Linear

Queue:-

Store elements in particular order

FIFO (first in first out).

It follows linear data structures in which we can add and remove elements.

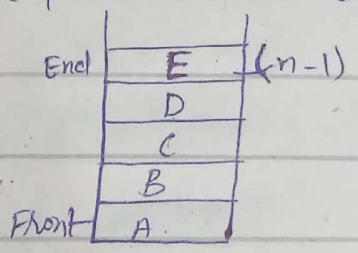
i- Enqueue
ii- Dequeue.
iii- Empty

} Primitive operation.

* Pointers

i- Front (~~Points 1st element~~) deletion
ii- Rear (~~Points last element~~) Insertion

"ABCDE"



algo.

```
Class Queue {
    Private int front, rear;
    Private int queue[];
    Private int maxsize;
Queue (int max Queue siz) {
    queue = new int [maxsize];
```

```
maxsize = maxqueuesize;
    front = rear = -1;
    }
boolean IsFull() {
if( rear == maxsize-1)
        return true;
else
        return false
        }
Bodean IsEmpty(){
  if (front == rear)
      return true;
else
      return false; }
Public void add (int x){
      if (IsFull()){
 S.O.P("Queue overflow");
    System.exit(-1);
      }
    queue[++rear] = x;
      }
int delete(){
    if(IsEmpty()){
```

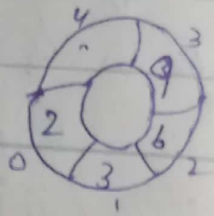
```
S.O.P("underflow");
    System exit(-1);
int x = queue[++front];
    return x;
    }
```

Circular queue.

```
Class Queue {
Private int queue[];
private int front, rear, maxsize;
  Queue(int size){
    queue = new int[Maxsize];
    maxsize = size;
front = rear = Maxsize-1;
}
 void Add(int x){
int k = (rear+1) % maxsize;
if (front == k){
  S.O.P("overflow");
    System.exit(1); }
else { queue[k] = x;
      rear = k; } }
```

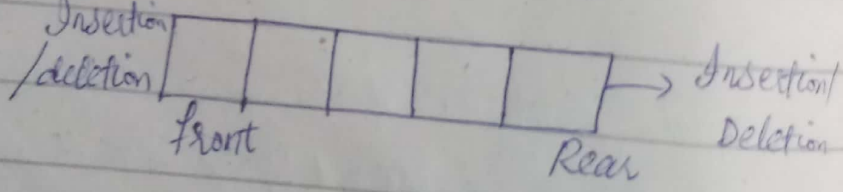
```
int Delete(){
if (front == rear){
 S.O.p("underflow");
   System.exit(1); }
 front =(front+1) % Maxsize;

   x = queue[front];
   return x;
        }
```



De-queue.

(double-ended queue)



Operations

i- Insert at front

ii- Insert at rear

iii- Delete from front

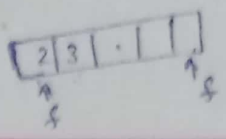
iv- Delete from rear

~~enqueue~~
algo.

```
Class Queue {
Private int queue[];
Private int front, rear, maxsize;
   Queue(int size) {
    queue = new int [maxsize]
       front = rear = -1;
    enqueuefront (int x) {
    if (front == 0 && rear == size-1) || (front ==
       S.O.P ("overflow");                    rear+1) {
    System exit(-1);
              }
Elseif (front == -1 && rear == -1) {
       front = rear = 0;
        queue [front] = x; }
elseif (front == 0) {
            front = size-1;
     queue [front] = x; }
   else {
          front--;
    queue [front] = x; } }
```

```
Void enqueue(int x) rear{
if (front==0 && rear==size-1)||(front==rear+1){
        S.o.p ("overflow");
            System.exit(-1); }
    elseif (front==-1 && rear==-1){
front=rear=0;
queue[rear]=x; }
        elseif (rear==size-1){
            rear=0;
        queue[rear]=x; }
    else{
        rear++;
    queue.[rear]=x;
            }

Void dequeue(){
if (front==-1 && rear==-1){
    S.o.p("underflow");
        System.exit(1); }
elseif (front==rear){
    S.o.p(queue[front]);
front=rear=-1;
        }
```



```
elseif (front == size-1){
        S.O.P (queue [front])
                front = 0 ; }
else{ S.O.P(queue[front]) ;
                front++; }}
Void dequeueRear() {
    if (front == -1 && rear == -1){
     S.O.P ("underflow");
          System.exit(1) ; }
elseif (front == rear) {
     queue [rear] = x;   S.O.P(x);
        front = rear-1 ;
            }
elseif ( rear == 0){
        S.O.P (queue [rear]);
          rear = size-1;
            }
    else {
      S.O.P ( queue [rear]) ;
          rear --;
              }}
```

* Priority queue:

i- Ascending queue (min priority)
ii- Descending queue. (max priority)

* ways of Implementation -

Two ways

(1) i- Normal insertion -> special priority deletion
ii- Special deletion operation
∴ (elements with high priority will be popped out first)

(2) i- Special insertion (elements with high priority will be insert first)
ii- Normal deletion

Algo.

Special insertion / normal deletion.

```
void enqueue(int item){
if (n == maxsize-8){
        S.O.P("Queue overflow");
        System.exit(-1); }
int i = n-1;
while (i>=0 && itemqueue[i]){
        queue[i+1] = queue[i];
        i--; }
```

```
queue[i+1] = item;
        n++; }
int dequeue() {
    int item;
    if (n==-0) {
     S.o.p("underflow");
      system.exit(-1);
                }
item = queue[n-1];
    n = n-1;
    return item;
            }
```

Book 2 (Chptr 6) 117 Pg.