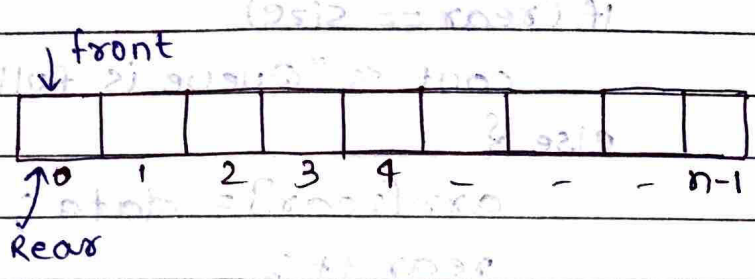
Queue

-By Nikhil Kumar

https://www.linkedin.com/in/nikhilkumar0609/

QUEUE

Q Implement a Queue



→ if (front == rear) → Queue is empty

→ if (rear == size) → Queue is full.

Program :-

```
class Queue {
    int* arr;
    int qfront;
    int rear;
    int size;
public:
    Queue () {
        size = 100001;
        arr = new int[size];
        qfront = 0;
        rear = 0;
    }

    bool isEmpty () {
        if (qfront == rear)
            return true;
        else
            return false;
    }
```

```
void enqueue (int data){
    if (rear == size)
        cout << "Queue is full" << endl;
    else {
        arr[rear] = data;
        rear ++;
    }
};
int dequeue (){
    if (qfront == rear)
        return -1;
    else {
        int ans = arr[qfront];
        arr[qfront] = -1;
        qfront ++;
        if (qfront == rear) {
            qfront = 0;
            rear = 0;
        }
        return ans;
    }
};
int front(){
    if (qfront == rear)
        return -1;
    else
        return arr[qfront];
};
```

→ T.C = O(1)

Q Implement Circular Queue

```
class CircularQueue {
    int* arr;
    int* front;
    int rear;
    int size;

public:

CircularQueue(int n){
    size = n;
    arr = new int[size];
    front = rear = -1;
}

bool enqueue(int value){
    // to check whether queue is full
    if((front==0 && rear == size-1)||(rear== (front-1)%(size-1)))
        return false;
    else if(front == -1) // first element to push
        front = rear = 0;
    else if(rear == size-1 && front != 0)
        rear=0; // to maintain cyclic nature
    else
        rear++; //normal flow
    // push inside the queue
    arr[rear] = value;
    return true;
}
```

```
int dequeue(){
    //to check queue is empty
    if (front == -1)
        return -1;

    int ans = arr[front];
    arr[front] = -1;


    if (front == rear) //single element is present
        front = rear=-1;

    else if (front == size-1)
        front = 0; //to maintain cyclic nature

    else
        front++; //normal flow.

    return ans;
}
};
```

Implement Deque

```
class Deque{
    int* arr;
    int front;
    int rear;
    int size;

public:
    Deque(int n){
        size = n;
        arr = new int[n];
        front = rear = -1;
    }

    bool isEmpty(){
        if(front == -1)
            return true;
        else
            return false;
    }

    bool isfull(){
        if((front == 0 && rear == size-1) ||
           (front != 0 && rear == (front-1) % (size-1))) {

            return true;
        else
            return false;
    }
```

```
int getFront () {
    if (isEmpty ()) {
        return -1;
    }
    return arr [front];
}

int getRear () {
    if (isEmpty ()) {
        return -1;
    }
    return arr [rear];
}

bool pushFront (int x) {
    if (isFull ()) {
        return false;
    }
    else if (isEmpty ()) {
        front = rear = 0;
    }
    else if (front == 0 && rear != size-1) {
        front = size-1;
    }
    else
        front--;

    arr [front] = x;
    return true;
}
```

```
bool pushRear (int x) {
    if (isFull()) {
        return false;
    }
    else if (isEmpty()) {
        front = rear = 0;
    }
    else if (rear == size-1 && front != 0) {
        rear = 0;
    }
    else
        rear++;
    arr[rear] = x;
    return true;
}

int popFront () {
    if (isEmpty())
        return -1;
    int ans = arr[front];
    arr[front] = -1;

    if (front == rear) //single element is present
        front = rear = -1;
    else if (front == size-1)
        front = 0; //to maintain cyclic nature
    else //normal flow
        front ++;

    return ans;
}
```

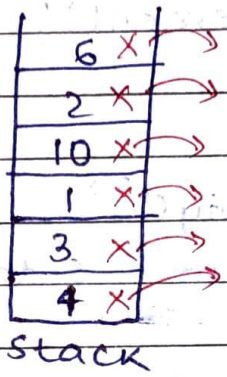
```
int popRear(){
    if (isEmpty())
        return -1;

    int ans = arr[rear];
    arr[rear] = -1;

    if (front == rear) //single element is present
        front = rear = -1;

    else if (rear == 0)
        rear = size-1; //to maintain cyclic nature

    else
        rear--;

    return ans;
}
};
```

Reverse Queue

i/p :→ [4, 3, 1, 10, 2, 6]
o/p :→ [6, 2, 10, 1, 3, 4]

Approach 1 :→ (use stack)

Step 1 :→ Queue se 1 by 1 element nikalo & stack mein daalo.



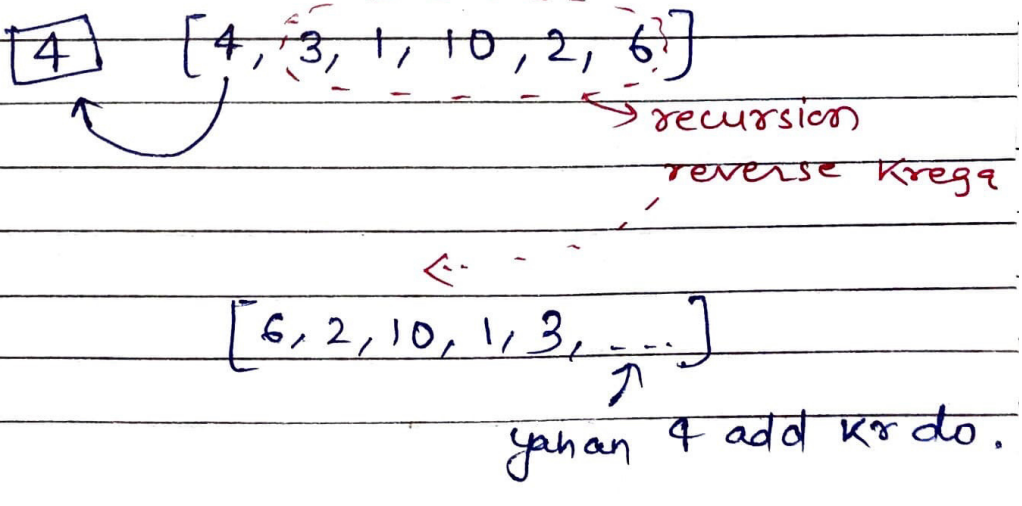
[6, 2, 10, 1, 3, 4]

Step 2 :→ Stack se 1 by 1 element nikalo & Queue mein daalo.

T.C → O(n), S.C → O(n)

Approach 2 :→ (using Recursion)

[4, 3, 1, 10, 2, 6]



Program (Approach 1):-

```
queue<int> rev(queue<int>q) {
    Stack<int> s;

    while (!q.empty()) {
        int element = q.front();
        q.pop();
        s.push(element);
    }

    while (!s.empty()) {
        int element = s.top();
        s.pop();
        q.push(element);
    }
    return q;
}
```

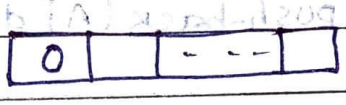
Q. First negative integer in every window of size K.

i/p ⇒ {-8, 2, 3, -6, 10}, K = 2

o/p ⇒ {-8 0 -6 -6}

Approach :⇒ deque<int> dq;

Step 1 :⇒ first K-size window
⇒ if we find -ve no, then we store its index.



answer = dq.front(); (or) 0 ↓ if dq.size == 0.

Step 2 :⇒ Now processing remaining windows

K → n-1

{-8, 2, 3, -6, 10}

↓

Remove krna hoga window se aur 3 ko add krna hoga.

//Removal ⇒ for (K → < n) {
if (index = dq.front() >= k)
dq.pop_front();

//addition
if (arr[index] < 0)
dq.push_back(index);

ans → dq.size() > 0 → ans = dq.front();
dq.size() == 0 → 0

index = dq.front() >= K ⇒ Jo bhi front mein pehi wali hai, wo current wali K size window mein aati hai ya nhi.

Program :>

```
Vector<long long printFirstNegativeInteger(long long int A[],
        long long int N, long long int K){
    deque <long long int> dq;
    Vector <long long> ans;
    // process first window
    for(int i=0; i<K; i++){
        if (A[i] <0)
            dq.push_back(i);
    }
    //push ans for first window
    if (dq.size() >0)
        ans.push_back(A[dq.front()]);
    else
        ans.push_back(0);
    //Now process for remaining windows
    for(int i=K; i<N; i++){
        //first pop out of window element
        if (!dq.empty() && (i-dq.front()) >=K){
            dq.pop_front();
        }
        // Then push current element
        if (A[i] <0)
            dq.push_back(i);
        // put in ans.
        if (dq.size() >0)
            ans.push_back(A[dq.front()]);
        else
            ans.push_back(0);
    }
    return ans;
}
```

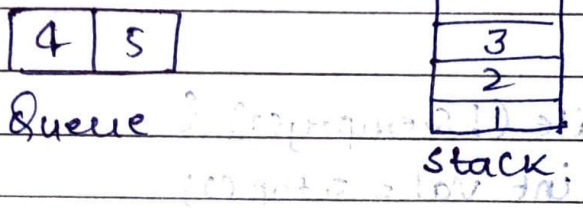
Q Reverse first K elements of Queue

i/p : {1, 2, 3, 4, 5}, K = 3

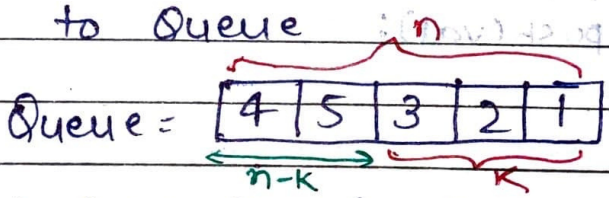
o/p = {3, 2, 1, 4, 5}

Approach :-

Step 1 :- fetch 1st K element from Queue & put into stack



Step 2 :- fetch element from stack & put in to Queue



Step 3 :→ fetch first (n-K) elements from Queue & push back.

T.C → O(K) + O(K) + O(n-K)
= O(n).

S.C → O(K)

program :→

```
queue<int> modifyQueue(queue<int>q, int k){
    stack<int>s;

    for(int i=0; i<k; i++){
        int val = q.front();
        q.pop();
        s.push(val);
    }

    while(!s.empty()){
        int val = s.top();
        s.pop();
        q.push(val);
    }

    int t = q.size()-k;

    while(t--){
        int val = q.front();
        q.pop();
        q.push(val);
    }

    return q;
}
```

First non-repeating character in a stream

i/p → a a b c
o/p → a # b b

Approach :-

Step 1 :→ Jo ~~saare~~ character aaye unka count store krnge.

```
map<char, int> count;
```

Step 2 :→ Queue mein push kr dnge character ko

Step 3 :→ q. front()

→ Repeating (count > 1)
  q.Pop()

→ Non-Repeating
  ans store

→ q.empty()
  return #

Program :-

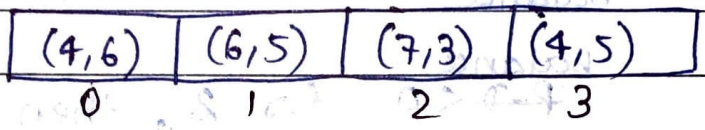
```
class Solution{
    public:
    string FirstNonRepeating(string A){
        map<char, int> m;
        string ans = "";
        queue<char> q;

        for(int i=0; i<A.length(); i++){
            char ch = A[i];

            q.push(ch);
            m[ch]++;

            while(!q.empty()){
                if(m[q.front()] > 1)
                    q.pop();
                else{
                    ans.push_back(q.front());
                    break;
                }
            }
            if(q.empty()){
                ans.push_back('#');
            }
        }
        return ans;
    }
};
```

Circular Tour

i/p :→ N = 4
Petrol = {4, 6, 7, 4}
Distance = {6, 5, 3, 5}

o/p :→ 1

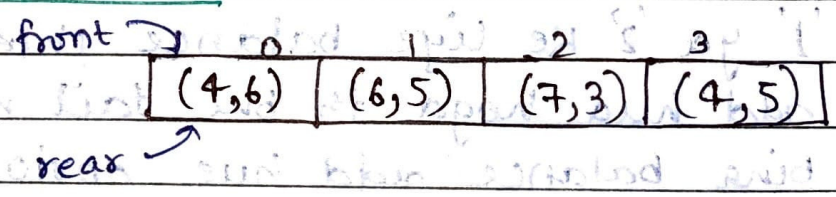
Approach 1 :→ Brute-Force



Phle index '0' ke liye check krunga, fir '1' ke liye, aise krke 'N-1' tk ke liye one by one check kr lunga.

T.C → $O(n^2)$

Approach 2 :→



Algo :→ If travel possible from one block to other (P-D >= 0), then rear++;

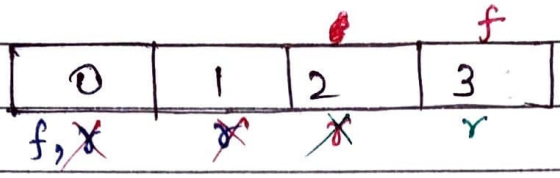
```
if (front == rear)
    Circle Complete
```

else → travel not possible

front = rear + 1;

rear = front;

NOTE :→ front = rear+1 isliye kiye hai kyuki,



agar balance P−D >= 0 for 0 index, then rear = 1

balance P−D >= 0 for 1 " , then rear = 2

balance P−D < 0 for 2, then front = 3.

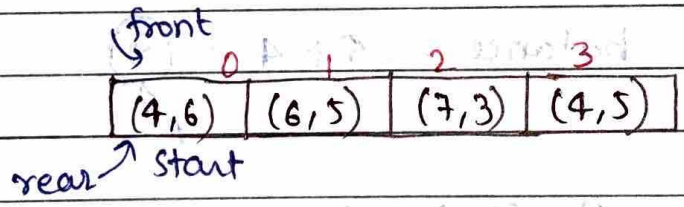
kyuki ab front++ se front '1' hoga jo ki fayda nhi hai kuch '1' ke liye check krne se, naa hi '2' ke liye kyuki '2' pe fail kr chuka hai, toh ab start 3 se hi check krna chahiye.

'1' ya '2' ke liye balance phle wala bhi add hua hoga tb bhi fail kr gya toh bina balance add hue obviously fail hi krega.

Algorithm :-

```
if (travel possible from start to next block)
        rear ++;

else
        front = rear + 1;
        start = front;
        rear = front;

if (front == rear)
        cycle Complete
```

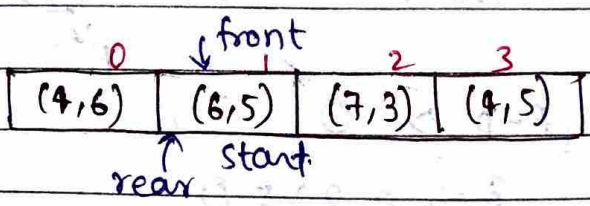
Dry Run



P - D >= 0 → false

⇒ 4 - 6 = -2 < 0

⇒ front = rear + 1.
start = front;
rear = front;



P - D >= 0 → True

6 - 5 = 1 >= 0

⇒ rear = 2

(4,6)	(6,5)	(7,3)	(4,5)

front → (6,5)
start → (6,5)
rear → (4,5)

balance = 1 (from last one remaining Petrol)

balance = 1 + 7 = 8.

P

P - D >= 0 → True

8 - 3 = 5 >= 0

rear = 3

(4,6)	(6,5)	(7,3)	(4,5)

front → (6,5)
start → (6,5)
rear → (4,5)

balance = 5 + 4 = 9

P

P - D >= 0 → True

9 - 5 = 4 >= 0

rear = 0

(4,6)	(6,5)	(7,3)	4,5)

rear → (4,6)
start → (6,5)
front → (6,5)
rear → (4,5)

balance = 4 + 4 = 8

P

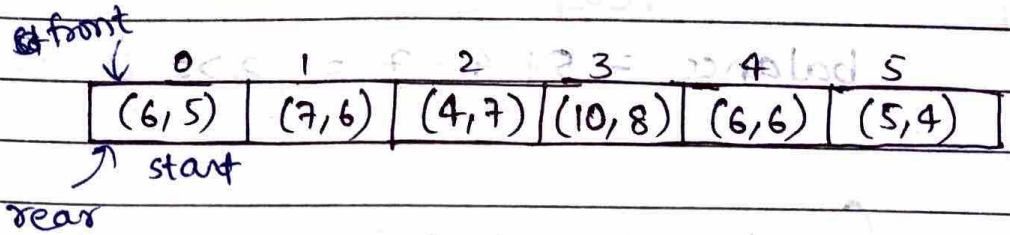
P - D >= 0 → True

8 - 6 = 2 >= 0

rear = 1

Now, rear == front ⇒ Cycle Complete

2nd example



int balance = 0

balance = balance + P - D

= 0 + 6 - 5 = 1 >= 0 → True

rear++;

rear=1;

balance = 1 + 7 - 6 = 2 >= 0 → True

rear++;

rear=2;

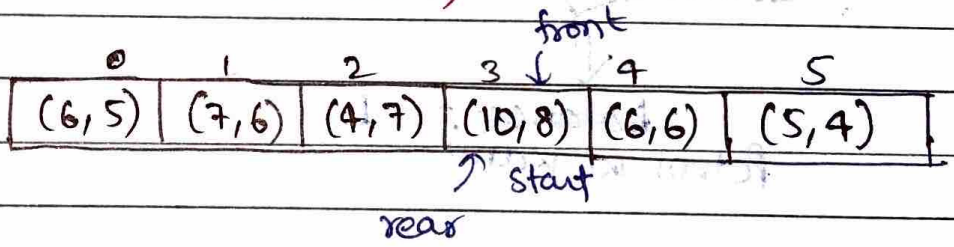
balance = 2 + 4 - 7 = -1 >= 0 → false

front = rear+1;

front = 3;

rear = 3;

start = 3;



balance = 0 + 10 - 8 = 2 >= 0 → T

rear = 4

balance = 2 + 6 - 6 = 2 >= 0 → T

rear = 5

balance = 2 + 5 - 4 = 3 >= 0 → T

rear = 0

balance = 3 + 6 - 5 = 4 >= 0 → T

rear = 1

balance = 4 + 7 - 6 = 5 >= 0 → T

rear = 2

balance = 5 + 4 - 7 = 2 >= 0 → T

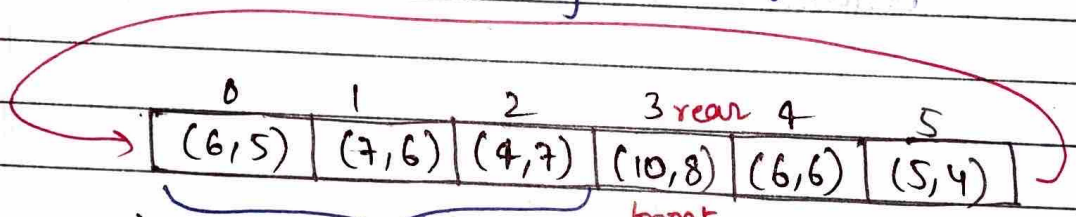
rear = 3

(front == rear) → Cycle Complete

NOTE :- In the above algorithm, in the worst case we need to visit the blocks twice.

⇓

for single visit



⇓

~~balance~~ = -1

Petrol ki kami

balance + kami >= 0 → Cycle Complete.

Program :-

```
class Solution {
    public:

    int tour (petrolPump p[], int n){
        int deficit = 0;
        int balance = 0;
        int start = 0;

        for (int i=0 ; i<n ; i++){
            balance += p[i].petrol - p[i].distance;

            if (balance <0) {
                start = i+1 ;
                deficit += balance;
                balance = 0;
            }
        }

        if (balance + deficit >=0) {
            return start;
        }
        return -1;
    }
};
```

Q Interleave the first half of the queue with second half.

i/p → [1, 2, 3, 4]
o/p → [1, 3, 2, 4]

i/p → [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
o/p → [11, 16, 12, 17, 13, 18, 14, 19, 15, 20]

Approach 1:-

Step 1 :→ fetch first half element from i/p queue & push into a new queue

i/p queue = | 16 | 17 | 18 | 19 | 20 |

New queue = | 11 | 12 | 13 | 14 | 15 |

Step 2 :→
```
while (!newqueue.empty())
{
    int val = newqueue.front()
    newqueue.pop();
    queue.push(val);
    val = q.front();
    q.pop()
    q.push(val)
}
```
q ⇒ queue

T.C → O(n), S.C → O(n)

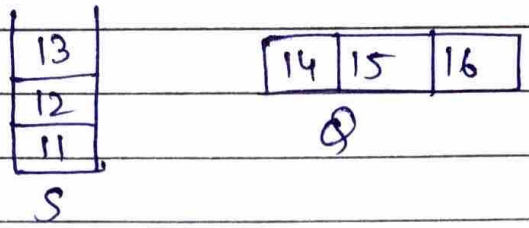
Approach 2:→

i/p → [11, 12, 13, 14, 15, 16]

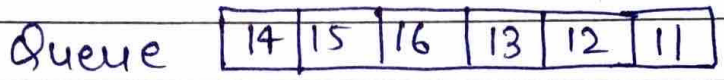
o/p → [11, 14, 12, 15, 13, 16].

Algorithm:→

Step 1:- first half of Q in stack.

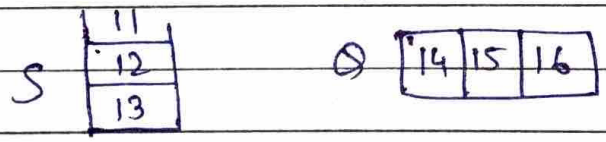


Step 2:→ stack → Queue

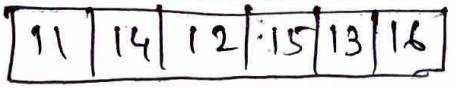


Step 3:→ first half of Q Pop & Push

Step 4:→ first half of Q in stack



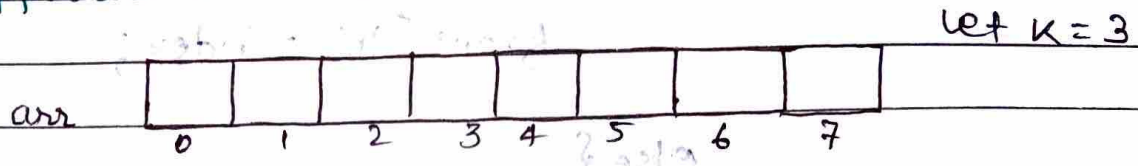

```
while (!S.empty()) {
    int val = S.top()
    S.pop()
    q.push(val)
    val = q.front()
    q.pop()
    q.push(val)
}
```

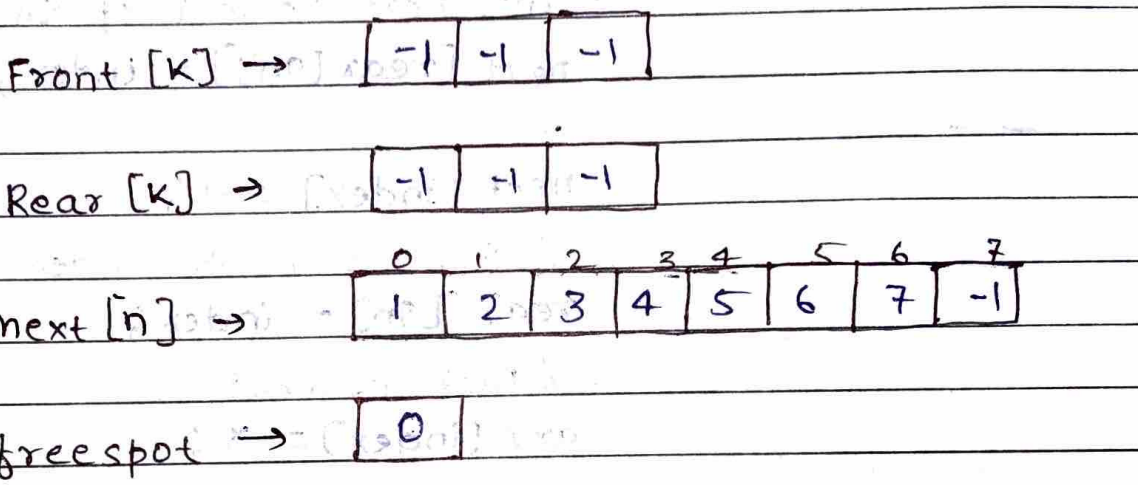
⇒ Queue = 

Q K-Queues in an array

Approach 1 :→ Brute force
Same as N-stacks in an array.

Approach 2 :→





→ Push Algo

Step :→ Check Overflow → freespot = -1.

→ find index where we want to insert.
int index = freespot;

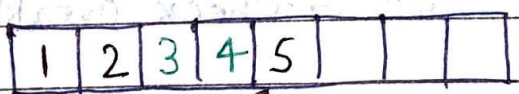
→ //update freespot
freespot = next [index]

NOTE:- Hmne front aur rear starting mein '-1' liye hue hai toh 1st element alag se handle krna prega.

```
→ // if first element
    if (front[qn] == -1)
        front[qn] = index;
    .
    else {
        // Link next element of qn in array
        next[rear[qn]] = index;
        .
        next[index] = -1;
        // Point rear to index
        rear[qn] = index;
        // Push element
        arr[index] = x;
```

Explanation :→

q1. Push(1);
q2. Push(2);
q2. Push(3)
q2. Push(4)
q2. Push(5)



In dono element ko link krke unka track rkh rha hai.

→ Pop Algo

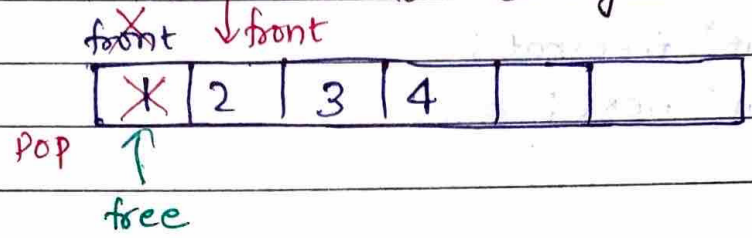
→ // is Empty → Underflow

→ // find index
int index = front [qn]

→ // front ko aage badhao

→ front [qn] = next [index]

→ // free slots ko manage kro



next [index] = free

free = index

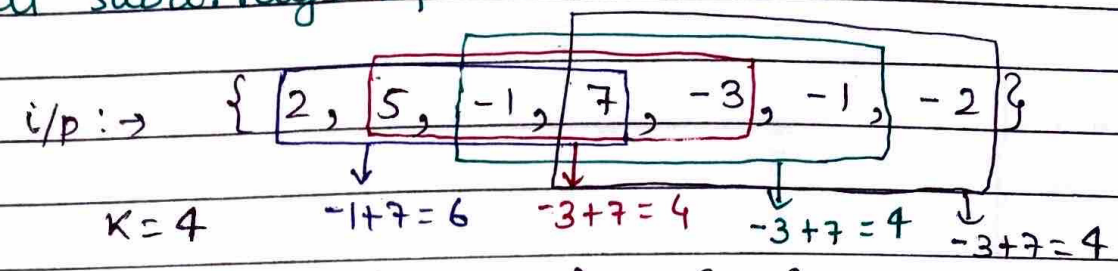
Program :→

```
class kQueue {
public:
    int n;
    int k;
    int * front;
    int* rear;
    int* arr;
    int* freespot;
    int* next;

public:
    kQueue (int n, int k){
        this->n = n;
        this->k = k;
        front = new int [k];
        rear = new int [k];
        for (int i=0; i<k; i++){
            front [i] = -1;
            rear [i] = -1;
        }
        next = new int [n];
        for (int i=0; i<n; i++){
            next [i] = i+1;
        }
        next [n-1] = -1;
        arr = new int [n];
        freespot = 0;
    }
```

```
void enqueue(int data, int qn){
    //overflow
    if (freespot == -1){
        cout << "No Empty Space" << endl;
        return;
    }

    //find first free index
    int index = freeSpot;

    //update freespot
    freeSpot = next[index];

    //check whether first element
    if (front[qn-1] == -1){
        front[qn-1] = index;
    }
    else{
        //link new element to the prev element
        next[rear[qn-1]] = index;
    }

    //update next
    next[index] = -1;
    //update rear
    rear[qn-1] = index;
    //push element
    arr[index] = data;
}
```

```
int dequeue (int qn) {

    if (front [qn-1] == -1) {
        cout << "Queue is Empty" << endl;
        return -1;
    }

    // find index to pop
    int index = front [qn-1];

    // front ko aage badhao
    front [qn-1] = next [index];

    // freeSlots ko manage kro
    next [index] = freeSpot;
    freeSpot = index;
    return arr [index];
}
};
```

Q = Sum of minimum and maximum elements of all subarrays of size K.



o/p :→ {6, 4, 4, 4} = {18}

→ Brute-force approach

```
For ( 0 → < n)
   { for (K times)        T.C → O(n*K)
      { max, min
      }
   Sum → store .
```

→ Approach 2 (T.C → O(n))

(i) deque (maxi) → decreasing order m element honge.
maxi.front() → max^m element in K-size window

(ii) deque (mini) → increasing order m element honge.
mini.front() → min^m element in K-size window