Priority Queue :

Priority Queue is a queue in which every element of queue has some priority and based on that priority it will be processed. So the element of more priority will be processed before the element which has less priority.

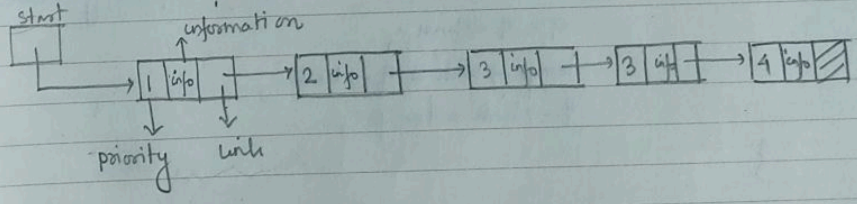
If two elements have same priority then in this case FIFO rule will follow, means the element which comes first in the queue will be processed first.

Linked list implementation of Priority Queue.

To define the structure of elements which will be used for priority queue -

```
struct pq {
        int priority;
        int data;
        struct pq * link;
}
```

Here first member of structure is the priority for that element, second has information and third is link which has address of next element.



Here, priority 1 means the highest priority. If priority of an element is 2 then it means it has priority more than the element which has priority 3

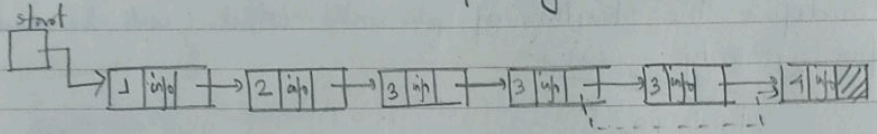
DATE

Operations in Priority Queue:

① Add
② Delete

Add operation in Priority Queue:

Here we insert the new element on the basis of priority of element. The new element will be inserted before the element which has less priority than new element.

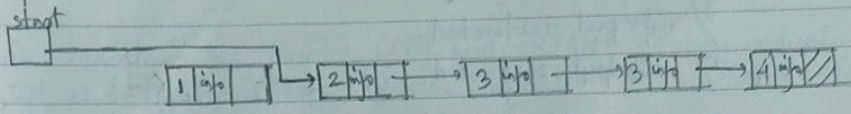



```
if (front == NULL || item_priority < front -> priority)
    {   temp -> link = front;
        front = temp;
    }
else {  q = start;
        while (q -> link != NULL && q -> link -> priority <= item_priority)
            q = q -> link;
        temp -> link = q -> link;
        q -> link = temp;
    }
```

DATE

Delete Operation in Priority Queue :

Delete operation will be the deletion of first element of list because it has more priority than other elements of queue.




```
if (front == NULL)
        printf ("Queue Underflow\n");
else
  {
        temp = front;
        printf ("Deleted item is: %d\n", temp->info);
        front = front->link;
        free(temp);
  }
```

DATE

Dequeue (Double Ended Queue)

The Dequeue is a linear list where we can add or delete elements on both sides.
Dequeue can be of two types:

1) Input restricted.
2) Output restricted.

1) Input restricted:

Here, element can be added on one end but we can delete elements from both sides.

2) Output restricted:

Here, element can be added at both side but deletion is allowed at only one end.

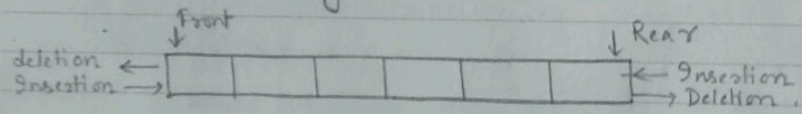


fig: Representation of Dequeue

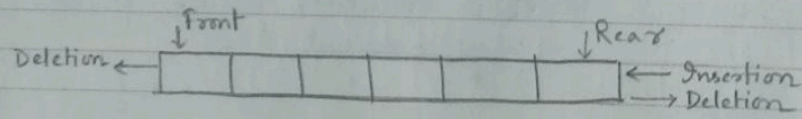


fig: Representation of Input restricted dequeue.

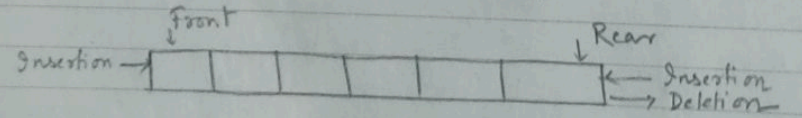
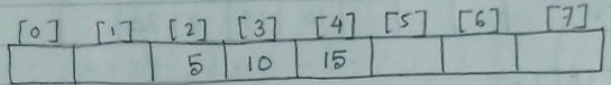


fig: Representation of Output restricted dequeue.

Array Implementation of dequeue:

For implementing dequeue we are taking array.
Dequeue has two operations : i) Add.
ii) Delete.

We maintain two pointers LEFT and RIGHT which indicate left and right positions of the dequeue

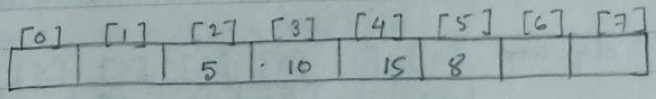


LEFT = 2
RIGHT = 4

Add and delete operation in Dequeue

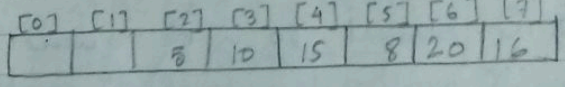
We assume this is circular array for operation of addition and deletion. Let us take it's an input restricted dequeue and we can add element only on the right side of queue but we can delete from both sides.

1. Add the element 8 in the queue



LEFT = 2
RIGHT = 5

2. Add the element 20, 16 in the queue



LEFT = 2
RIGHT = 7

DATE

3 Delete the element from left of the queue

dq_arr

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
			10	15	8	20	16

LEFT=3
RIGHT=7

4. Delete the element right of the queue.

dq_arr

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
			10	15	8	20	

LEFT=3
RIGHT=6.

5 Add the element 35 and 12 in the queue.

dq_arr

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
12			10	15	8	20	35

RIGHT = 0
LEFT = 3.

6. Add the element 25, 50 in the queue.

dq_arr

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
12	25	50	10	15	8	20	35

LEFT = 3
RIGHT = 2.

7 Add the element 60 in the queue.

Now it's overflow because RIGHT pointer will become equal to LEFT pointer after adding one element.