Date _____ Page _____

Day - 143

Queue - 2

* **Print all element in queue:**

2	4	6	3	5

⇒ We have print all the elements.
⇒ So, first we will print the front element then pop it.
⇒ We will do this until queue becomes empty.

Code

```
while (!q.empty()){
    cout << q.front();
    q.pop();
}
```

⇒ Now, if we want that the elements will not pop from the queue then —
⇒ Then we will store & print the elements.
⇒ After that, we will push all the elements again.

Code

```
vector<int> ans;
while(!q.empty()){
    cout<< q.front();
    ans.push_back(q.front);
    q.pop();
}
```

Date Page

```
for(int i=0; i<ans.size(); i++){
    q.push(ans[i]);
}
```

⇉ Now, if we want to solve it in O(1) S.C. then-
⇉ we can take a size variable.
⇉ After that, when we print the first element then we will push it into the queue at the same time.
⇉ And decrease the size by 1.

Code

```
int n = q.size();
while(n--){
    cout<< q.front();
    q.push(q.front());
    q.pop();
}
```

* Queue Reversal:

⇉

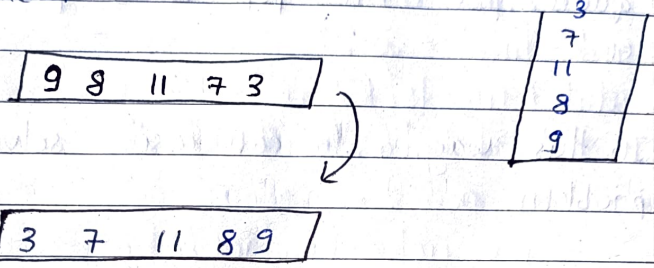
3	7	11	8	9

⇉ In brute force approach, we can pop all the elements & store in an array.
⇉ After that push all the elements into the queue from the end of array.

Date _____ Page _____

⇒ So, this is not the optimized approach.

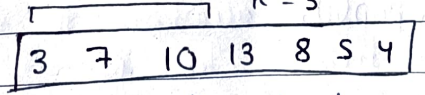
⇒ For more optimized approach, we can use stack.



Code

⇒
```
stack<int> st;
while(!q.empty()){
    st.push(q.front());
    q.pop();
}
while(!st.empty()){
    q.push(st.top());
    st.pop();
}
```

* Reverse First k elements of queue:



⇒ First we store the k elements into the stack.

⇒ After that, we will store the no. of remaining elements in queue.

Date Page

⇒ Now, we will push the stack elements in queue.

⇒ Now, the no. of remaining elements in queue, we will pop it & push into the end.

⇒ In this way, we can easily solve our problem.

Code

```
stack<int> st;
while(k--){
    st.push(q.front());
    q.pop();
}
int n = q.size();
while(!st.empty()){
        q.push(st.top());
        ~~q.pop~~ st.pop();
}
while(n--){
    q.push(q.front());
    q.pop();
}
```

* Time needed to buy Tickets:

Date _____ Page _____

	0	1	2	3	4
Tickets →	1	5	2	3	7

⇒ According to question, we have a tickets array in which every value represents that the no. of tickets that person wants.

⇒ Also, any person can take only 1 ticket at a time.

⇒ If ~~they~~ he want more tickets then he has to stand in the line again for tickets.

⇒ Also, 1 ticket takes 1sec.

⇒ We also given a 'k' value, so we have to return the time taken or required for k^{th} person to buy all their tickets.

⇒ As, we are seeing that a person will come and then go to back of the line.

⇒ So, here we will use queue DS.

⇒ We will store index of elements in queue.

⇒ If that person wants ticket we will increase value of time & push it again back to the queue.

⇒ And when the value of tickets of k^{th} person becomes 0 return the value of time.

Date Page

Code

```
queue<int> q;
for(i=0; i<n; i++){
    q.push(i); }
int time = 0;
while (Tickets[k]!=0){
    Tickets[q.front()]--;
    q.pop();
    if( Tickets[q.front()])
        q.push(q.front());
    q.pop();
    time++;
}
return time;
```

⇒ Worst case T.C. of above approach can be — $O(n^2)$.

⇒ When the tickets array like —

tickets →

5	5	5	5	5

⇒ So, to optimize this approach, we will calculate time according to that required person time.

0	1	(2)	3	4
2	6	4	3	7

⇒

Date _____ Page _____

for 0 → k
Time += min(Ticket[k], Ticket[i]);
for k+1 → n
Time += min(Ticket[k] - 1, Ticket[i]);

* Implement queue using stack!

⇒ Push

When we push elements into the queue, we will push it into the stack.

⇒ Pop

⇒ When we pop elements then we will use another stack and push all the elements of stack 1 into stack 2.

⇒ Then pop the top element & return it.

So,
for push → stack 1
for pop → stack 2.

Code

```
class Queue {
    stack<int> st1;
    stack<int> st2;
    public:
    bool empty(){
        return st1.empty() && st2.empty();
    }
```

Date _____ Page _____

```
void push(int x){
    st1.push(x);
}

int pop(){
    if(empty())
        return 0;
    if(!st2.empty()){
        int element = st2.top();
        st2.pop();
        return element;
    }
    else{
        while(!st1.empty()){
            st2.push(st1.top());
            st1.pop();
        }
        int element = st2.top();
        st2.pop();
        return element;
    }
    int peek(){
        if(empty())
            return 0;
        if(!st2.empty())
            return st2.top();
        else{
            while(!st1.empty(){
```

Date _____ Page _____

```
        st2.push(st1.top());
        st1.pop();
    }
    return st2.top();
}
```

T.C. → O(1)

* Implement stack using queue

Push

⇒ when we push elements into the stack, we will push in that queue that have elements.

Pop

⇒ we will first push all the elements without the last element into the second queue.

⇒ Now, we will pop the element from the first queue.

Code

```
class stack{
    queue<int> q1;
    queue<int> q2;
    public:
    bool empty(){
        return q1.empty && q2.empty();
    }
```

Date Page

```
void push( int x){
    if( empty() )
        q1. push(x);
    else if (q1. empty ())
        q2. push (x);
    else
        q1. push (x);
}

void int pop(){
    if (empty())
        return 0;
    else if(q1. empty()){
        while (q2. size()>1){
            q1. push (q2. front());
            q2. pop();
        }
        int ele = q2. front();
        q2. pop();
        return element;
    }
    else{
        while( q2. size()>1 ){
            q2. push( q2. front()){
            q1. ~~q2. pus~~ pop();
        }
        int ele = q1. front();
        q1. pop();
    } } return ele;
```