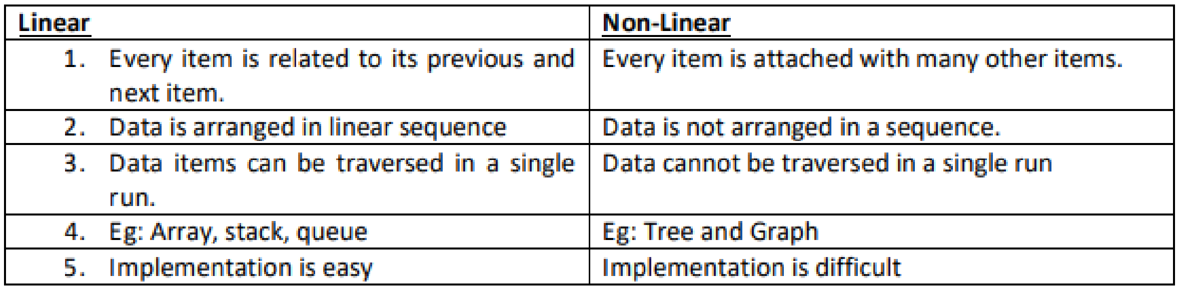
Easy Theory questions – Module I

(Study infix to postfix conversion and postfix evaluation from note book)

1.Difference between linear and non-linear data structures



Linear	**Non-Linear**
1. Every item is related to its previous and next item.	Every item is attached with many other items.
2. Data is arranged in linear sequence	Data is not arranged in a sequence.
3. Data items can be traversed in a single run.	Data cannot be traversed in a single run
4. Eg: Array, stack, queue	Eg: Tree and Graph
5. Implementation is easy	Implementation is difficult

2. Basic operations on all data structures

a) Insertion – adding a new data item
b)deletion - deleting an existing data item
c)searching – for a particular data item
d)traversal – accessing all data items exactly once
e)sorting- arranging data items in an order
f)merging – combining data items in two collections

3. Explain stack

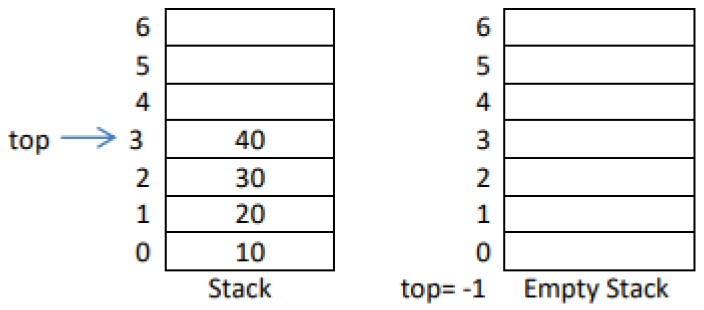
a)A linear DS in which insertion and deletion can be done at ONE END ONLY. This end is called top.

b)Stack is a Last In First Out(LIFO) data structure, means the last inserted element will be first deleted.

4. Array representation of stack

(First explain stack)

```
struct stack
 {
int st[20];
  int top;
}
```



5. Basic operations on stack

a)Push – insertion to stack is called 'push'

b)pop- deleting an element from stack is called 'pop'

c)traverse- accessing all elements in the stack(ex. For displaying)

6. Implementation of stack using array(Algorithm & code)

<u>Push a new data item(Algorithm)</u>

Check If stack is full, that is, if top==max-1.

If yes, Print stack is full

Else, Increment top to point to the next free location

Then insert the new data at location pointed by top.

```
push(data)
{
If(top==max-1)
        printf("stack full/overflow);
        else
        {
        top++;
        st[top]=data;
        }
}
```

<u>Popping a data from stack(Algorithm)</u>

Check if stack is empty. That is, if top==-1

If yes, print stack is empty

Else, print the stack top element and decrement top.

```
pop()
{
        If(top==-1)
Printf("stack is empty");
else
{
X=st[top];
Printf("%d",X);
top--;
}
}
```

Traversal

Check if stack is empty. That is, if top==-1
If yes, print stack is empty
Else,
print the stack elements from 0th to top.

```
traverse()
{
.                   If(top==-1)
          Printf("stack is empty");
          else
          {
          For(i=0;i<top;i++)
          Printf("%d",st[i]);
          }
}
```

To check whether stack is empty
Check if top==-1
If yes, print stack is empty

```
isEmpty()
{
If(top==-1)
Printf("stack is empty");
}
```

To check whether stack is full
Check if top==max-1
If yes, print stack is full

```
          isFull()
{
          If(top==-1)
Printf("stack is full");
}
```

7. Applications of stack

Some applications of stack are:
1. Conversion of an infix expression into its corresponding postfix form.
2. Evaluation of arithmetic expressions.
3. Recursion.
4. Parenthesis checking.
5. Reversing a list
6. Number system conversion (Ex. Decimal to binary, decimal to hex etc)

8. Explain Queue

a)A linear DS in which insertion and deletion can be done at TWO ENDS called REAR and FRONT. Insertion at REAR end and deletion at FRONT end.

b)Queue is a First In First Out(FIFO) data structure, means the firsst inserted element will be first deleted.

9. Basic operations on queue

a)enqueue – insertion to queue is called 'enqueue'

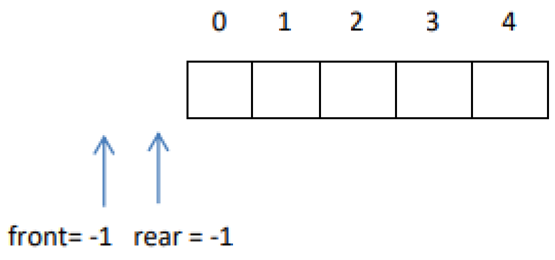
b)dequeue- deleting an element from queue is called 'dequeue'

c)traverse- accessing all elements in the queue(ex. For displaying)

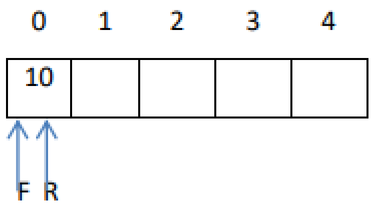
10. Array representation of queue

```
struct queue
 {
 int q[20];
   int rear,front;
}
```

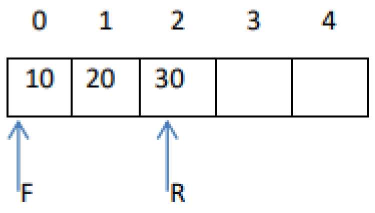
Example of an empty queue



When the first element is inserted,



When remaining elements are inserted,



11. Basic Operations on queue

<u>Insert a new data item(Algorithm)</u>

Check If queue is full, that is, if rear==max-1.

If yes, Print queue is full

Else , check if queue is empty, that is if front==-1

Increment front and rear

Else, Increment rear to point to the next free location

Then insert the new data at location pointed by rear.

```
enqueue(data)
  {
  If(rear==max-1)
          printf("queue full/overflow);
  else if (front==-1)
  {        front++;       rear++;
          q[rear]=data;  }
  else
  {
  rear++;
  q[rear]=data;
  }
}
```

<u>Deleting a data from queue(Algorithm)</u>

Check if queue is empty. That is, if front==-1

If yes, print queue is empty

Else, print the front element of queue and increment front.

```
dequeue()
     {
     If(front==-1)
     Printf("queue is empty");
     else
     {
     X=q[front];
     Printf("%d",X);
     front++;
     }
  }
```

<u>Traversal</u>

Check if queue is empty. That is, if front==-1

If yes, print queue is empty

Else,

print the queue elements from front to rear.

```
  traverse()
{

  If(front==-1)
  Printf("queue is empty");
  else
```

```
        {
        For(i=front;i<rear;i++)
        Printf("%d",q[i]);
        }
    }
```

<u>To check whether queue is empty</u>
Check if front==-1 or front>rear
If yes, print stack is empty

```
isEmpty()
{
If((front==-1) || (front>rear))
Printf("queue is empty");
}
```

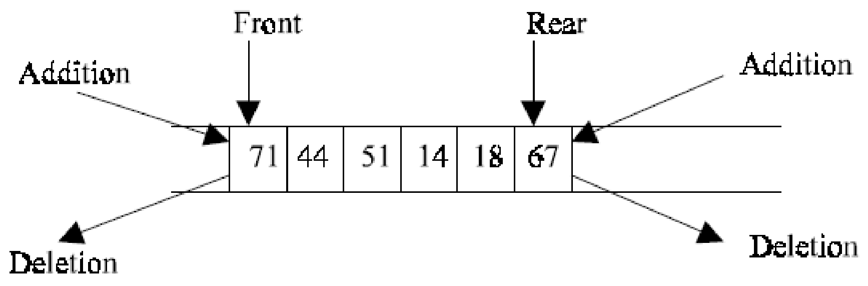
<u>To check whether queue is full</u>
Check if rear==max-1
If yes, print queue is full

```
isFull()
{
If(rear==-1)
Printf("queue is full");
}
```

12. Types of queue

DOUBLE ENDED QUEUE (DEQUEUE)

In double ended queue insertions and deletions can be done from both the ends.



There are five operations in double ended queue.

1. Insertion at rear end.
2. Insertion at front end.

3. Deletion from rear end.
4. Deletion from front end.
5. Traversal.

PRIORITY QUEUE
a)Priority queue is a variation of simple queue
b)In Priority queue each element has a value associated with it called its priority.
c)When deletion operation is performed we remove the element with highest priority.
d)In a priority queue the elements are stored in the ascending order of priority value so that the element with highest priority will always be at the front position. So elements are inserted into the queue in the order of priority.

	0	1	2	3	4
Value →	10	20	30	40	50
Priority →	1	5	7	8	9

CIRCULAR QUEUE
Consider the following case in a simple queue of maximum size 5.

0	1	2	3	4
		30	40	50
		F		R

Here Front points to the index 2 and rear points to 4 ie MAXSIZE-1. If we try to insert a new element into the above queue, it will not be possible because rear=MAXSIZE-1 eventhough two vacant positions are there at front. This is the limitation of a simple queue. To avoid such limitation we can use circular queue. In this queue the rear is incremented in such a way that after rear points to MAXSIZE-1 it will point to 0 if that position is empty. In this case the queue becomes full only if rear+1 points to front.

The pictorial representation of a circular queue is:

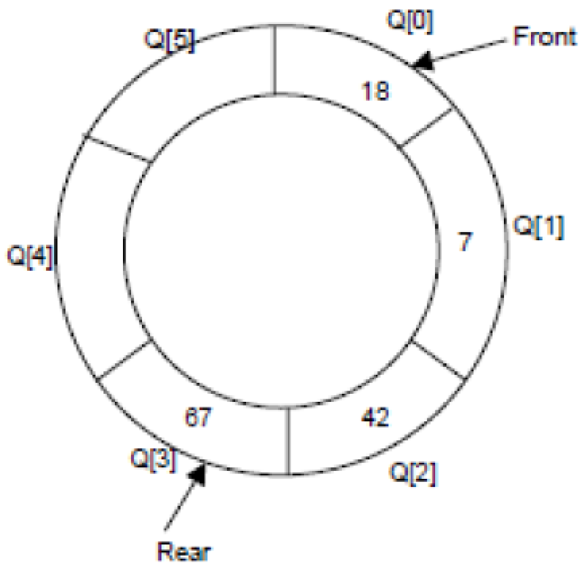


Fig. 4.11. A circular queue after inserting 18, 7, 42, 67.

After deleting 18 and 7, the queue becomes:

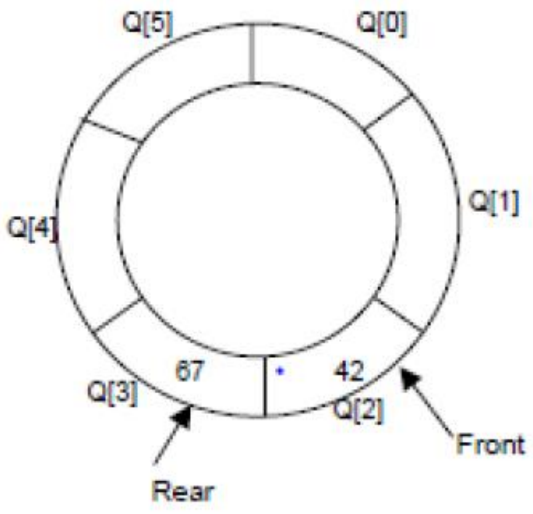


Fig. 4.12. A circular queue after popping 18, 7

Now we can insert 4 more elements into the above queue. But in a simple queue we can insert only 2 more elements.
After inserting three elements 30, 47 and 14 queue becomes

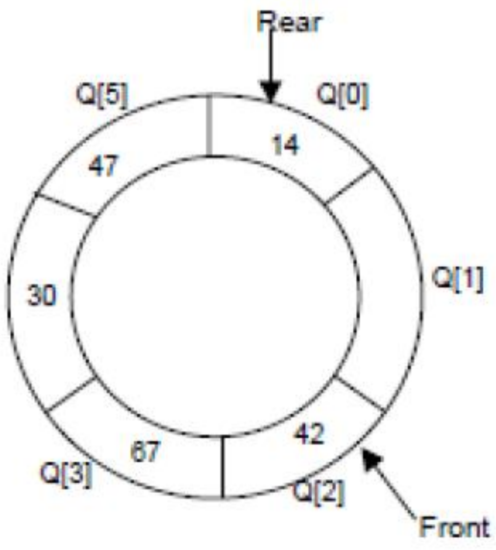


Fig. 4.13. A circular queue after pushing 30, 47, 14