PONJESLY COLLEGE OF ENGINEERING

NAGERCOIL.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER III

CS8391- DATA STRUCTURES

UNIT II

LINEAR DATASTRUCTURES – STACKS,QUEUES

Stack ADT – Operations - Applications - Evaluating arithmetic expressions- Conversion of Infix to postfix expression - Queue ADT – Operations - Circular Queue – Priority Queue - deQueue – applications of queues.

TABLE OF CONTENTS

LINEAR DATASTRUCTURES – STACKS,QUEUES

2.1 THE STACK ADT

Q1)a) Define stack and its operations. Explain its operations with array and linked list implementation.(15)

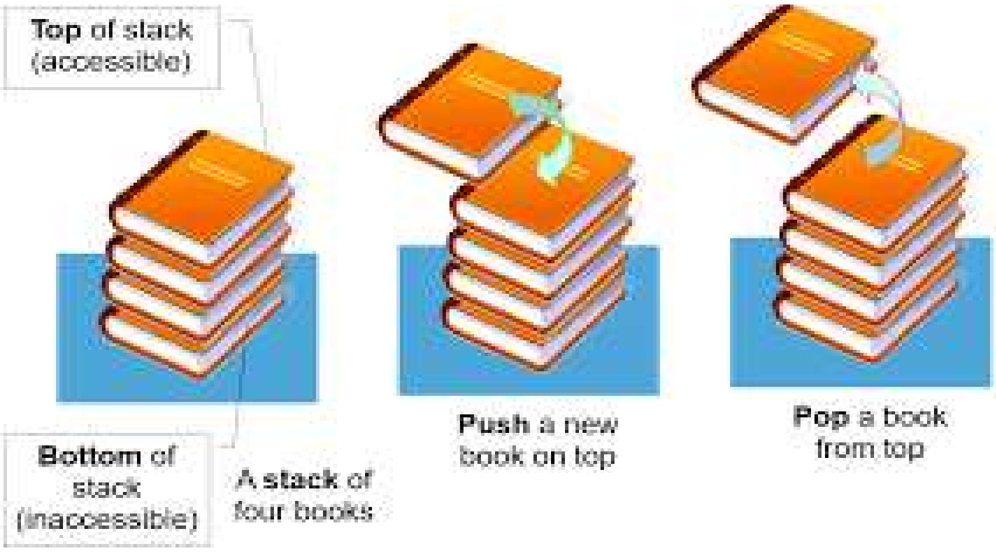
Or

Q1) b) Briefly explain different ways of stack implementation with example.(15)

ANSWER:

- A stack is a linear data structure used to store similar data items which follows **Last In First Out (LIFO) principle**, in which both insertion and deletion occur at only one end of the list known as TOP.TOP value of the empty stack is initialized as -1.

  Example: A stack of 4 books



OPERATIONS ON STACK

- The fundamentals operations performed on a stack are
  1. Push(insert)
  2. Pop-(delete)

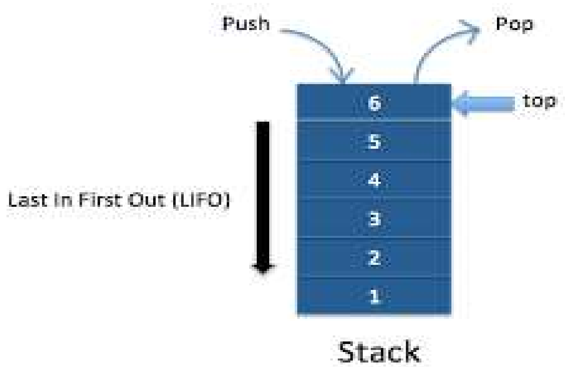
PUSH (INSERT)

- The process of inserting a element to the top of the stack . For every push operation the top is incremented by 1.

POP (DELETE)

- The process of deleting an element from the top of stack is called pop operation.

After every pop operation the top pointer is decremented by 1.



Stack

Exceptional conditions :

- Overflow :Attempt to insert an element ,when the stack is full is said as overflow
- Underflow :Attempt to delete an element ,when the stack is empty is known as underflow.

IMPLEMENTATION OF STACK

- Stack can be implemented as
  1. Arrays
  2. Pointers (or) Linked List

ARRAY IMPLEMENTATION

- In this implementation each stack is associated with a pop pointer, Which is -1 for empty stack
  - To push an element X on to the stack ,Top pointer is incremented and then set stack[Top]=X.
  - To pop an element ,the stack [top] value is returned and the top pointer is decremented
  - Po on an empty stack or push on a full stack will exceed the array bounds.
- The size of the array must be declared first which is not a overhead .Initially the top is 0 to denote the stack is empty.
- A stack can be declared as a structure containing two objects.
  1. An array to hold the elements of the stack
  2. An integer to indicate the position of the current stack top element within the array

Stack declaration

```
 #define size10
struct stack
{
  int s[size];
  int top;
}st;
```

Routine for empty operation:

```
int  stempty()
{
   if (st.top==-1)
    return1;
else
    return 0;
}
```

Routine for stack is full

```
int  stfull()
{
   if (st.top>==size -1)
    return1;
else
    return 0;   }
```

Routine To Return Top Element Of The Stack

```
int TopElement (Stack S)
{
if (! IsEmpty (S))
return s[Top];
else
Error ("Empty Stack");
return 0;  }
```

Routine To Push An Element Onto A Stack

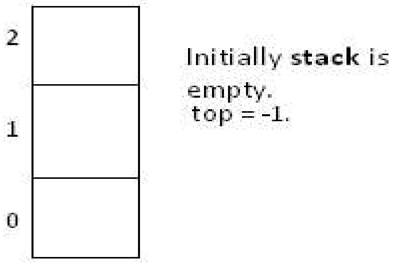
```
void push (int item)
{
St.top++;
St.S[St.top]= item;
}
```

Routine To Pop An Element From A Stack

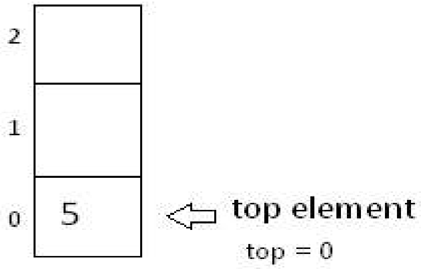
```
int pop()
{
int item;
item = St.s[St.top];
st.top--;
return (item);
}
```

Example: Insert elements 5,10,24,12 into a stack of size 3 and delete all elements to make empty stack.

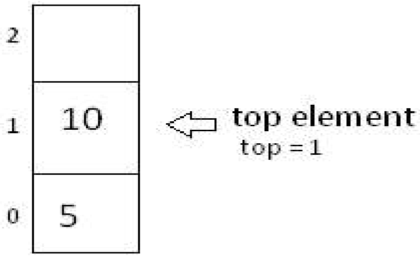
1)



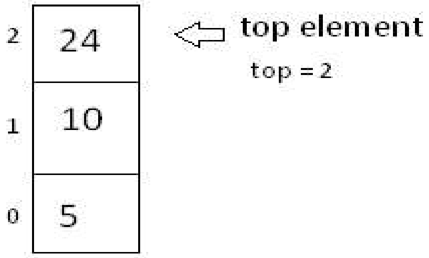
2) push(stack, 5, 3)



3) push(stack, 10, 3)



4) push(stack, 24, 3)

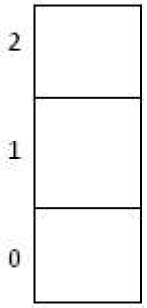


5) As top = 2, current size of stack is top+1 , i.e 3 . Now stack is full,as 3 is maximum size of stack

6) push(stack, 12, 3)

As ,stack is full ,it will show
OVERFLOW CONDTION!

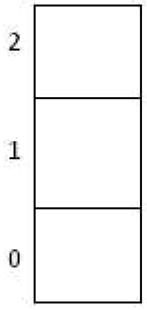
7) Deleting all elements from stack. ⇒ pop(stack, 3)
pop(stack, 3)
pop(stack, 3)



EMPTY STACK !!

top = -1

8) pop(stack, 3)



As **stack** is empty, further deleting will cause

UNDERFLOW CONDITION!

LINKED LIST IMPLEMENTATION OF STACK:

- A stack can also be implemented as a linked structure .In such an implementation the stack consists of a sequence of nodes.
- Each node is a structure containing data item and a pointer to the next node if one exists. This pointer is called a ***link node***.
- The first node is considered to be the top of the stack and the pointer is called top.
- The last node in the bottom of the stack and its pointer is set to NULL.

- An empty stack will have top=NULL.
- The memory for each node is dynamically allocated using malloc.
- When an item is pushed, a node for it is created and when an item is popped its node is freed using free.

Declaration

```
Struct Stack
{
int data;
Struct Stack * next;
}
struct Stack *topptr;
```

Routine To Check Whether The Stack Is Empty

```
int IsEmpty(Stack S)
{
if ( S → Next == NULL)
return(1);
}
```

Routine To Return Top Element In A Stack

```
int Top (stack S)
{
if (! Is Empty (S))
return S → Next → Element;
Error (“empty Stack”);
return 0;
}
```

Routine To Push

```
void push( int item)
{
Struct stack *new;
new = (struct stack *) malloc (sizeof (Struct stack));
new → data = item;
new → next = topper;
topper = new;
}
```

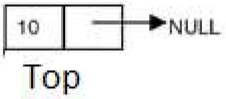
Routine To Pop

```
int pop()
{
int item;
if(topper != NULL)
{
item = topper → data;
topper = tppper → next;
return (item);
}
else
{
printf (" Stack is empty");
}
}
```

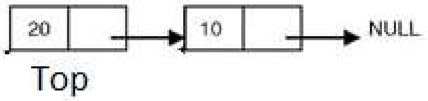
Example: Insert elements 10,20,30,40 into a stack and delete all elements to make empty stack.

1) Top=NULL

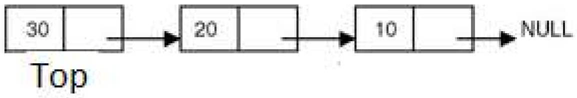
2) Push(10)



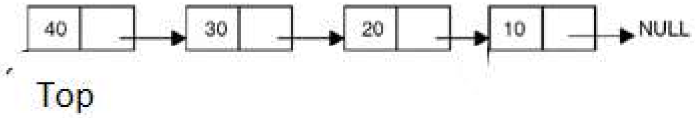
3) Push(20)



4) Push(30)

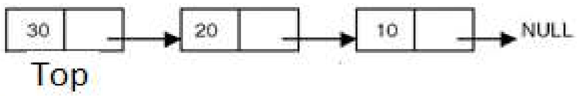


5) Push(40)

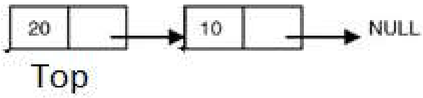


6) Deleting elements to make stack empty

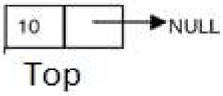
Pop()



7) Pop()



8) Pop()



9) Pop()

Top=NULL

2.2 APPLICATIONS OF STACK

Q2) a) What are the applications of stack?(2)

Or

Q2) b) Name some applications of stack.(2)

ANSWER:

- Some of the applications of stack are,
  1) Evaluating arithmetic expression
  2) Balancing the symbols
  3) Towers of Hanoi
  4) Function calls
  5) 8 Queen problem

Q3) a) What are the different way of representing the algebraic expression(2)

Or

Q3) b) Define infix, postfix and prefix notation.(2)

ANSWER:

- There are 3 different way of representing the algebraic expression .They are
  - Infix Notation
  - Postfix Notation
  - Prefix Notation

Infix Notation

- In infix notation the arithmetic operator appears between the two operands to which it is being applied.

- For eg :
  A/B+C

Postfix Notation

- The arithmetic operator appears directly after the two operands to which it applie .Also called reverse polish notation.
- For eg :
  AB/C+

Prefix Notation

- The arithmetic operator is placed before the two operands to which it applies. Also called as prefix notation.
- For eg:
  +/ ABC

CONVERSION OF INFIX TO POSTFIX

Q4) a) Write down the steps to convert infix to postfix and evaluation of arithmetic expression with examples(15)

Or

Q4) b) Briefly explain about evaluating arithmetic expression(15)

ANSWER:

To evaluate arithmetic expression first convert the given infix expression into postfix expression and then evaluate that postfix expression using stack.

Steps For Conversion of Infix To Postfix:

- Read the infix expression one character at a time until it encounters the delimiter #

Step 1: If the character is an operand placed it on to the output

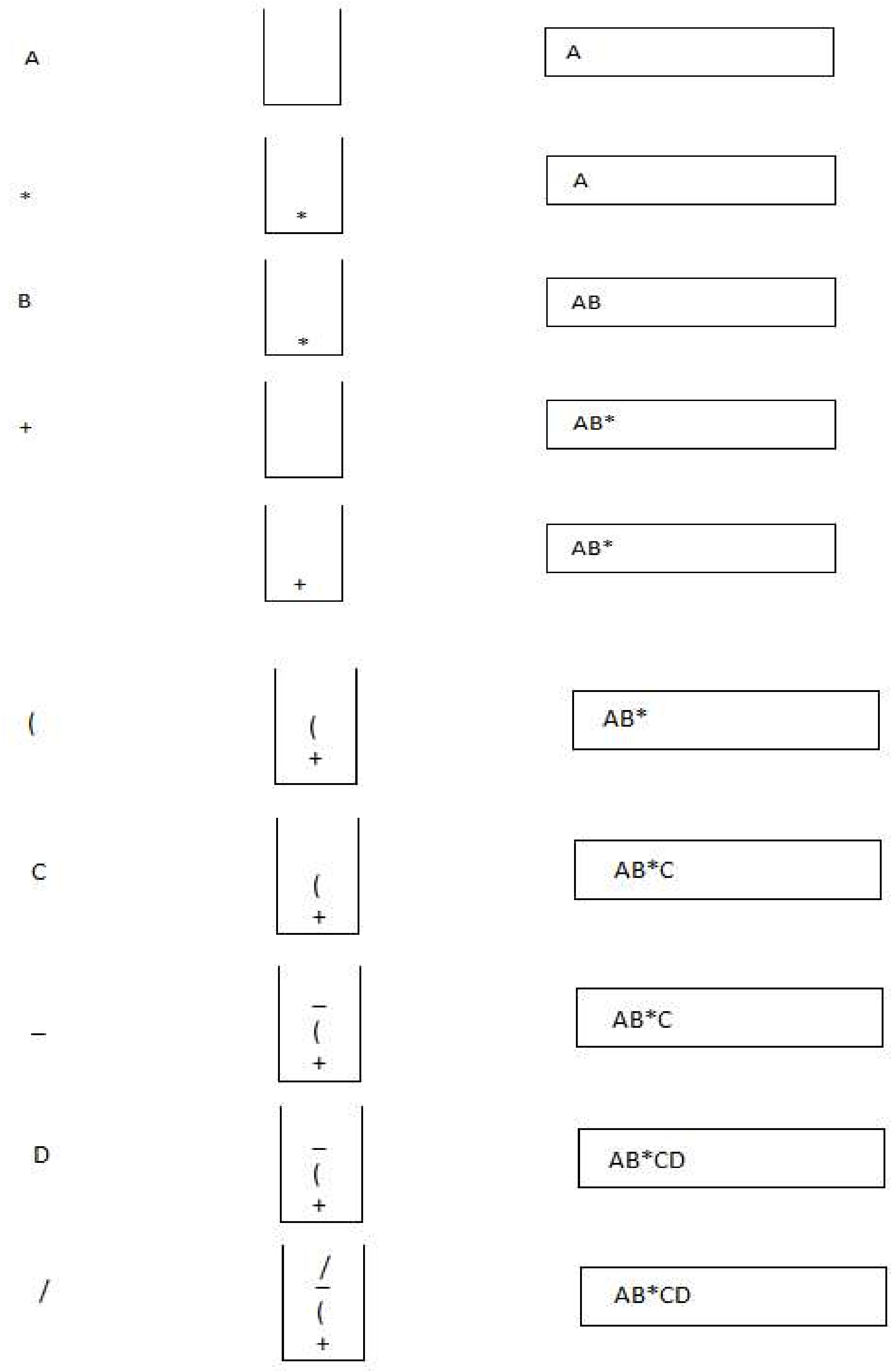
Step 2: If the character is an operator push it on to the stack .If the stack operator has a higher or equal priority than input operator then pop that operator from the stack and place it on to the output.

Step 3: If the character is a left parenthesis push it on to the stack

Step 4: If the character is right parenthesis pop all the operator from the stack till it encounters left parenthesis, discard both the parenthesis in the output.

Example 1: Infix Expression :A*B +(C-D/E)#

Read Character Stack output



A		A
*	*	A
B	*	AB
+		AB*
	+	AB*
(	(+	AB*
C	(+	AB*C
–	– (+	AB*C
D	– (+	AB*CD
/	/ – (+	AB*CD

E	/ – (+	AB*C DE
)	+	AB*CDE/-
#		AB*CD E/-+

EVALUATING ARITHMETIC EXPRESSION

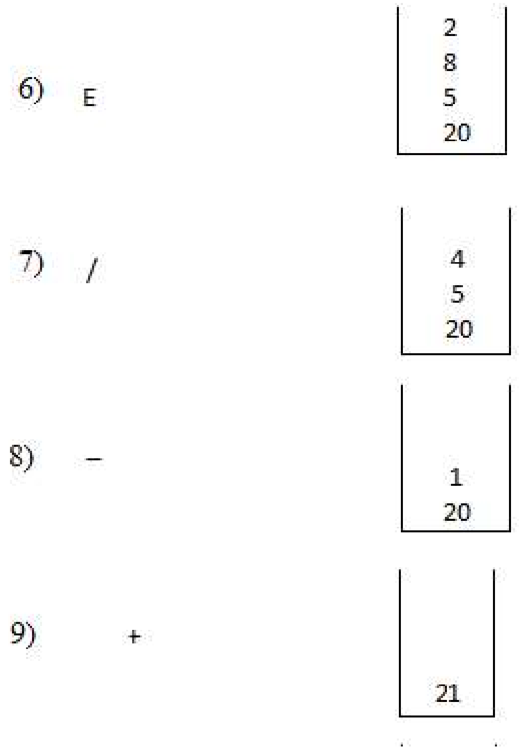
Convert infix to postfix expression and Read the postfix expression one character at a time until it encounters the delimiter # and the follow the below steps.

Step1: If the character is an operand pus its associated value on to the stack

Step 2: If the character is an operator POP two values from the stack, apply the operator to them and push the result on to the stack

Let us consider the symbols A,B,C,D,E has the associated values and evaluate AB*CDE/ -+ as A=4,B=5,C=5,D=8,E=2.

Read Character		Stack
1)	A	4
2)	B	5 4
3)	*	20
4)	C	5 20
5)	D	8 5 20



Result = 21

Q5) a) Briefly explain some applications of stack.(13)

Or

Q5) b) Explain the following

i) Balancing the symbols(5)

ii) Towers of Hanoi (8)

iii) Function calls(2)

ANSWER:

i) BALANCING THE SYMBOLS:

- Read one character at a time until it encounters the delimiter #

Step 1: If the character is an opening symbol push it on to the stack.

Step 2: If the character is a closing symbol and if the stack is empty report an error as missing opening symbol

Step 3: If it is a closing symbol and if it has corresponding opening symbol in the stack pop it from the stack.Otherwise report an error as mismatched symbols

Step 4: At the end of file. if the stack is not empty ,report an error as missing closing symbol .Otherwise ,report as Balanced symbols

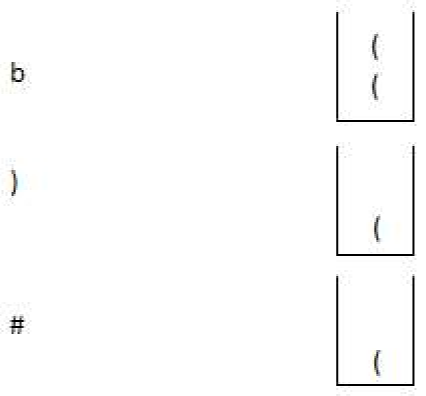
Eg: Let us consider the expression as (a+b)#

Read Character	Stack
(	(
a	(
+	(
b	(
)	
#	

Stack is Empty

Eg.2: Consider the expression ((a + b)#

Read character	Stack
(	(
(	((
a	((
+	((



Stack is not Empty

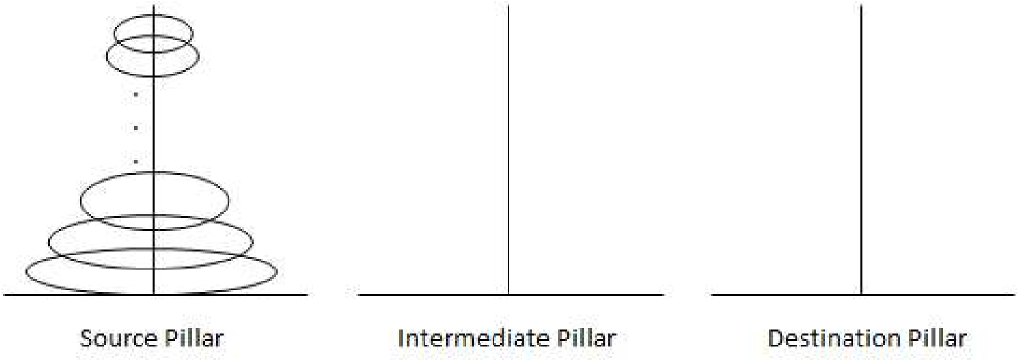
ii) **TOWERS OF HANOI:**

It is one of the example illustrate the recursion technique. The problem is moving a collection of N disks of decreasing size from one pillar to another pillar .The movement of the disk is restricted by following rule

Rule 1: Only one disk could be moved at a time

Rule 2: No large disk could ever reside on a pillar on top of a smaller disk;

Rule 3: A 3rd pillar could be used as an intermediate to store one or more disks, while they were being moved from source to destination



Recursive Solution:

N- represents the number of disks.

Step1: If N=1 move the disks from A to c

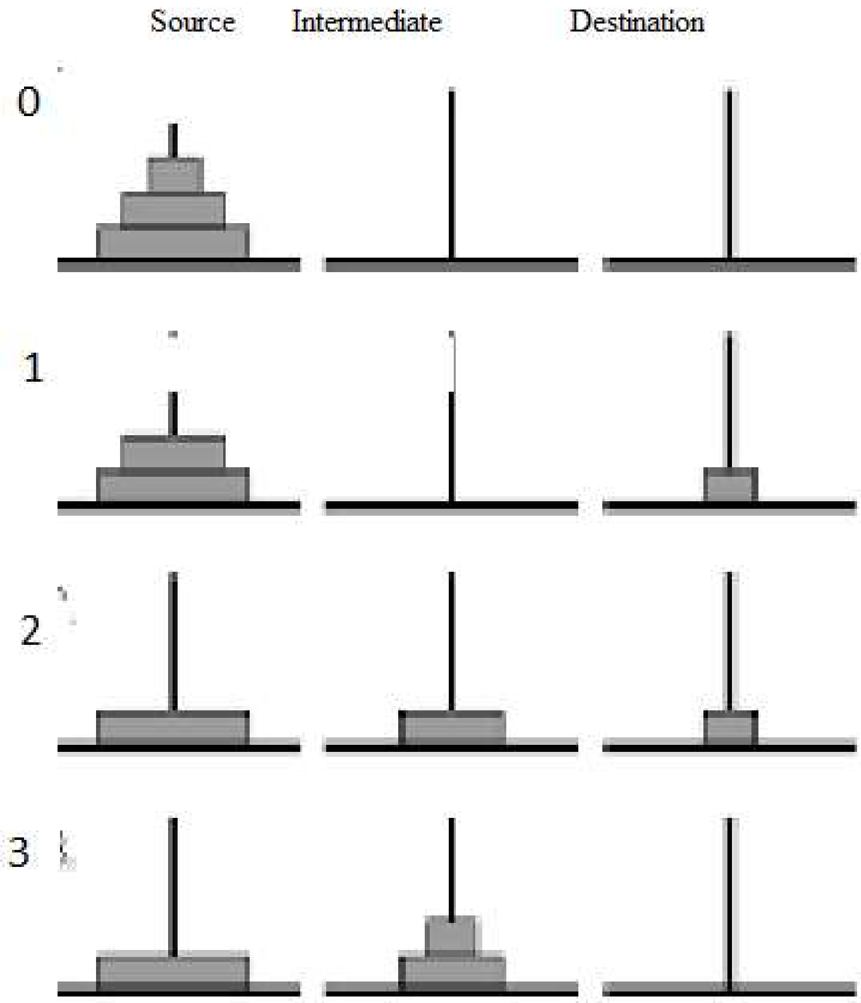
Step2: If N=2 move the 1st disk from A to B .Then move the 2nd disk from A to C. Then move the 1st disk from B to C

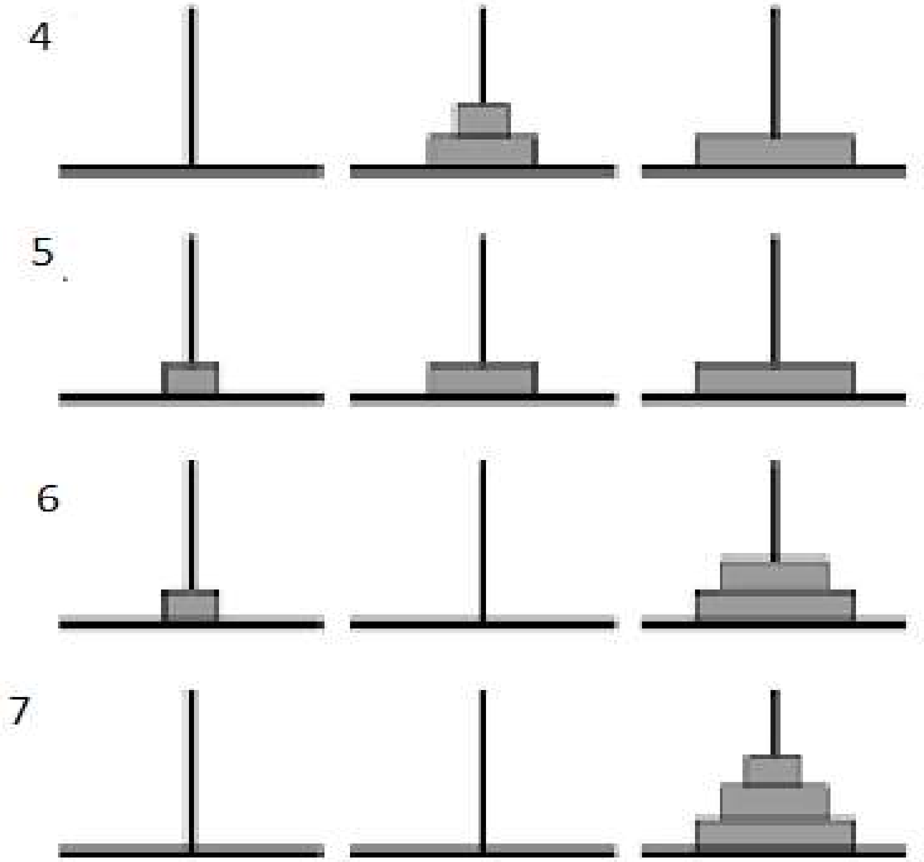
Step 3: If N=3 .Repeat the step 2 to move the first 2 disks from A to B using intermediate .Then the 3rd disk is moved from A to C .Then repeat the step 2 to move 2 disks from B to C using A intermediate

In general, to move N disks. Apply the recursive technique to move N-1 disks from A to B using C as an intermediate. Then move the Nth disk from A to C .Then again apply the recursive technique to move N-1 disks from B to C using A as an intermediate

Recursive Routine For Towers Of Hanoi

```
void Hanoi ( int n, char s,char d, char i)
{
/*n = no of disks,s= source,d= destination, i= intermediate*/
if(n = = 1)
{
print (s,d);
return;
else
{
hanoi(n-1,s);
print (s,d);
Hanoi(n-1, I,d,s);
return;
}
}
```

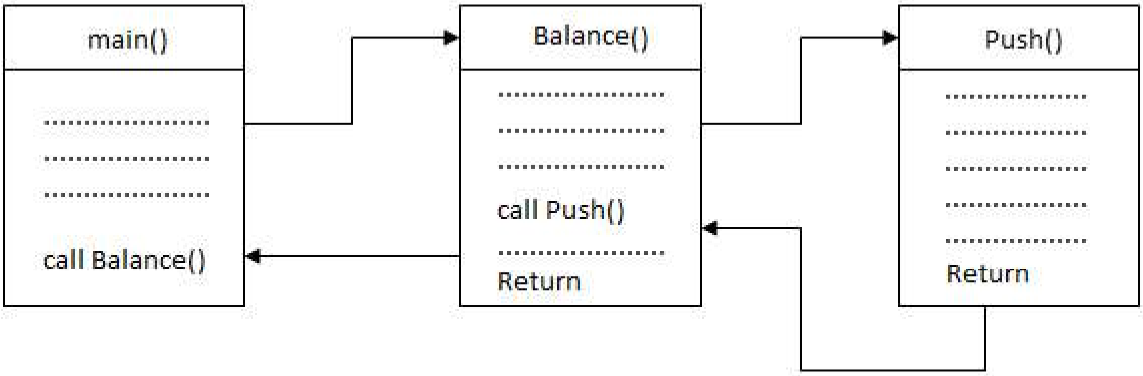




FUNCTIONS CALLS

When a call is made to a new function all the variables local to the calling routine need to be saved, otherwise the new function will overwrite the calling routine variables.

Similarly the current location address in the routine must be saved so that the new function knows where to go after it is completed.



2.3THE QUEUE ADT

Q6)a) Define Queue and its operations. Explain its operations with array and linked list implementation.(15)

Or

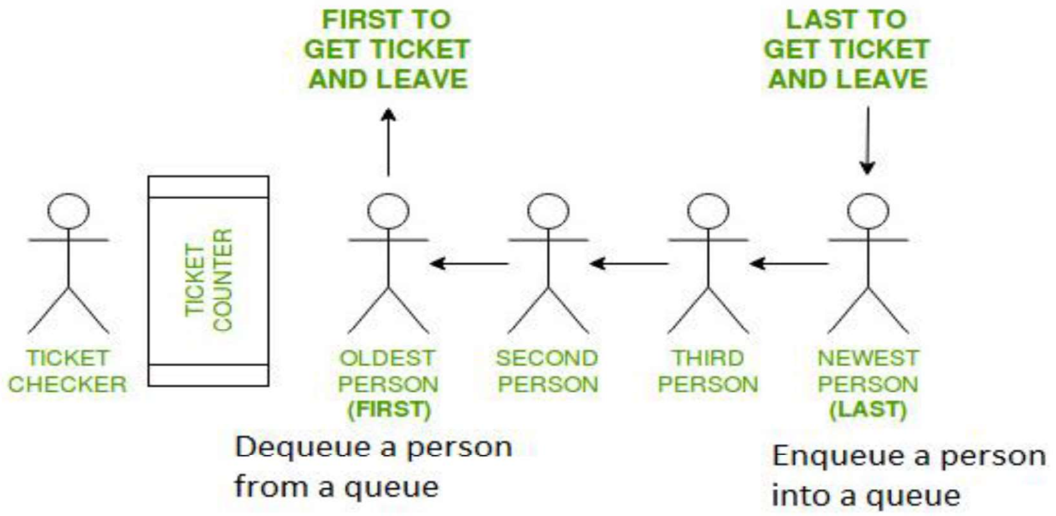
Q6) b) Briefly explain different ways of Queue implementation with example.(15)

ANSWER:

Queue Model:

- A Queue is a linear data structures which follows First In First Out (FIFO) principle in which insertion is performed at rear end and deletion is performed front end.

Example: Waiting line in Reservation counter


FIRST TO GET TICKET AND LEAVE
LAST TO GET TICKET AND LEAVE
TICKET CHECKER
TICKET COUNTER
OLDEST PERSON (FIRST)
SECOND PERSON
THIRD PERSON
NEWEST PERSON (LAST)
Dequeue a person from a queue
Enqueue a person into a queue

OPERATIONS ON QUEUE:

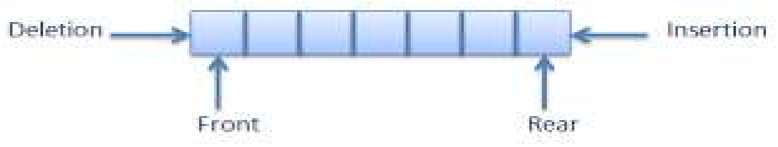
- The fundamentals operations performed on queue are
  1. Enqueue
  2. Dequeue

Enqueue:

The process of inserting element

Dequeue:

The process of deleting an element



First In First Out(FIFO)

Exception Conditions:

Overflow :

Attempt to insert an element, the queue is full is said to be overflow condition.

Underflow:

Attempt to delete an element from the queue, when the queue is empty is said to be overflow.

IMPLEMENTATION OF QUEUE:

Queue can be implemented using

1. Arrays
2. Pointers (or) Linked List

Array Implementation:

Here queue Q is associated with two ends

1. Rear end
2. Frond end

Insertion or Enqueue:

To insert an element X into the queue Q ,the rear end is incremented by 1 and then set queue [Rear]=X

Routine To Enqueue

```
void enqueue(int X)
{
if(rear > = max_ArraySize)
print("Queue overflow");
else
{
Rear = Rear +1;
Queue [Rear] = X;
}
}
```

Deletion (or) Dequeue:

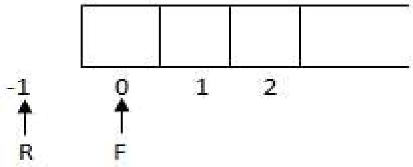
To delete an element, the queue [Front] is returned and the Front end is incremented by 1

Routine For Dequeue

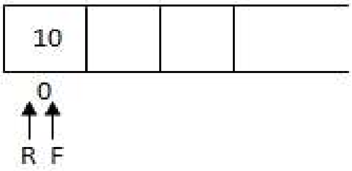
```
void delete()
{
if (Front < 0)
{
print ("queue underflow");
else
{
X= Queue[Front];
if (Front = = Rear)
{
Front =0;
Rear = -1;
}
else
Front = Front +1;
}
}
```

Example: Insert elements 5,10,24,12 into a Queue of size 3 and delete all elements to make empty stack.

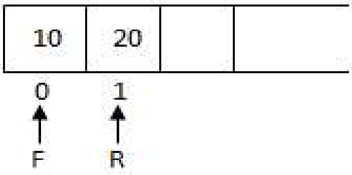
Empty Queue



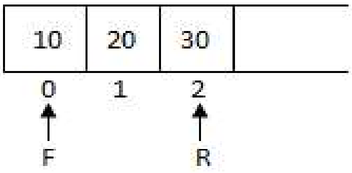
After Enqueue



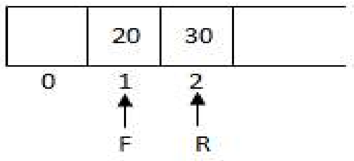
After Enqueue



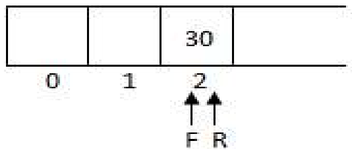
After Enqueue



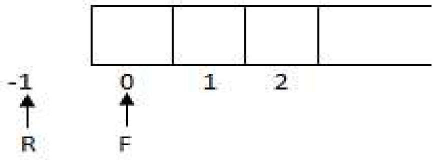
After Dequeue (10)



After Dequeue (20)



After Dequeue (30)



- ➢ In Dequeue operation, if Front = Rear, then reset both the ends to the initial values (ie, F = 0, R = -1)

Linked List Implementation:

Enqueue operation is performed at the end of the list.

Dequeue operation is performed at the front of the list.

Declaration

```
Struct node
{
int Element;
Struct Node * Next;
}* Front = NULL, * Rear + NULL;
```

Routine To Check Whether The Queue Is Empty

```
int IsEmpty (Queue Q)
{
if(Q → Next = = NULL)
return (1);
}
```

Routine To Check An Empty Queue

```
Struct CreateQueue()
{
Queue Q;
Q = Malloc (Sizeof (Struct Node));
if(Q = = NULL)
Error (“Out of space”);
MakeEmpty (Q);
return Q;
}
```

```
void MakeEmpty (Queue Q)
{
if (Q = = NULL)
{
Error ("Create Queue First');
else
while(! IsEmpty (Q)
Dequeue(Q);
}
}
```

Routine To Enqueue An Element In Queue

```
void Enqueue(int X)
{
Struct node * newnode;newnode = malloc (Sizeof (Struct node));
if(Rear = = NULL)
{
newnode → data = X;
newnode → Next = NULL:
Front = newnode;
Rear = newnode;
}
else
{
newnode → data = X;
newnode → Next = NULL:
Rear --. Next = newnode;
Rear = newnode;
}
}
```

Routine To Dequeue An Element From Queue

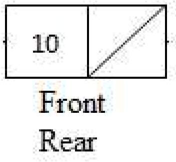
```
void Dequeue
{
struct node * temp;
if (front = = NULL)
Error("Queue Underflow");
else
{
temp = front;
if(front = = rear)
{
```

```
front  NULL;
Rear = NULL;
}
else
Front = Front → next;
print (temp) →
free (temp);
}
}
```

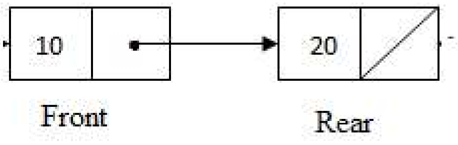
Example:Insert 10,20 into queue and perform delete operation to make a empty queue.

1) Front=Rear=NULL

2) Enqueue(10)

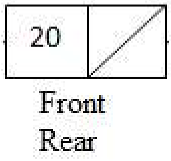


3) Enqueue(20)



4) Delete all elements to make queue empty

Dequeue()



5) Dequeue()

Front=Rear=NULL

2.4 CIRCULAR QUEUE

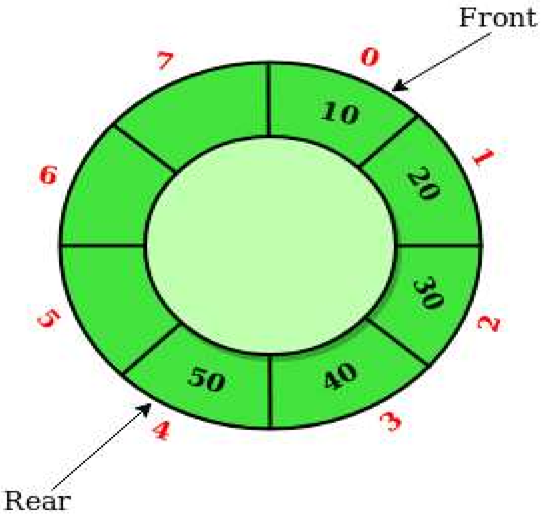
Q7)a) Define Circular Queue and its advantages. Explain its operations with array implementation.(15)

Or

Q7) b) Briefly explain Ring Buffer implementation with example.(15)

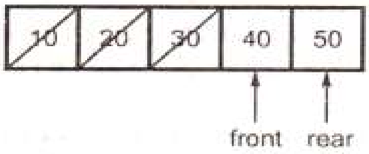
ANSWER:

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'.



Need of Circular Queue

As we have seen, in case of linear queue the elements get deleted logically as shown below,



Linear queue

- ➢ We have deleted the elements 10, 20 and 30 means simply the front pointer is shifted ahead. We will consider a queue from front to rear always.
- ➢ And now if we try to insert any more element then it won't be possible as it is going to give ***"queue full !"*** message.
- ➢ Although there is a space of elements 10, 20 and 30 (these are deleted elements), we cannot utilize them because queue is nothing but a linear array.
- ➢ Hence there is a concept called **circular queue**. The main **advantage** of circular queue is we can utilize the space of the queue fully.
- ➢

Insertion

To perform the insertion , the position of the rear end is calculated by the relation

rear=(Rear+1)%maxsize

Queue[Rear]=item

Routine To Insert An Element In Circular Queue

```
void CEnqueue (int X)
{
if( Front = = (rear +1) % MaxSize)
print (“Queue overflow”);
else
{
if(front = = -1;
front = rear = 0;
else
rear =  (rear +1)% MaxSize;
CQueue[rear]=X;
}
}
```

Deletion:

To perform the deletion, the position of the front end is calculated by the relation

Value=(Queue [front])

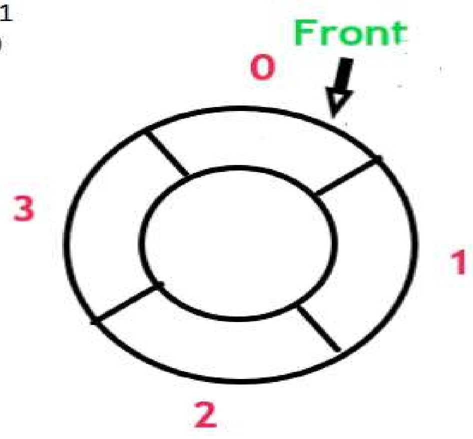
Front=(Front+1)%maxsize

Routine To Delete An Element From Circular Queue

```
void CDequeue ( )
{
if( Front = = -1)
print (“Queue underflow”);
else
{
X = CQueue [Front];
if(front = = Rear ;
front = rear = -1;
else
Front =  (Front +1)% MaxSize;
}
return(X);
}
}
```

Example: Insert elements 10,20,30 into circular queue of size 4 and perform delete operation and again insert elements 40 and 50 .

1)Rear =-1
Front=0

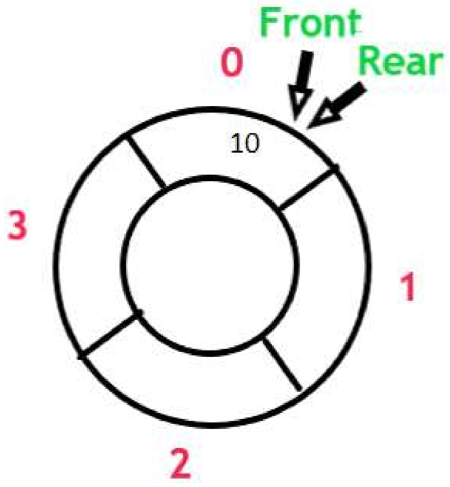


2)Enqueue(10)

Rear= (-1+1)%4

=0%4

Rear=0

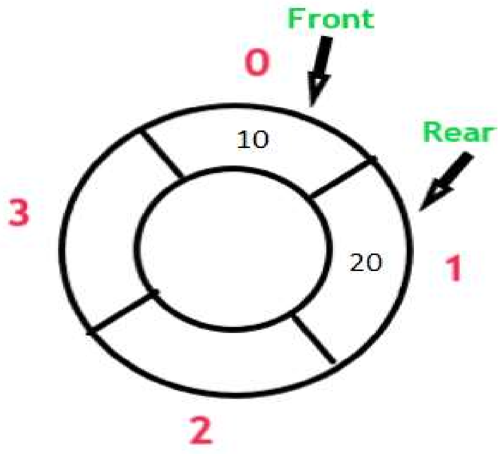


3)Enqueue(20)

Rear= (0+1)%4

=1%4

Rear=1

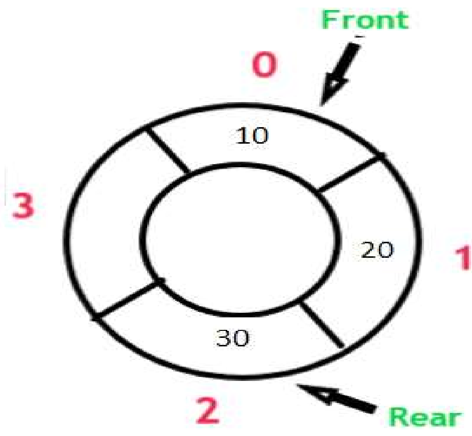


4)Enqueue(30)

Rear= (1+1)%4

=2%4

Rear=2



5) Dequeue()

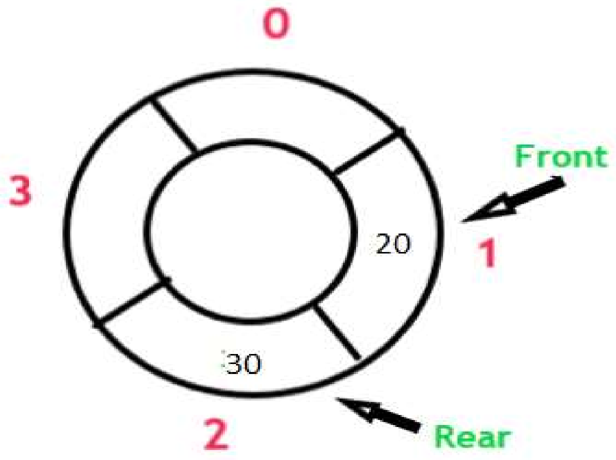
The element in front position is 10.

So delete it.

Front=(0+1)%4

=1%4

Front=1

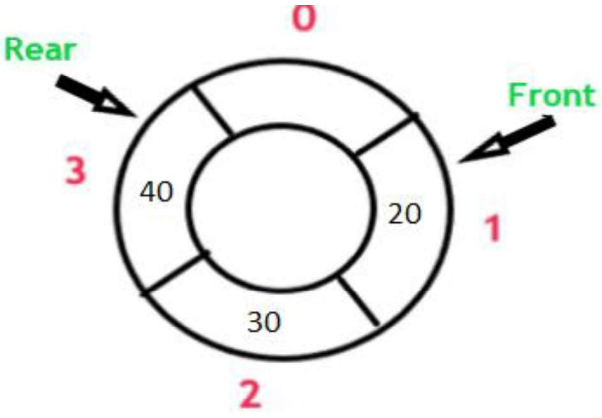


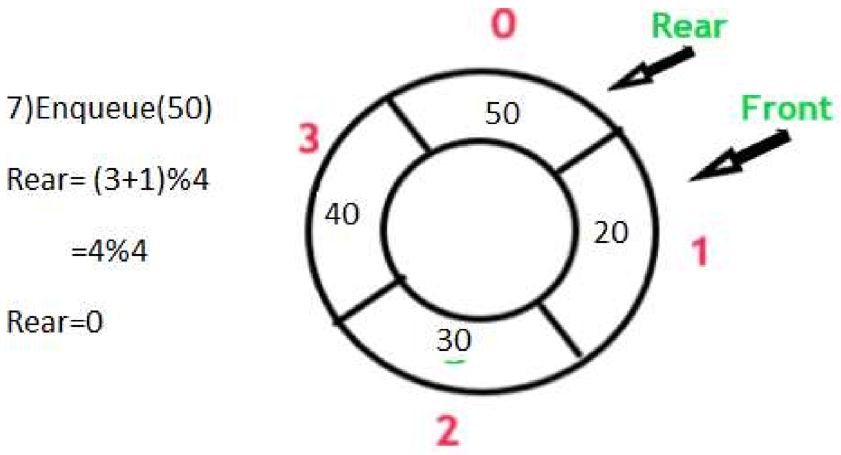
6)Enqueue(40)

Rear= (2+1)%4

=3%4

Rear=3





7)Enqueue(50)

Rear= (3+1)%4

=4%4

Rear=0

Applications of A Circular Queue

Memory management: circular queue is used in memory management.

Process Scheduling: A CPU uses a queue to schedule processes.

Traffic Systems: Queues are also used in traffic systems.

2.5 DOUBLE ENDED QUEUE

Q8a) Define Double Ended Queue and its types. Explain its operations with array implementation.(15)

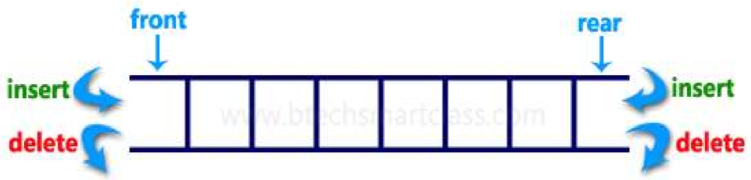
Or

Q8) b) Briefly explain Different operations of dequeue with example.(15)

ANSWER:

Double Ended Queue

Double ended queue is a more generalized form of queue data structure which allows insertion and removal of elements from both the ends, i.e , front and back. It is also often called a head-tail linked list.



Types of Double Ended Queue

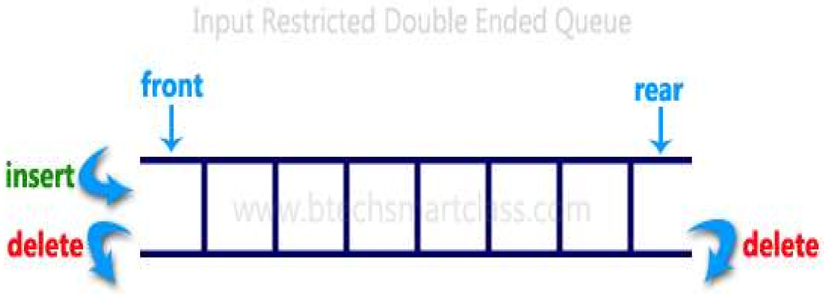
Double Ended Queue can be represented in TWO ways, those are as follows...

1) Input Restricted Double Ended Queue

2) Output Restricted Double Ended Queue

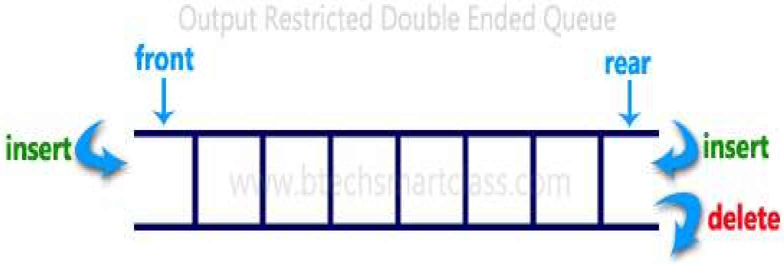
Input Restricted Double Ended Queue

In input restricted double-ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



Output Restricted Double Ended Queue

In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



Operations on Deque:

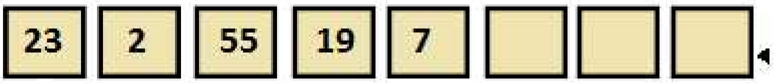
Mainly the following four basic operations are performed on queue:

insertFront(): Adds an item at the front of Deque.
insertRear(): Adds an item at the back of Deque.
deleteFront(): Deletes an item from front of Deque.
deleteRear(): Deletes an item from back of Deque.

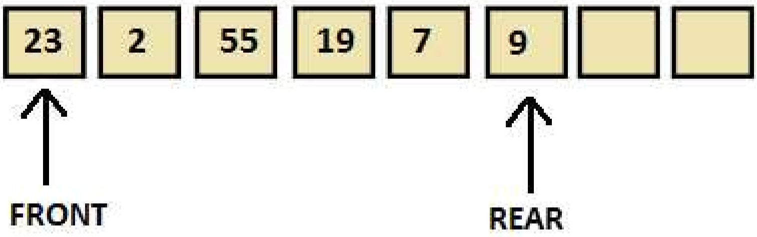
Insertion at Rear(Back)

Example:

Initial Double ended Queue



Perform InsertRear(9)



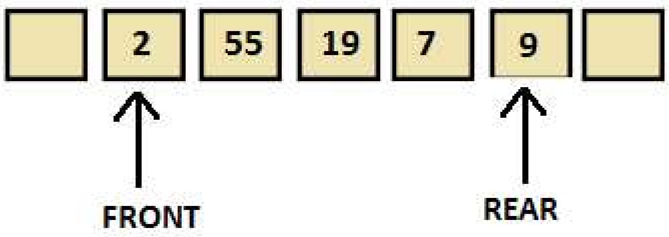
```
InsertRear(int item)
{
if(rear==MAX)
     Print("Queue is Overflow”);
else
{
     rear=rear+1;
    deq[rear]=item;

}
if rear=0
    rear=1;
 if front=0
    front=1;
return;
}
```

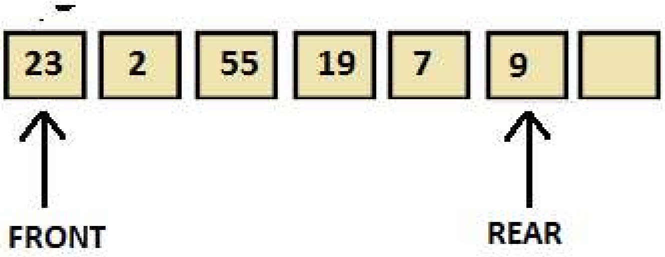
Insertion at Front

Example

Initial Double Ended Queue



Perform InsertFront(23)



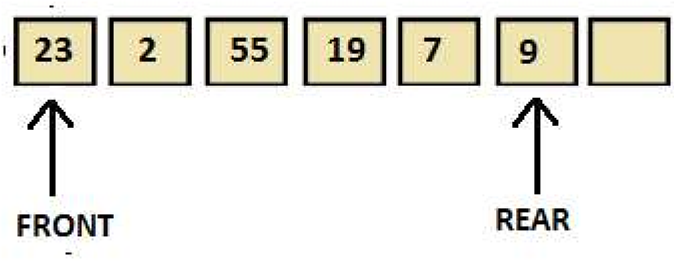
```
InsertFront(int item)
{
if(front<=1)
        Print("Cannot add item at the front”);

else
{
      front=front-1;
      deq[front]=item;
}
   return;
}
```

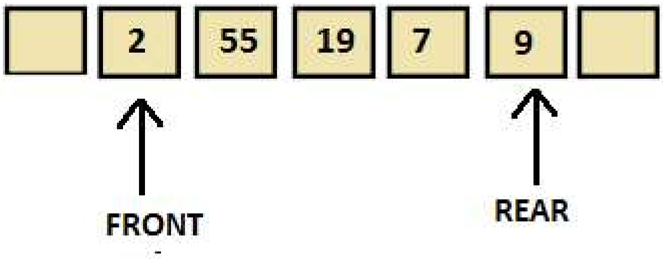
Deletion From Front

Example:

Initial Double Ended queue



Perform DeleteFront()

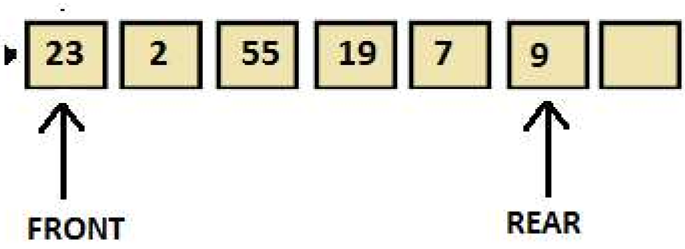


```
DeleteFront()
{
if front=0
    print(" Queue is Underflow”);
else
{
   item=deq[front];
   print(“Deleted element is”,no);
}
if front=rear
{
  front=0;
  rear=0;
}
else
    front=front+1;
return;
}
```

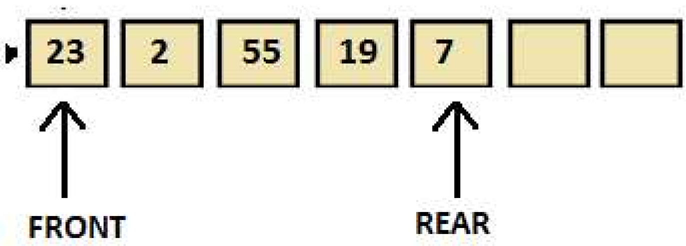
Deletion from Rear(Back)

Example

Initial Double Ended queue



Perform DeleteRear()



```
DeleteRear()
{
if rear=0
   print("Cannot delete value at rear end");
else
   item=deq[rear];
if front= rear
{
  front=0;
  rear=0;
}
else
  rear=rear-1;
return;
}
```

2.6 PRIORITY QUEUE:

Q9) a) Define Priority Queue and Explain its implementation with examples.

Or

Q9) b) What is the efficient way of implementing priority queue.Explain it with all operations.

Or

Q9) c) Define Binary Heap and its propertie.Explain its operation with examples.

ANSWER:

- The priority queue is a data structure having a collection of elements which are associated with specific ordering.

BINARY HEAPS

The efficient way of implenting priority queue is Binary heap. Binary heap is merely referred as Heaps. Heap have two properties namely,

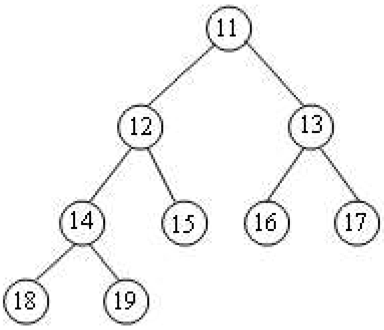
1) Structure Property
2) Heap Order Property

1) Structure Property

A heap should be complete binary tree ,which is a completely filled binary tree with the possible exception of the bottom level,which is filled from left to right.

A complete binary tree of height H has between 2H and 2H+1 -1 nodes.This implies that the height of a complete binary tree is [log n] (ie) O(log n)

Eg:



2) Heap Order Property

The property that allows the operation to be performrd quickly is the heap order property. There are two types of Heap

Min Heap

Every parent should have minimum value.To find the minimum element quickly it makes sense that the smallest element should be at the root.

Max Heap

Every parent should have maximum value.To find the maximum element quickly it makes sense that the smallest element should be at the root.

Declaration of Heap

```
Struct Heapstruct
```

```
{
        int capacity;
        int size;
        int * elements;
};
```

3) Basic Heap Operations

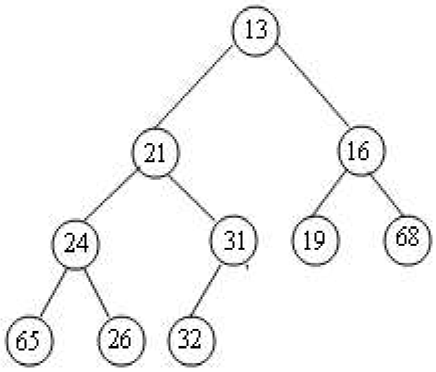
To perform the insert and Deletemin opeartions ensure, that the heap order property is maintained.

(i) Insert

To insert an element X into the heap. Create the hole in the next available location.If X can be placed in the hole without violating heap order, then place the element X there itself. Otherwise, we slide the element that is in the hole's parent node into the hole, thus bubbling the hole up toward the root. This process continues until X can be placed in the hole. This general strategy is known as percolate up. In which the new element is percolated up the heap until the correct location is found.

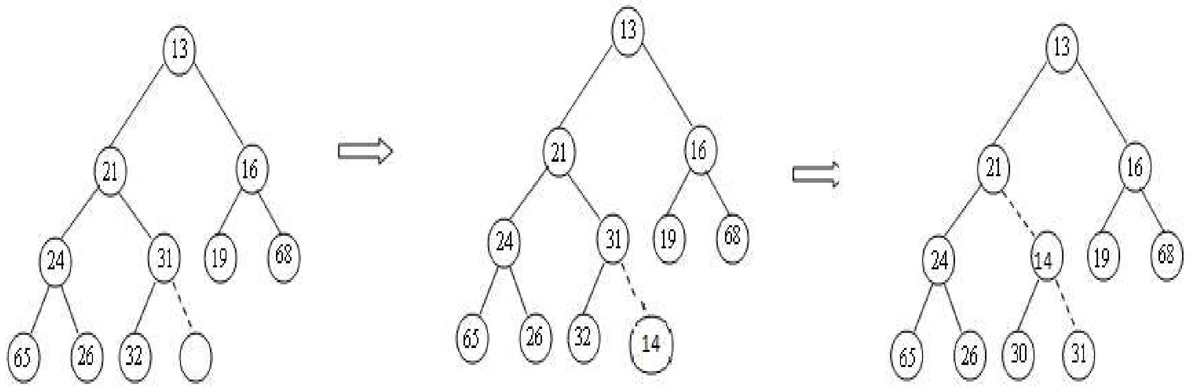
Example :

Insert element 14 in this Heap



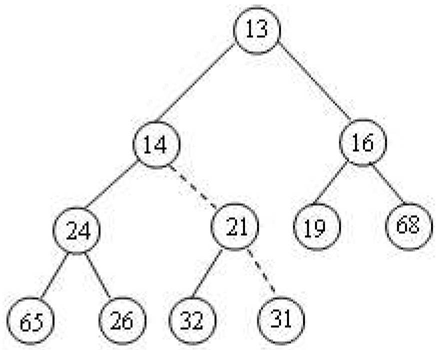
Solution :

Step1: Create Hole in the next available position and insert 14.But 14 is smaller than 31.So percolate up 14.



Step 2:

Again 14 is smaller than 21.So percolate up 14.



Now 14 is greater than 13.So it's a Final Heap.

Routine to perform INSERT operation

```
Void insert ( int X, Priority Queue H)
{
        int  I;
        if( IsFull(H))
        {
           Error ( “Priority Queue is Full”);
           Return;
        }
        for(i=++ H⍰ size; H⍰Elements[i/2]>X;i/=2)
            H ⍰ Elements [i] = H ⍰ Elements [i/2]
            H ⍰ Elements [i] = X;
        }
```

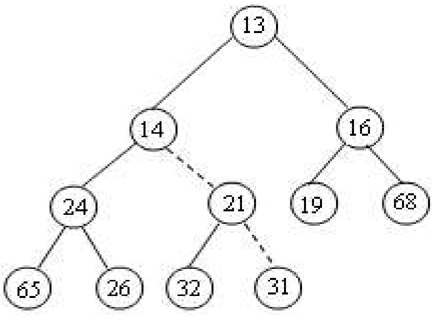
ii) DeleteMin

DeleteMin operation is deleting the minimum element from the heap. In binary heap the minimum element is found in the root. When this minimum is removed a hole is created at the root. Since the heap becomes one smaller makes the last element X in the heap is to move somewhere in the heap.

If X can be placed in hole without violating heap order property place it. Otherwise we slide the smaller of the hole's children into the hole, thus pushing the hole down one level.

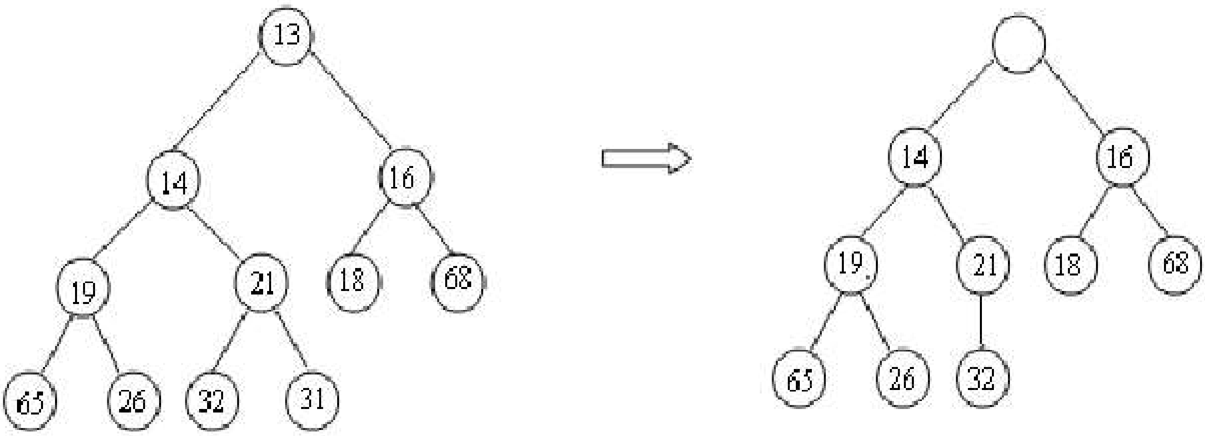
We repeat until X can be placed in the hole. This general strategy is known as percolate down.

Example:Perform Deletemin Operation in this Heap

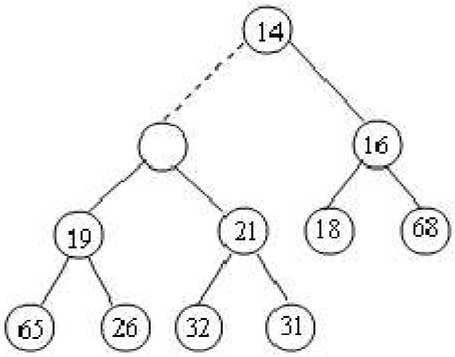


Solution:

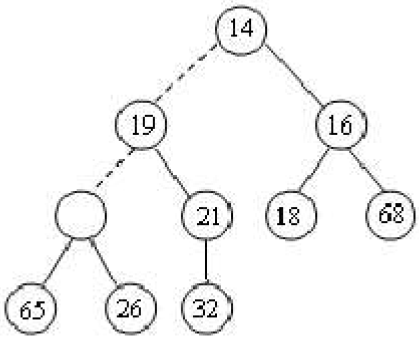
Step1:13 is the smallest number.So delete it.



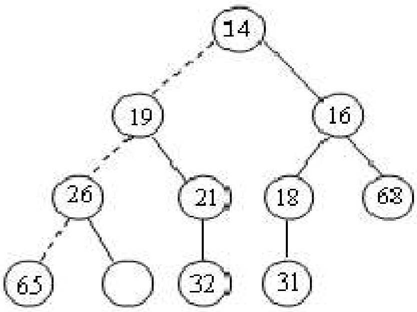
Step2:Rplace the hole by the smallest no 14.



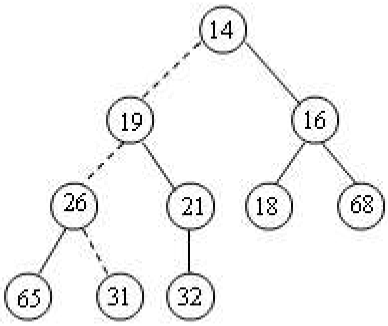
Step 3: Replace the hole by the smallest no 19.



Step4:eplace the hole by the smallest no 26.



Step 5: Replace the hole by the last element 31.



Routine to perform DELETEMIN

```
int Deletemin ( Priority Queue H)
{
int I, child;
int MinElement, LastElement;
if(IsEmpty(H))
{
Error("Priority Queue is Empty ");
Return H ⍰ Elements [0];
}
MinElement = H ⍰ Elements[i];
LastElement = H ⍰ Elements [ H⍰ Size --];
for(i=1; i*2<=H⍰size;i= child)
{
child=i*2;
if(child!= H⍰size && H⍰Elements[child +1]< H⍰Elements [child])
child++;
if(LastElement > H⍰ Elements [child])
H⍰ Elements [i] = H⍰ Elements [child];
else
break;
}
H⍰. Elements [i]= LastElement;
Return MinElement;
}
```

OTHER HEAP OPERATIONS

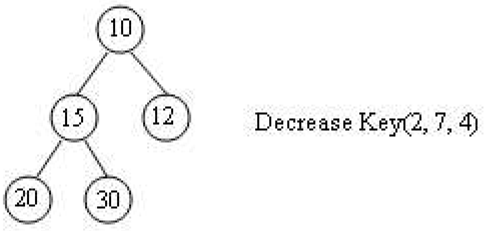
The other heap operations are

- Decrease key
- Increase Key
- Delete
- Bulid Heap

1)Decrease Key

The decrease key(p, ,H). Operation decreases the value of the key at position p by a positive amount .This may violates the heap order property which can be fixed by percolate up.

Example :



Solution:

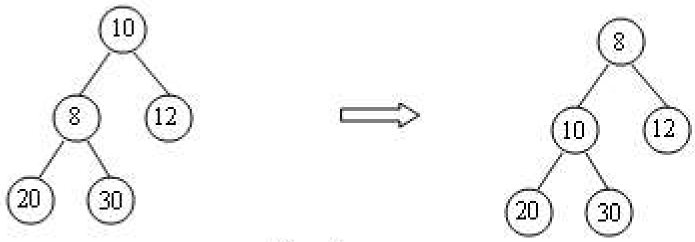


Fig. A

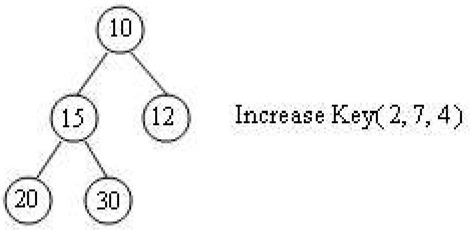
Element at position 2 is 15.decrease that element by 7.Now the position 2 has the value 8.which violates the heap order property .

This can be fixed by percolating up strategy.(Fig A)

2) Increase key

The increase key (p, , H), operation increases the value of the key at position p by a positive amount. This may violate order property.

Example:



Solution:

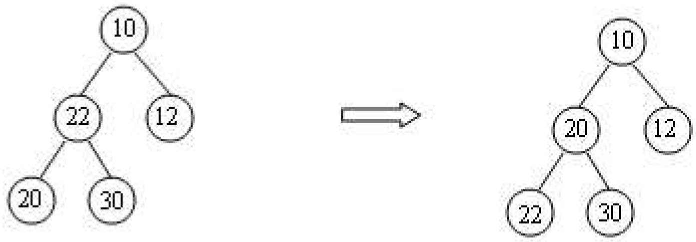


Fig. B

Here the element at position 2 is 15.Increase that value by 7.Now the position 2 has the value 22,which violates the heap order property.This can be fixed by percolate down.

3)Delete:

The delete(p,H) operation removes the node at the position p from the heap.

This can be done by,

(i) Perform the decrease key operation

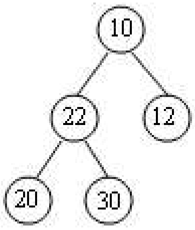
Decrease key(p, ,H)

(ii) Perform DeleteMin operation

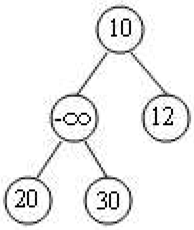
DeleteMin(H)

Example:

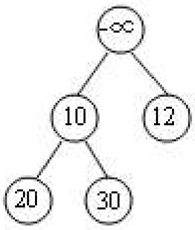
Delete 22 from this heap



(i) Decreasing by Infinity



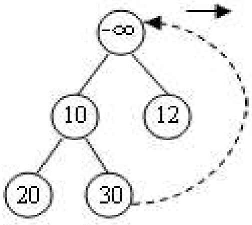
After decreasing the value at position .The value changes to which is the least element in heap.



Since occupies the root position,apply DeleteMin opration.

(ii) DeleteMin

After deleting the minimumelement ,the last element will occupy the hole.Then rearrange the heap till it satisfies heap order property.



Build Heap

The BuildHeap(H) operations takes an input N keys and places them into an empty heap by maintaining structure property and heap order property. This can be done with N successive insertion, since each insert will take O(1) average and o(log N) worst case time.

Application of Priority Queue

- The typical example of priority queue is scheduling the jobs in operating system. Typically operating system allocates priority to jobs.
- The jobs are placed in the queue and position of the job in priority queue determines their priority. In operating system there are three kinds of jobs.
- These are real time jobs, foreground jobs and background jobs. The operating system always schedules the real time jobs first.
- If there is no real time job pending then it schedules foreground jobs. Lastly if no real time or foreground jobs are pending then operating system schedules the background jobs.
- In network communication, to manage limited bandwidth for transmission the priority queue is used.
- In simulation modeling, to manage the discrete events the priority queue is *used.*

2.8 APPLICATIONS OF QUEUES

Q10)a) What are the applications of Queue.

Or

Q10) b) Name some areas where we can use Queue Data Strcuture.

ANSWER:

1. Batch processing in an operating system.
2. To implement priority queues.
3. Priority queues can be used to sort the elements using sort.
4. Simulation.
5. Mathematics user queuing theory.
6. Complete networks where the server takes the job of the client as per the queue strategy.
7. When jobs are submitted to a printer they are arranged in order of arrival .Thus jobs sent to a line printer are placed to a queue.
8. Virtually every lifeline is a queue. For instance lines at ticket counter are queue because service is FCFS.
9. In computer networks there are many network setup of personal .Computers in which the disk is attached to one machine known as the file services .Users on other machines are given access to files on FCFS Basis. So the data structures is queue.
10. The large universities where resources are limited. Students must sign a waiting list if all terminals are occupied. The student who has off first and the student who has been waiting the longest is the next user to be allowed on.
11. Calls to large companies are generally placed on a queue when all operation.