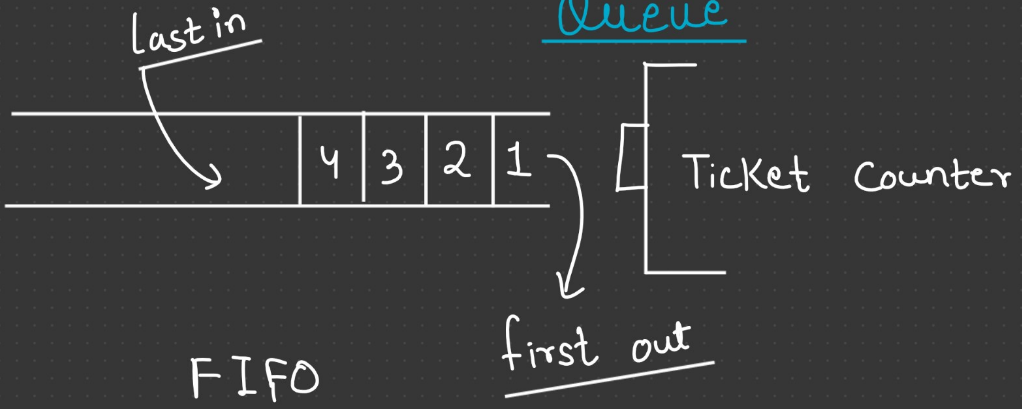
Queue



FIFO

1) Insert → Enqueue

2) Delete → Dequeue

STL

1) Array ✓ →

2) Dynamic Array ✓

3) Linked List ✓

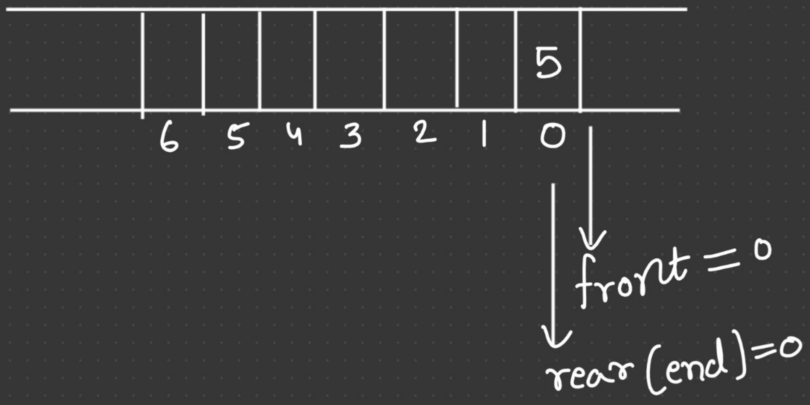
} ✓ { → insert at end() → enqueue

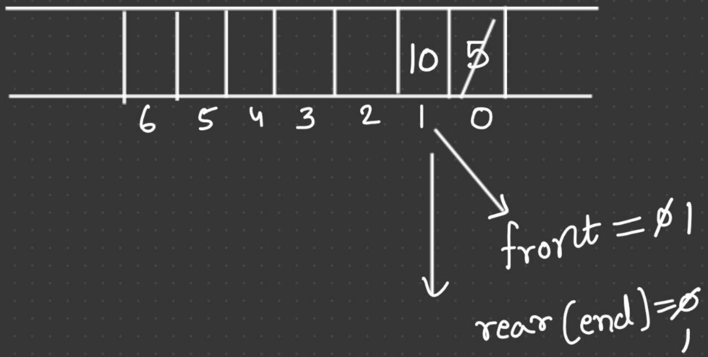
→ Deletion at begin() → Dequeue.

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

```
Class Queue
{
 public:
   int q[100];
   int front;
   int rear;
   int size;
};
```

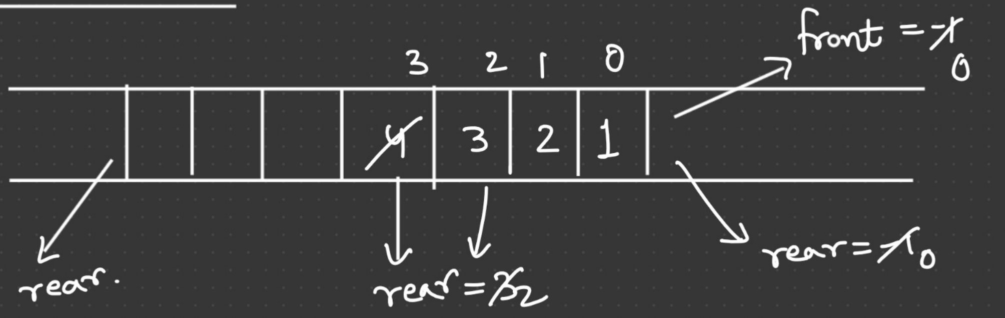
```
Queue qu;
1) void Enqueue(int v)
   {
      if(rear == -1)
      {
           rear++; front++;
           q[rear] = v;
      }
   else if( rear == Size-1)
           cout <<"Overflow";
```





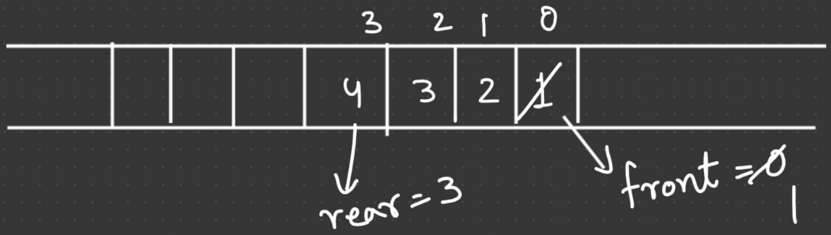
```
    else
     {
        rear++;
        q[rear]=v;
      }
   }

2)  void Dequeue()
     {
        if (front ==-1)
          cout<<"underflow";
       else if ( rear == front)
         {
            cout<<q[front]<<" deleted";
            front = rear = -1;
         }
       else{
           cout<<q[front]<<" deleted";
           front++.
          }
     }

   void print()
  {
     for(i= front ; i <= rear ; i++)
          cout << q[i]<"  ";
   }
```

Double Ended Queue :-



```
1)  void insertAtEnd(int v)
    {
        if(rear == -1)
        {
            front++ ; rear++;                  ⟶ O(1)
            q[rear] = v;
        }
        else if ( rear == size-1)
        {                                      ⟶ Drawback
            cout << "overflow";

        }
        else {      q[++rear] = v;

                }
    }                  0, 1, 2 . . . . . 7, 8
```

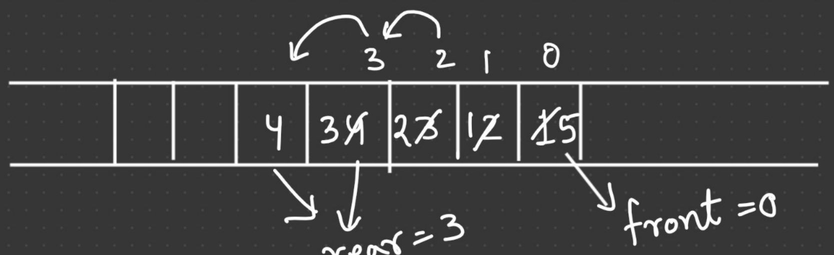
```
2)  void deletionAtEnd( )
    {
        if (rear == -1)
        {
            cout<<"underflow";                 ⟶ O(1)
        }
        else if ( rear == front)
        {
            cout << q[rear]<" deleted";
            front = rear = -1;
        }
        else{       cout<< q[rear--]<<"deleted";

                }
    }
```



```
3) void deleteAtBegin()
   {
      if(front==-1)
         cout<<"underflow";
     else if (rear==front)
     { cout<<q[front]<<" deleted";
         front =rear=-1;
     }
      else
      { Cout<<q[front++]<<" deleted";
      }
   }
```

→ O(1)

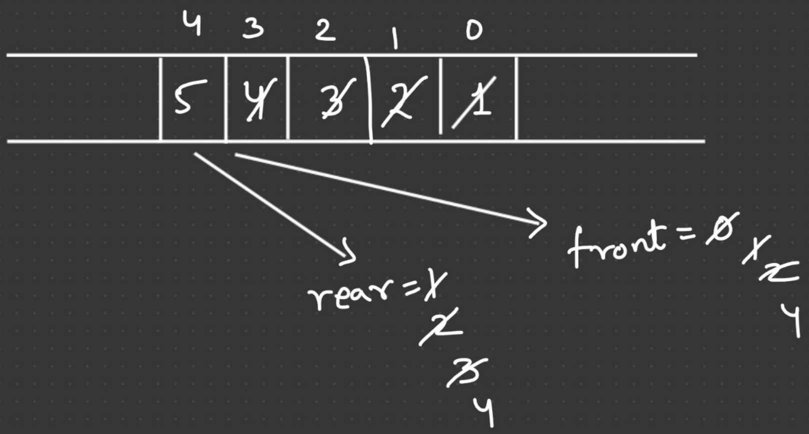
```
4) void insertAtBegin(int v)
   {
      if(front== -1)
      {
         front++; rear++;
         q[front]=v;
      }else if(rear ==size-1)
           cout<<" overflow";
      else{  for(int i=rear;i>=front;i--)
             {
                q[i+1]=q[i];
             }
              rear++;
              q[front]=v;
          }
   }
```

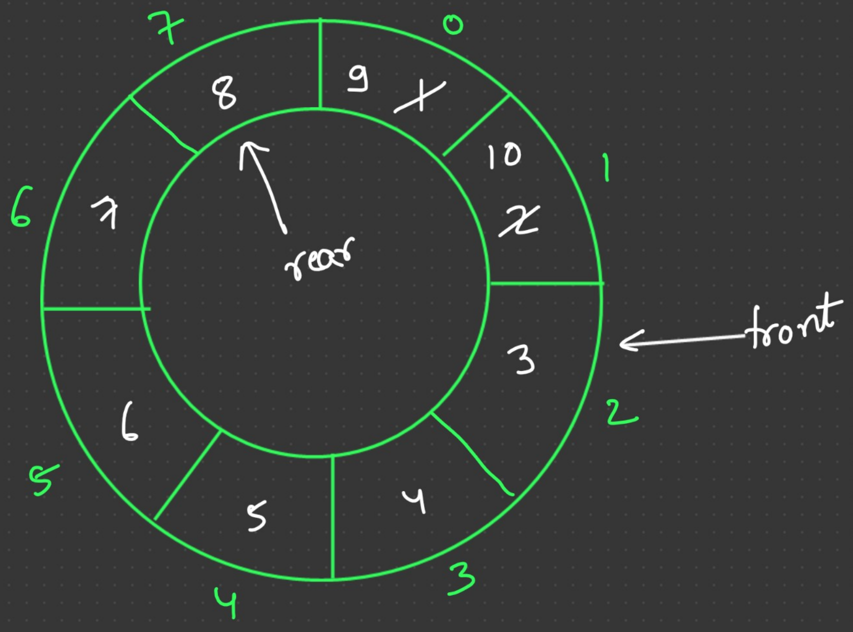


→ O(n)

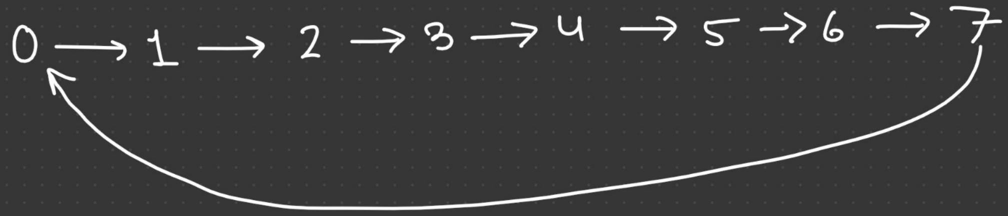
Circular Queue :—

Problem ? →





```
while (rear+1 != front)
{
    insertion Allow
}
```



0 % 5 = 0

1 % 5 = 1

2 % 5 = 2

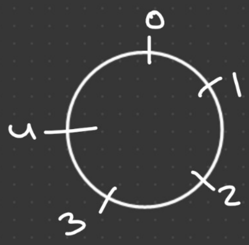
3 % 5 = 3

4 % 5 = 4

5 % 5 = 0

6 % 5 = 1

Divide 5 → Remainder ⇒ [0, 1, 2, 3, 4]



(rear+1) % 8 ⇒ [0 – 7]

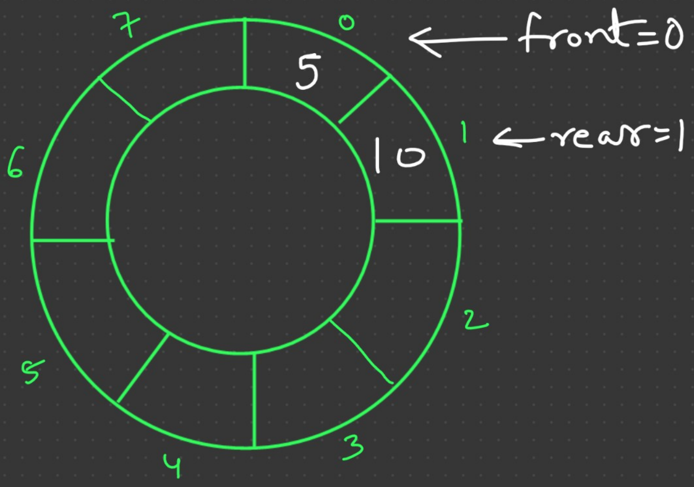
Concept

$$\boxed{rear = (rear+1) \% size}$$

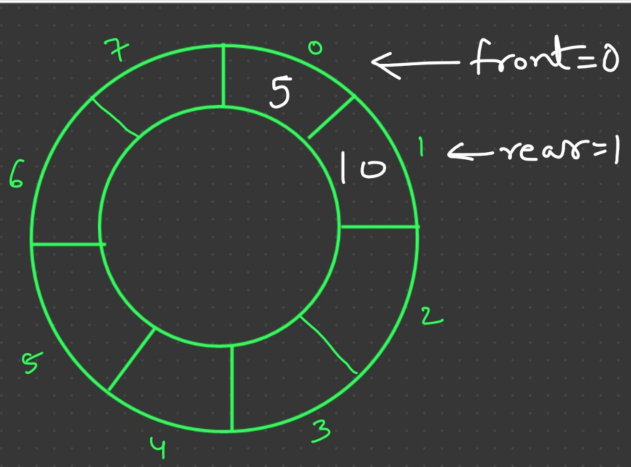
```
void Enque (int v)
{
    if (rear == -1)
    {
        rear++;
        front++;
        q[rear]=v;

    }
    else if ((rear+1)%size == front)
     {
      cout<<"overflow";
      }
    else {  rear = (rear+1) % size;

}      }  q[ rear ]=v;
```

(1+1)%8

$\boxed{2}$

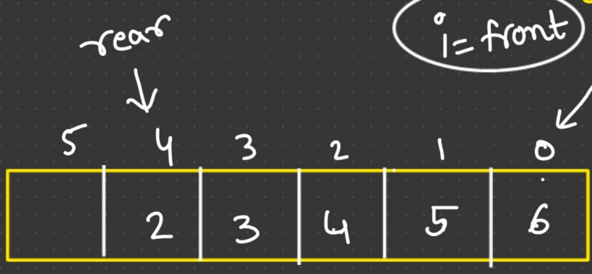


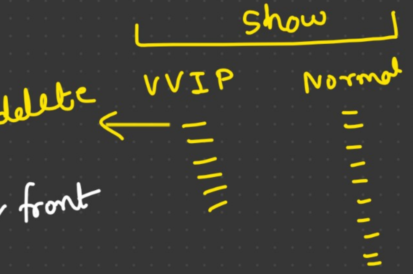
```
void  Dequeue()
 {
    if(front == -1)
   {
     Cout<<"underflow";

   }else if ( rear ==front)
    {
       cout<<q[front]<<" deleted";
        rear= front =-1;
     }
     else{
           cout<< q[front]<<"deleted";
              front  =  (front+1)%size;
          }

  }
```



Enqueue() → O(1)
Dequeue() → O(1)

Priority Queue:-





2, 4, 6, 3, 5, 8

```
Enqueue(int v)
{
    if (front == -1)
        q[front] = v;
```

PQ ⟶

Enqueue ( ) ⟶ o(n)

Dequeue( ) ⟶ O(1)

```
for (i = front ; i <= rear ; i++)
{
    if ( v > q[i])
        break;
}
for(j=i; i <= rear ; i++)
{   q[i+1] = q[i];

}
    q[j]=v;
```