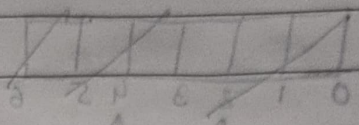
Date
Page 67

Queue

→ Work on FIFO (First in First Out)

- Insert from rare end and delete from front end



Queue ADT

Data:

1. Space for storing element.
2) Front → for deletion
3) Rear → for Insertion.

Operation:

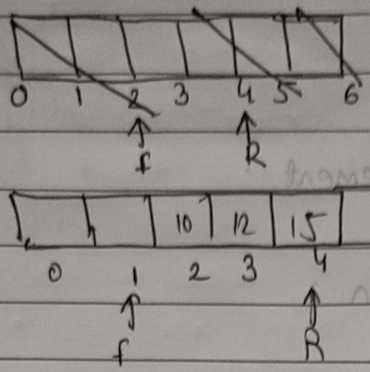
enqueue()
dequeue()
isEmpty()
isFull()
first()
last()

- Queue can be implemented using
  → Array
  → Linked List

- Queue using one pointer.
  → Insertion will take order of 1 time
  → Deletion will take n time as the rest of the element needs to be shifted as deletion of any element will make vacant space

- Queue using 2 pointer.

→ Initially Front = Rear = -1



enqueue - O(1)
dequeue - O(1)

Empty:
if (front == Rear)

→ Here insert using Rear pointer
→ Delete using front Pointer (i.e delete element of address at which it stands)

- Struct Queue {
  int size;
  int front;
  int Rear;
  int *q;
  }

Date
Page 68

```
* Struct Queue
{
    int size;
    int front;
    int Rear;
    int *Q;
};

int main()
{
    Struct Queue q;
    printf("Enter size");
    Scanf("%d",&q.size)
    q.Q = (int*) malloc (q.size* sizeof (int))
    q.front = q.Rear = -1
}

Void enqueue (Queue *q , int x)
    if (q.Rear == q.size-1)
        printf ("Queue is full");
    else
        q.Rear++;
        q.Q[q->Rear] = x;

void dequeue (Queue*q)   //deque.
  { int x=-1;
    if (q->front == q->Rear)
        printf("Queue is Empty");
    else
    {   q->front++;
        x = q-> Q[q->front];
    }
    return x; }
```

* Drawback

→ Deleted spaces cannot be reused.

Solutions

1) Resetting pointer.

```
if (Front == Rear)
    front = Rear = -1;
```

2) Circular Queue.

- Wherever front is pointing should be empty.

→ using mod value can be inserted.

Rear = (Rear+1) % size

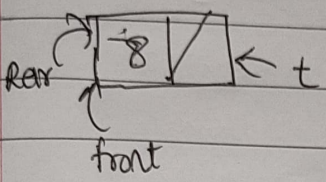
0	(0+1) % 7	1 → Next index of Rear
1	2 % 7	2
2		
3		
4		
5		
6	7 % 7	0

* Queue using Linked list

→ For Empty:
if (front == NULL)

→ Full
Node *t = New Node
if (t == Null)

→ for one node



* Double ended Queue.

↳ Insertion as well as deletion from both front & rear is possible

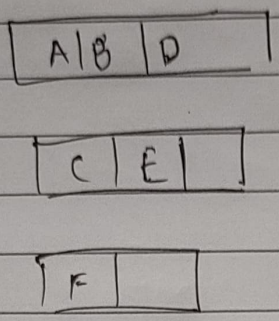
	Insert	Delete
front	✓	✓
Rear	✓	✓

* Priority Queue.

→

Element	A	B	C	D	E	F
Priority	1	1	2	1	2	3

Priority Queue



→ First this will be pop out completely.

* when ~~priority is~~ element itself is priority.

element → 8, 8, 3, 5, 1, 3, 2, 1

Methods to solve such issues of Queue

1) Insert in same order.
   Delete Max priority by searching it.

2) Insert in increasing order of Priority.
   Delelete last elenet of array

* Method 1 for priority queue

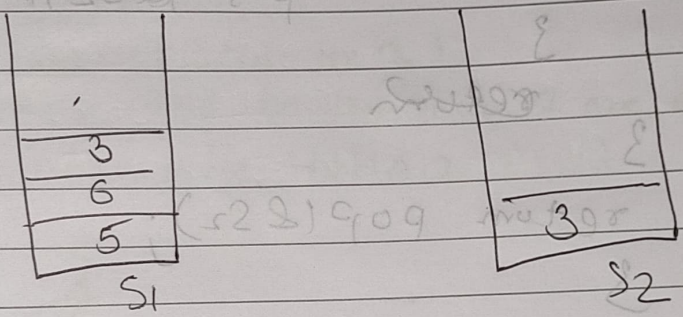
Search → O(n)
Shift → O(n)
O(2n) = O(n)

* Method 2

Insert → O(n)
Delete → O(1)

* Implementation of Queue using 2 stack

- To delete 5 shift 3, 6 from S_1 to S_2



Logic

→ If S_2 is empty transfer element from S_1 to S_2 and then delete S_2 element

→ So it will act as Queue instead of stack

→ If element are there in S_2 stack delete normally

```
enqueue (int x)
{ push (&S1, x);

int dequeue ()
{
    int x=-1
    if (is Empty (S2))
    {   if (isEmpty(S1))
        {   print(Queue Empty)
            return x;
        }
        else
        {
            while (isEmpty (S1))
                push(&S2, pop(&S1));
        }
        return
    }
    return pop(&S2);
}
```