneous data elements.

Eg: Arrays
Lists.
files. etc.

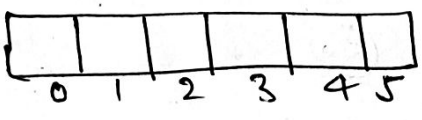
There are two types of data structures
① Linear Data Structures.
② Non-Linear data structures.

Linear data structures:

A data structure is called linear if all of its elements are arranged in the linear order. Each data elements of linear data structure has successors and predecessor except the first and last elements.

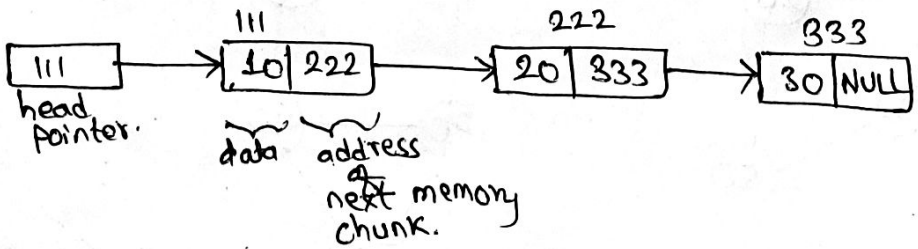
Eg: Array, stack, Lists, Queue.

Array: Group of homogeneous elements in a single unit.

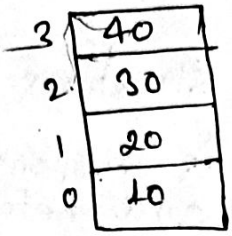


Lists: list is the fundamental linked data structure where elements are stored in distinct chunks of memory which are linked together with pointers.

Lists don't need resizing.

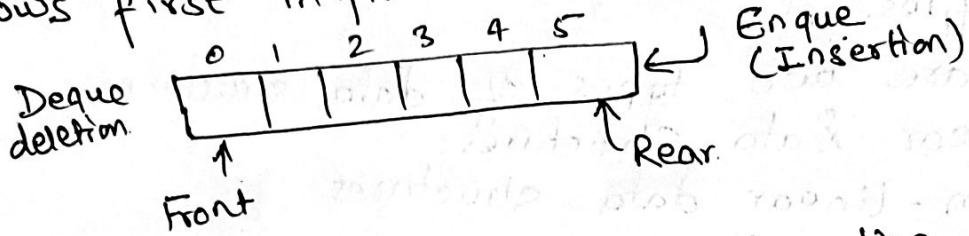


Stack. (LIFO)



→ stack is a linear data structure in which an element may be inserted or deleted only at one end.

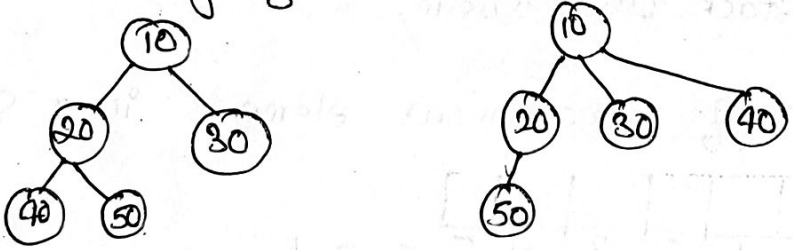
Queue: Queue is a linear data structure which follows first in first out mechanisms.



Note: Rear is incremented while inserting an element into the queue and front is incremented while deleting the element from the queue.

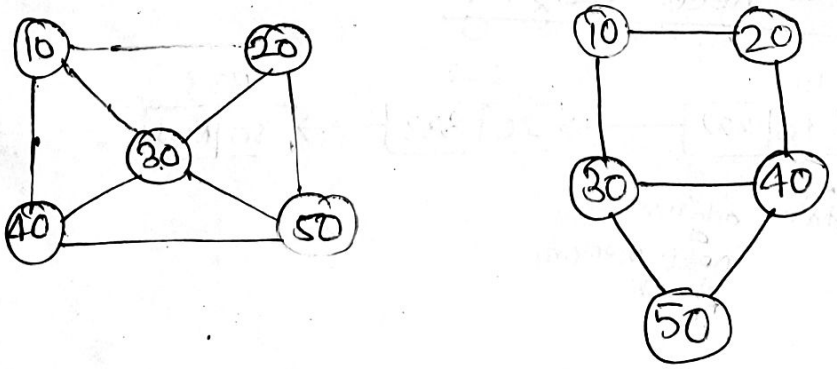
Tree

A tree is a non-linear data structure that models a hierarchical organization.



Graphs

Graph is a nonlinear datastructure consisting a set of points known as nodes (or vertices) and set of links known as edges (or Arcs) which connects the vertices.



Data Structure Operations.

1. Insertion
2. Deletion
3. Traversing
4. Searching
5. Sorting
6. Merging.

Abstract Data Type (ADT)

→ An abstract data type (ADT) consists of a data type together with a set of operations which define how the type may be manipulated. ~~without~~
It exists conceptually and concentrate on the mathematical properties of the data type ignoring implementation constraints and details.

→ Most commenly used data types are stacks and queues.

It consists of two parts:
1. Declaration of data.
2. Declaration of operations.

Lets take example of stack:

Declaration of data: It works on the basis of LIFO mechanisms.

Declaration of operations :- push(), pop() etc.

Algorithm

→ An algorithm is any well-defined computational procedure that take some value or set of value as input, and produces an output.

→ An algorithm is finite sequence of instructions for solving a stated problem. Eg: Sorting algorithm, searching algorithm.

Characteristics of an algorithm.

① Input
② Output
③ Finiteness
④ Unambiguous
⑤ Effectiveness.

Q. Algorithm to find largest number among given three numbers.

```
step1: start
step2: Declare variables a,b,c
step3: Read values a,b,c
step4: if a>b
         if a>c
           print a is the greatest number
         else
           print c is the greatest number
       else
         if b>c
           print b is the greatest number
         else
           print c is the greatest number
step 5: stop.
```

Algorithm analysis (Efficiency of algorithm).

Once an algorithm is given for a problem and determined to be correct, the next step is to determine the amount of resources, such as time and space that the algorithm will require. This step is called algorithm analysis.

When we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm (Time & Space). But when we calculate complexity of an algorithm it does not provide exact amount of resource required. So instead of taking exact amount of resource, we represent that complexity in a general form which produces the basic nature of that algorithm. We use that general form for analysis process.

Efficiency of algorithm is measured with the help of asymptotic notations.

Asymptotic notations: The study of change in performance of the algorithm with the change in the order of the input size is called asymptotic analysis.

Big oh (O) notation.

When we have only asymptotic upper bound then we use Big oh (O) notation.

It is given by:

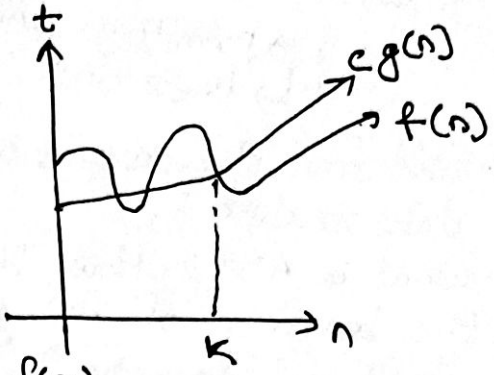
$f(n) = O\,g(n)$ such that

$f(n) \leq c\,g(n)$

$c > 0$

$n \geq k$

$k \geq 0$



Q. Find big oh of given function $f(n) = 3n^2+4n+7$

Soln

$f(n) = 3n^2+4n+7 \leq 3n^2+4n^2+7n^2 \leq 14n^2$

$f(n) = 14n^2$

where, $c = 14$

$g(n) = n^2$

Thus, $f(n) = Og(n) = O(n^2)$

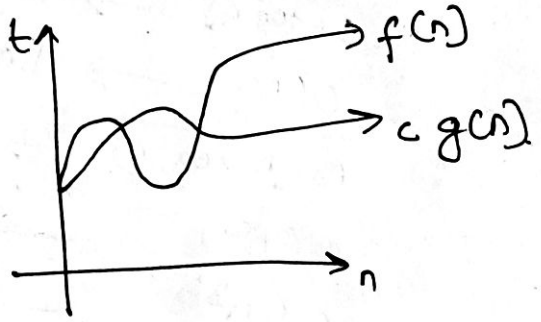
Big Omega (Ω) notation.

Big omega notation gives asymptotic lower band.

It is given by,

$f(n) = \Omega\, g(n)$

$f(n) \geq c \cdot g(n)$



Q. Find big Ω of given function $f(n) = 3n^2+4n+7$.

Soln

$f(n) = 3n^2+4n+7 \geq 3n^2$

$f(n) \geq = 3n^2$

where, $c = 3$ and $g(n) = n^2$

Thus,

$$f(n) = \Omega(g(n)) = \Omega(n^2)$$

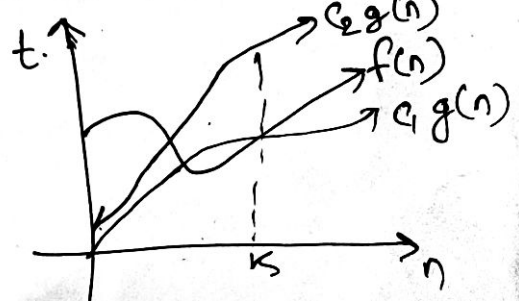
tightest bound

Big Theta (Θ) notation

When we need asymptotically tight bound then we use this notation.

It is given by:

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$



(little o and little ω) Read yourself

Q. what do you understand by data structure? What are its importance? List out its application areas.

Q. Differenciate between:
- i → primitive and non-premitive data structure.
- ii → linear and non-linear data structure.

Q. What are the various operations that can be performed on data structure?

Q. What is ADT? How it differs from data type? What are the benefits of using ADT?

Q. What is algorithm? What are the characteristics of good algorithm?

Q. what are asymptotic notations? Explain its types.

Q. ADT supports abstraction. Explain.

Q. How can you say stack is an ADT? Explain.

Q. Point out limitations and advantages of different asymptotic notations.

$O(1)$	→ constant time	Feasible
$O(\log n)$	→ logarithmic time	Feasible
$O(n)$	→ linear time	Feasible
$O(n \log n)$	→ log linear time	Feasible
$O(n^2)$	→ quadratic time	Sometime Feasible
$O(n^3)$	→ cubic time	Sometime Feasible
$O(2^n)$	→ exponential time	Rarely Feasible

Chapter - 2
Stack and Queue.

Stack:
Stack is a linear data structure in which an element may be inserted or deleted at only one end called top of the stack.
Stack is also called last-in-first-out (LIFO) list because the element which is inserted last, is removed first.
Stack is said to be in Overflow state when it is completely full and is said to be in underflow state if it is completely empty.

The stack ADT

1. CreateEmptyStack(S): Create or make stack S be an empty stack.
2. Push(S,x): Insert x at one end of the stack, called top.
3. Top(S): If stack S is not empty, then retrive the element at it's top.
4. Pop(S): If stack S is not empty; then delete the element at it's top.
5. IsFull(S): Determine whether stack S is full or not. Return true is S is full; return false otherwise.
6. IsEmpty(S): Determine whether stack S is empty or not. Return true if S is an empty stack; return false otherwise.
7. peek - get top data element of stack, without removing it.

Implementation of stack.

1. Array Implementation of stack (static implementation)
2. Linked-list implementation of stack (dynamic implementation)

1. Array Implementation of stack.

Implementation of stackEmpty (Underflow) and (overflow)

StackEmpty (underflow)

```
int IsEmpty()
  {
  if (top == -1)
      return 1;
  else
      return 0;
  }
```

Stack Full (overflow)

```
int IsFull()
  {
  if (top == MAXSIZE - 1)
    return 1;
  else
    return 0;
  }.
```

Stack operation:

(1) push
(2) pop
(3) peek / display.

(1) push operation.

(a) Algorithm.

1. Start.
2. check for stack overflow as
   if top == MAXSIZE-1 then
   print "stack Overflow" and exit the program.
   Else
   Increase top by 1 as
   Set, top = top+1
3. Read elements to be inserted say element.
4. Set, stack[top] = element // insert ~~int~~ item in top position.
5. Stop.

program

```
Void push (int val)
  { if (top == MAXSIZE-1)
      printf ("stack Overflow");
    else
    { top = top+1;
      stack[top] = val;
    }
```

(2) algorithm for pop operation.

1. start
2. check for stack underflow as,
   if top<0 then
   print "stack underflow" and exit the program.
   else
   remove the top element and set this element to the variable as
   Set, element = stack[top]
   Decrement top by 1 as.
   Set, top = top-1
3. Print 'element' as a deleted item from stack.
4. stop.

program.

```
Void pop()
 {
 int item;
 if (top<0)
     printf ("The stack is Empty");
 else
 {
 item = stack[top]; //Storing top element to item
                      variable.
 top = top-1;
 printf (" Popped item is = %d ", item); //displaying
                                           the deleted
                                           items.
 }
}
```

(3) Peek/display.

Algorithm.

1. Start.
2. check fo the stack Underflow as.
   if top<0 then
   print "stack underflow" and exit the program
   Else
   Set, i = top
   while (i>=0) {
   Display stack[i]
   Decrement i by 1
   End while Set, i = i-1

3. stop.

C- IMPLEMENTATION of "stack" USING ARRAY.

```
#include <stdio.h>
#define MAXSIZE 10
 int stack[MAXSIZE];
 int  top = -1;
 int isempty()
   {
     if(top ==-1)
          return 1;
      else
          return 0;
      }

  int isfull()
      {
       if(top == MAXSIZE-1)
                return 1;
          else
           return 0;
        }
 Void  peek()
       {
       if(isempty())
            printf (" stack is empty.\n");
       else
          printf (" Element at top: %d", stack[top])
       }

Void  pop()
       { int data;
         if (isempty())
                  printf ("stack is empty");

           else
             {
```

```
        data= stack[top];
        top = top-1;
        printf("popped element: %d", data);
    }
void push(int data)
  {
   if(isfull())
      printf("stack is full.\n");
    else
      {
       top = top+1;
       stack[top] = data;
       }
    }
 void display()
    {
    if (isempty())
       printf("stack is empty");
    else
      {
      for (int i=top; i>=0, i--)
          printf("%d \n", stack[i]);
      }
    }
int main()
    {
     push(3);
     push(5);
     push(9);
     peek()
     pop()
     pop()
     peek()
     display();
    }.
```

H.W Implement stack using array with "switch case".

Infix, Prefix and Postfix Notation.

One of the applications of the stack is to evaluate the expressions. We can represent the expression ~~Eqn~~ in following ways.

(i) Infix expressions.
(ii) Prefix expressions
(iii) Postfix expressions.

Infix expression:

It is an ordinary mathematical notation or expression where operator is written in between operands.

Eg: A + B Eg: (A+B) + (C-D)

Prefix expression: (Polish notation)

In this notation the operator preceds the two operands. That is ~~operand is~~ operator is written before the operand.

Eg: +AB Eg: ++AB-CD

Postfix expression: (Suffix / reverse polish notation)

In this notation the operators are written after the operands. In this notation operator follows the two operand.

AB+ AB+CD-+

Eg:

Infix	Prefix	Postfix
A+B	+AB	AB+
A+B-C	-+ABC	AB+C-
(A+B)*C-D	-*+ABCD	AB+C*D-

Precedence and associativity.

Precedence: precedence determines which of the operator in the expression is evaluated first.

For eg: A+B*C then B*C is evaluated first then A is added.

Associativity: When operators with same precedence occurs in a

expression then associativity determines which part of the expression to be evaluated first.
For eg: A+B=C

Operator	Precedence	Associativity
( )	4 –	left → right
^	1	Right → left
/, *	2	left to right
+, –	3	left → right

Conversion:

VVI (i) Infix to postfix.

Algorithm.

1. Start.
2. For given arithmetic infix expression, scan each symbol from left to right.
3. Initialize an empty stack.
4. If the scanned character is an operand, add it to the postfix string.
5. Repeat till there is data in infix expression.
   - If scanned character is operand, add it to the postfix string.
   - If the scanned character is operator and stack is empty push the character into the stack.
   - If the scanned character is an Operator and the stack is not empty, compare the precedence of the character with the element on the top of stack.
   - If top stack symbol has higher precedence over the scanned character then pop the top element of stack else push the scanned character to stack.
   - If symbol is '(' then push to stack.
   - If symbol is ")" then pop from stack until "(" is found and ignore both symbols.
6. Repeat 5 untill stack is not empty.
7. Stop.

Q. convert the following infix notation into postfix.

$(A/B) * (C * F + (A - \emptyset) * E)$

S.N.	Scan character	Op. stack.	postfix.
1	(	(	–
2.	A	(	A
3.	/	(/	A
4.	B	(/	AB
5.	)	∅	AB/
6.	*	*	AB/
7.	(	*(	AB/
8.	c	*(	AB/C
9.	*	*(*	AB/C
10.	F	*(*	AB/CF
11.	+	*(+	AB/CF*
12.	(	*(+(	AB/CF*
13.	A	*(+(	AB/CF*
14.	–	*(+(–	AB/CF*A
15.	∅	*(+(–	AB/CF*A
16.	)	*(+	AB/CF*A∅
17.	*	*(+*	AB/CF*A∅–
18.	F	*(+*	AB/CF*A∅–E
		∅	AB/CF*A∅–E*+*

Ans = AB/CF*A∅–E*+*

Q. $A + (B * C - (D / E - F) * G) * H$

Ans: ABC*DE/F–G*–H*+

Evaluation of postfix expression.

Algorithm.

1. For each character in postfix expression,
   if operand is encountered, push it onto stack.
   else if operator is encountered pop 2 elements.
   A → top element.
   B → Next to top element.

result = B operator A.
push result into stack.

2. Repeat step 1.
3. End.

Q. Evaluate:

123 + * 321 - + * Ans ⇒ 20.

For lab

Q. Postfix to infix
Q. Infix to prefix.

~~Queue~~

~~Queue is a linear~~

Stack Questions for board exam.

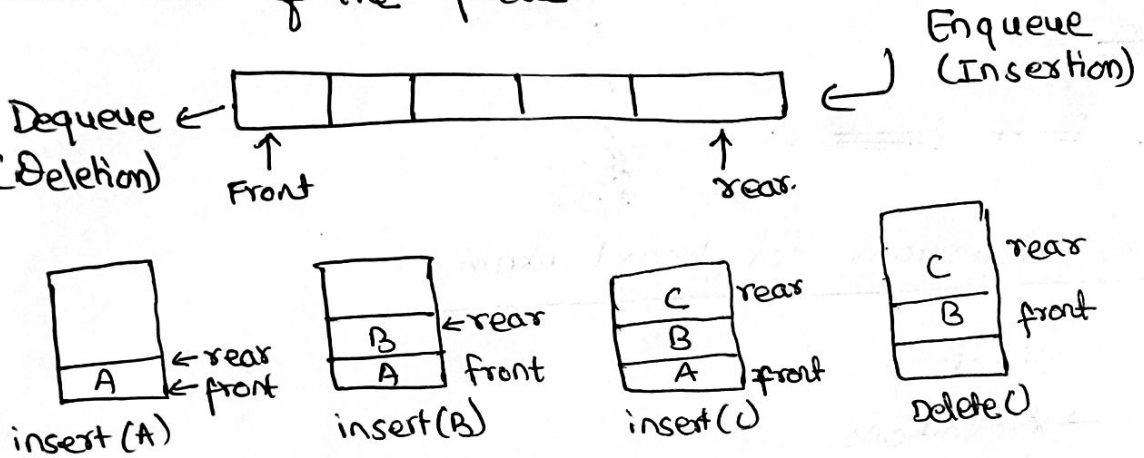
Stack
→ Definitions.
→ Operations
~~→ Push~~
→ Algorithm. for push, pop
→ Application.
↳ infix to postfix.
↳ postfix evaluation.
→ stack as ADT.

Queue.

Queue is a linear Data structure which follows First in First out mechanism.
It means, the first element inserted is the first one to be removed.

Queue uses two variables rear and front. Rear is incremented while inserting an element into the queue and front is incremented while deleting element from the queue.

Queue is also called First in First out (FIFO) system. Since the first element in queue will be the first element out of the queue.



Applications of Queue.

1. Call center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
2. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
4. Queues are used as buffers in most of the applications like MP3 player, CD player etc.
5. Queues are used in operating systems for handling interrupts

The Queue as ADT.

(a) MakeEmpty(q) : To make q as an empty queue.
(b) IsEmpty(q) : Return true is q is empty, else false.
(c) IsFull(q) : To check whether q is full or not. Return true if q is full, else true.
(d) Enqueue(q,x) : To insert data in queue.
(e) Dequeue(q,x) : To delete data from queue.
(f) Traverse(q) : To read all elements in queue. i.e display all elements

Basic Operations of Queue.

(a) Enqueue()
(b) Dequeue()
(c) Display()
(d) peek()
(e) isfull()
(f) isempty()

Implementation of Queue.

(a) Array implementation of queue (static memory allocation)
(b) Linked list implementation of queue (dynamic memory allocation).

Array Implementation of Queue.

(a) Linear array implementation (linear queue)
(b) Circular array implementation.
(c) Priority

Linear Queue.

By default the queue is linear.

In case of empty queue :

```
front = 0
rear = -1
```

Algorithm for insertion (Enque)

1. Start
2. Initialize front = 0 and rear = -1
   If rear >= MAXSIZE-1
   print "queue overflow"
   Else,
   set, rear = rear+1
   queue[rear] = item
3. stop.

Algorithm for deletion (Dequeue)

1. Start
2. If rear < front
   print "queue is empty"
   Else
   Item = queue [front++]
   print "item" as deleted element
3. Stop.