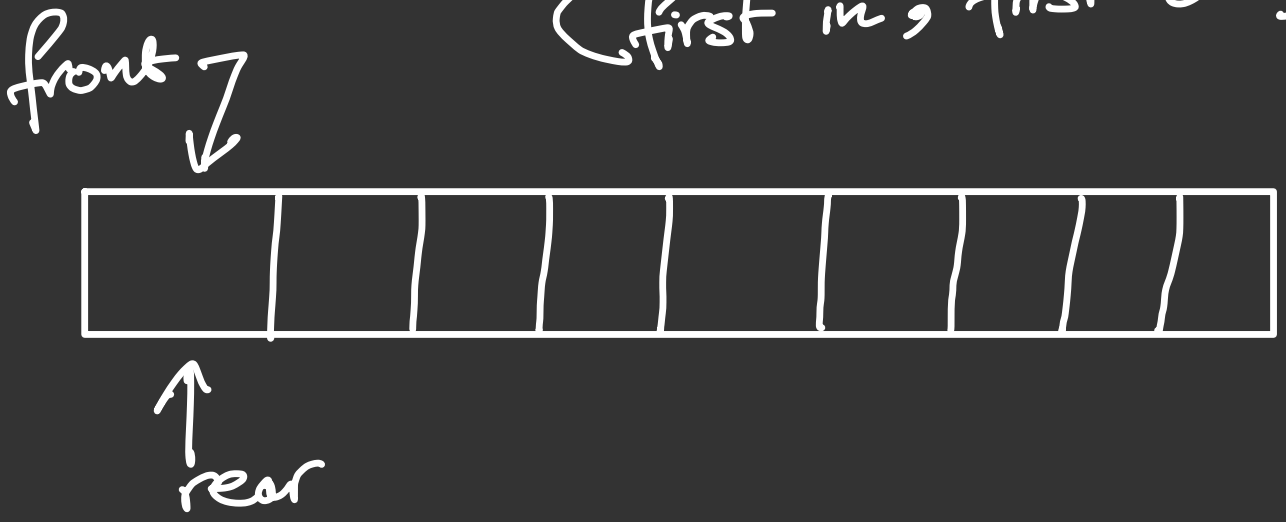
QUEUE's

Lecture 60

Queue → FIFO
(first in, first out)



push → rear
pop → front

Creation → queue<int> q;

q.push(17);
q.pop();
q.size(); → int
q.empty(); → bool

q.front();

↳ first element in the front

✡ You insert at rear, thus after each push, rear shifts.

push = enqueue
pop = dequeue

Implementation using Arrays

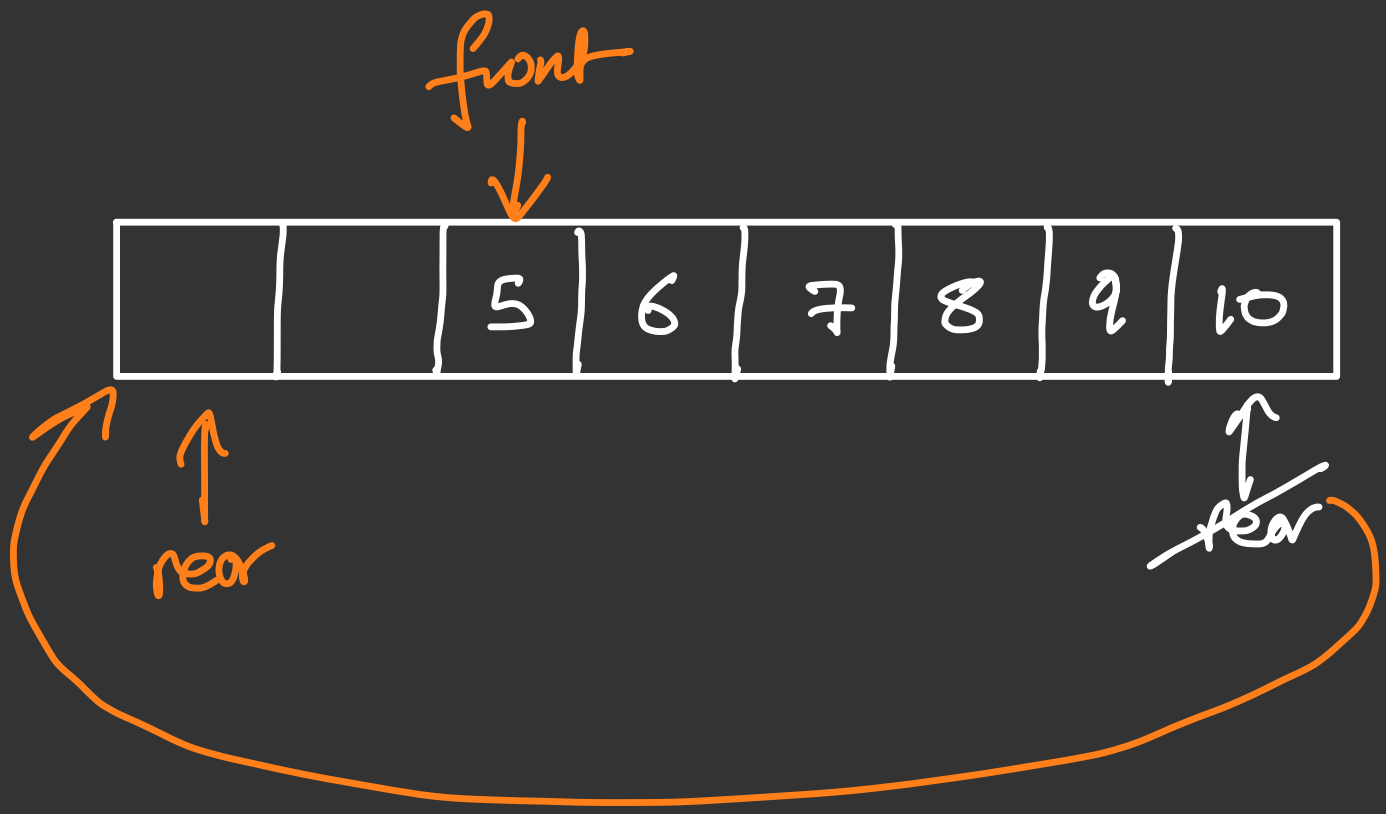
↳ queueImplementation_Arrays.cpp

Implementation using Linked List

↳ queueImplementation_LL.cpp

Circular Queue

★ If elements are stored till end of the array, rear now comes to arr[0].



⟶ circularQueue.cpp

Input Restricted Queue

- input only on side (rear)
- pop can be both sides
  - (i) pop_back
  - (ii) pop_front

Output Restricted Queue

- input both sides
  - (i) push_front
  - (ii) push_back
- output / pop from only front side

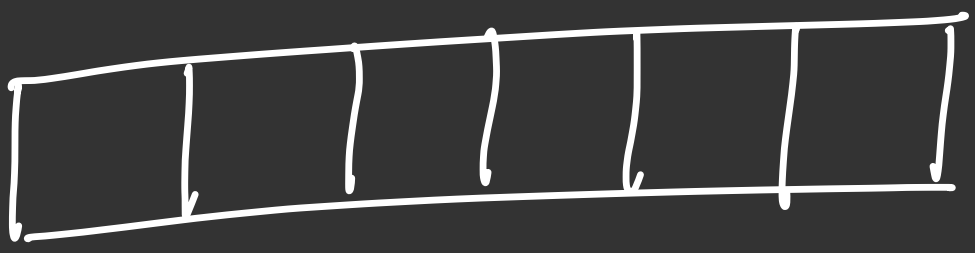
Double Ended Queue

- push_back
- push_front

- pop_back
- pop_front

Initially:

front = -1
rear = -1



→ doubleEndedQueue.cpp

PUSH FRONT

→ if full
    return -1;

→ front == -1 (first element)
    front = 0
    rear = 0

→ front = 0
    front = s-1;

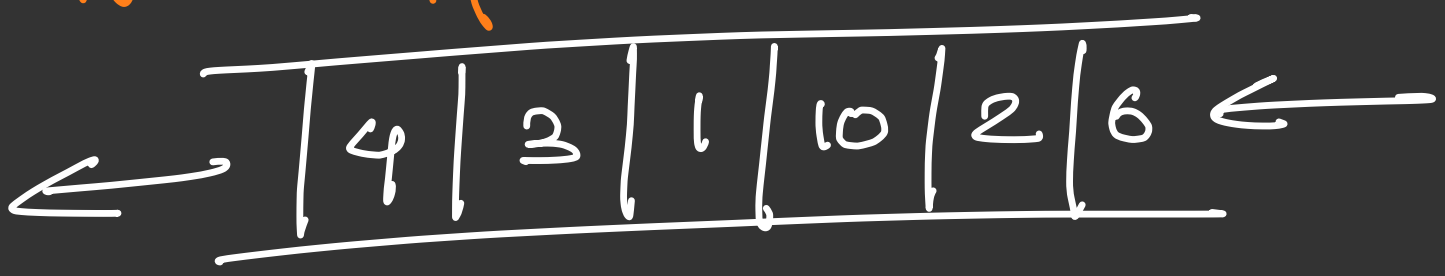
→ front--;
    insert at front

isFull

```
if ((rear+1)%s == front)
    return true;
else
    return false;
```

Lecture 61

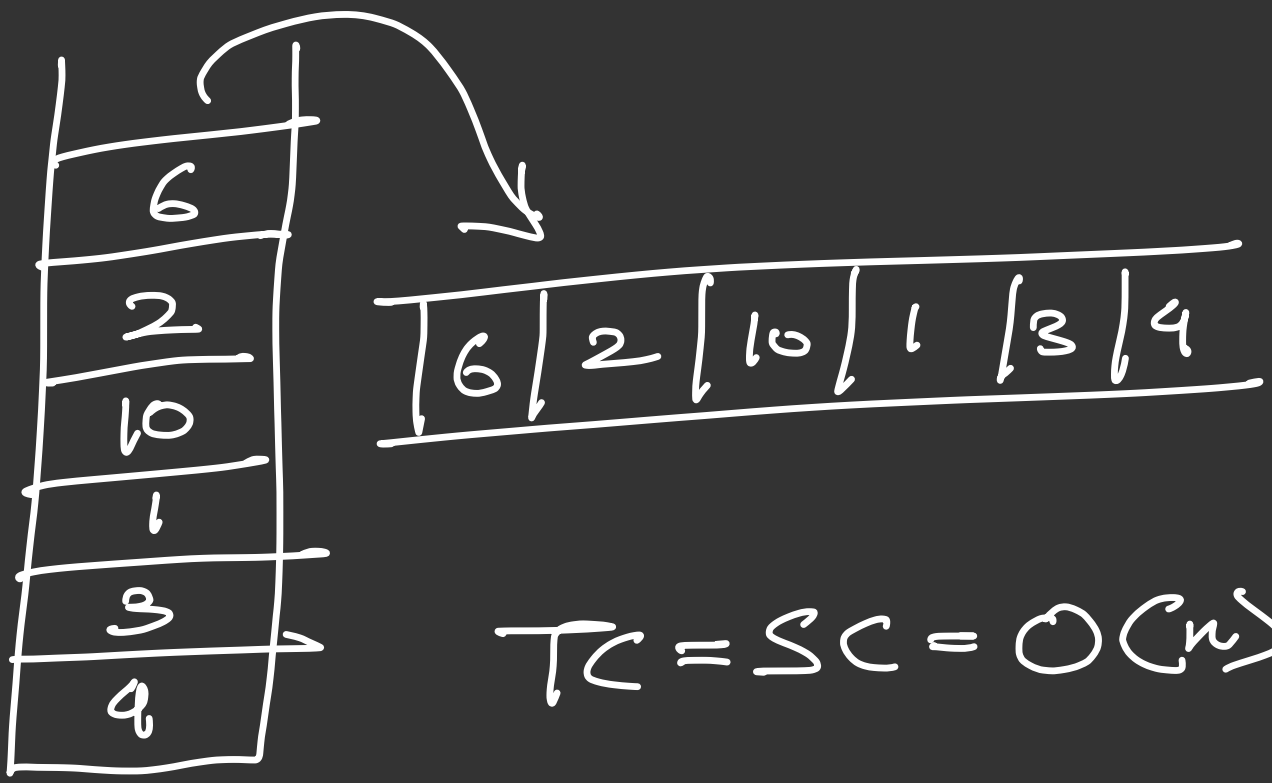
Reverse of Queue



Approach 1 →

Use Stack

→ whatever you put in stacks comes out in reverse.



TC = SC = O(n)

Approach 2 →

Recursion

Base Condition

```
if(q.empty())
    return;
```

1. Store front element
2. Pop front element
3. Recursive call
4. Push that element.

Find first negative integer in every window of size k.

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

window size = 2

→ one element from the right is entering the window and one is exiting

① Instead of pushing all the elements in queue, only push the negative ones.

→ Do it for first K elements

(2) Now loop (i = K ——— <N)

if queue is empty

↳ no negative in first window

else

↳ push front as it is first negative

✍ Then check if the index of that negative element is same as the first element of the last window, because that leaves the window

Thus, store only the index.

And check for A[i] if < 0

if yes then push

→ firstNeg_windowK.cpp

Reverse but only first K elements

→ Reverse the first K elements using Recursion

but now they will be at the back of Queue

→ So, now loop for (i = 1 → n − K)

we store, pop, and push.

→ reverse_Kelements.cpp

First non-repeating character Stream

→ all small case alphabets

1. An array of 26 elements marked at 0.
2. A queue with first character of the string
   → and first char arr[ch-97] = 1
3. Loop i ——— < s.length()
   → if arr[ch-97] == 0
   push in queue and mark as 1.

→ if already I make as 2.

→ Now,

```
if q.front() == 1
    store  q.front();
else
    while (q.front() == 2
        && !q.empty())
    {
        q.pop();
    }
```

(i) If q. is empty
store '#'

(ii) else
store q.front

→ firstNonRepeatingChar_Stream.cpp

Circular Tour

A circular road has N petrol pumps

→ litres of petrol at each petrol pump = A

→ distance to the next petrol pump. = B

A > B → for each possible solution petrol pump

Sum of B <= Sum of A

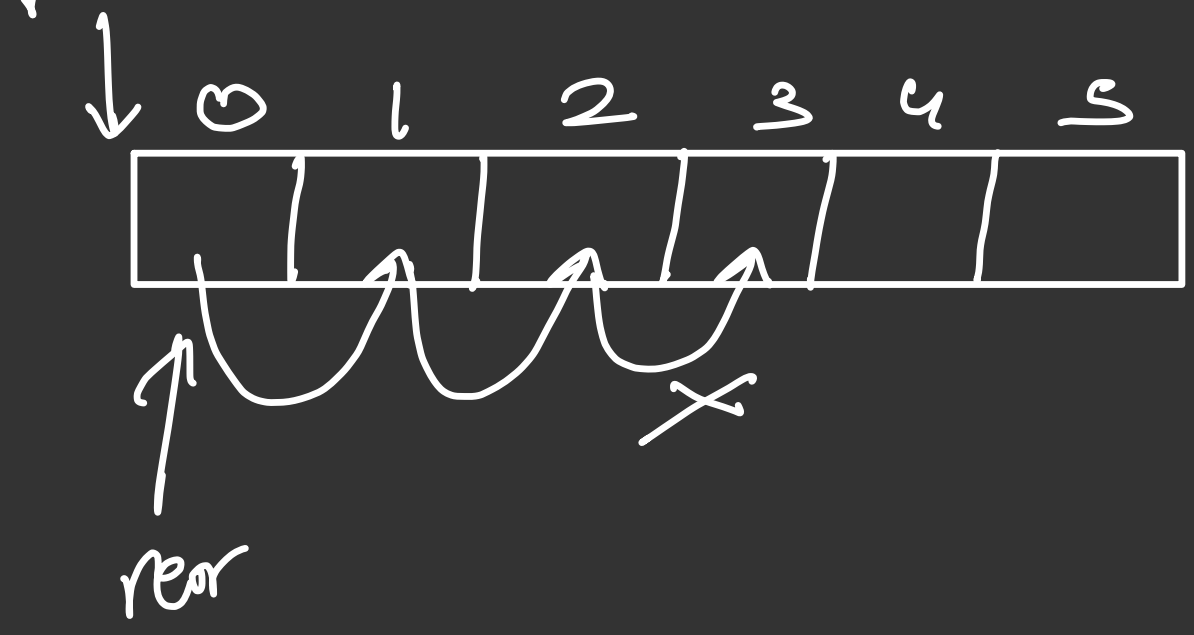
↳ for answer to exist.

Approach 1 ⟶

for all petrol pumps for where A > B, we check for them if they can be a possible solution

TC = O(n²)

Approach 2 ⟶



If after starting from 0, when rem = 1 some bal >= 0 must have been added, so if still there is a problem in going from 2 to 3, that means same would have been if we would have started from 1, 2.

⟶ So if the bal < 0 for a starting pump, ie. it needed that much (0 - bal) to be equal to zero or more than zero.

(J = deficit @ kami

Thus, we keep storing that (bal < 0) in d.

& so to check for full circle, if rear == n we check if the balance remaining is going to make d >= 0, if yes that is the ans, else -1.

Remember, we have to traverse from rear = 0 to rear = n-1 for O(n).

So, to check the 0-front part we use d.

⟶ circular_queue.cpp

Interleave the first half of the queue with second half. (even number of elements)

Example:

i/p ⟶ 1 2 3 4

o/p ⟶ 1 3 2 4

= {11, 16, 12, 17, 13, 18, 14, 19, 15, 20}

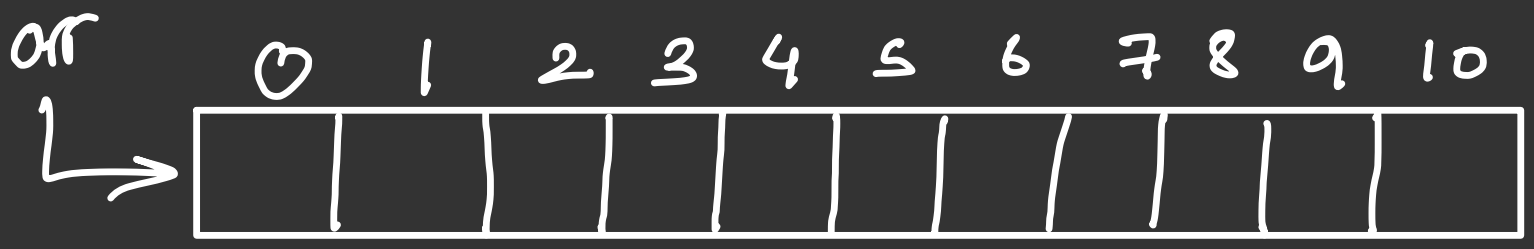
Using stack only as auxillary space

(1) Store first half in stack, reverse stack

(2) Now push from both one by one

⟶ interleave-queue.cpp

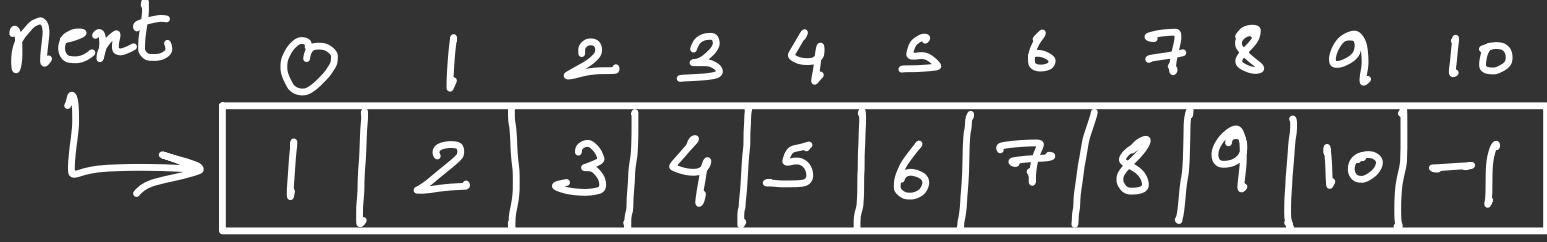
'K' Queues in an Array



front [K] = | -1 | -1 | -1 |

rear [K] = | -1 | -1 | -1 |

free = 0;



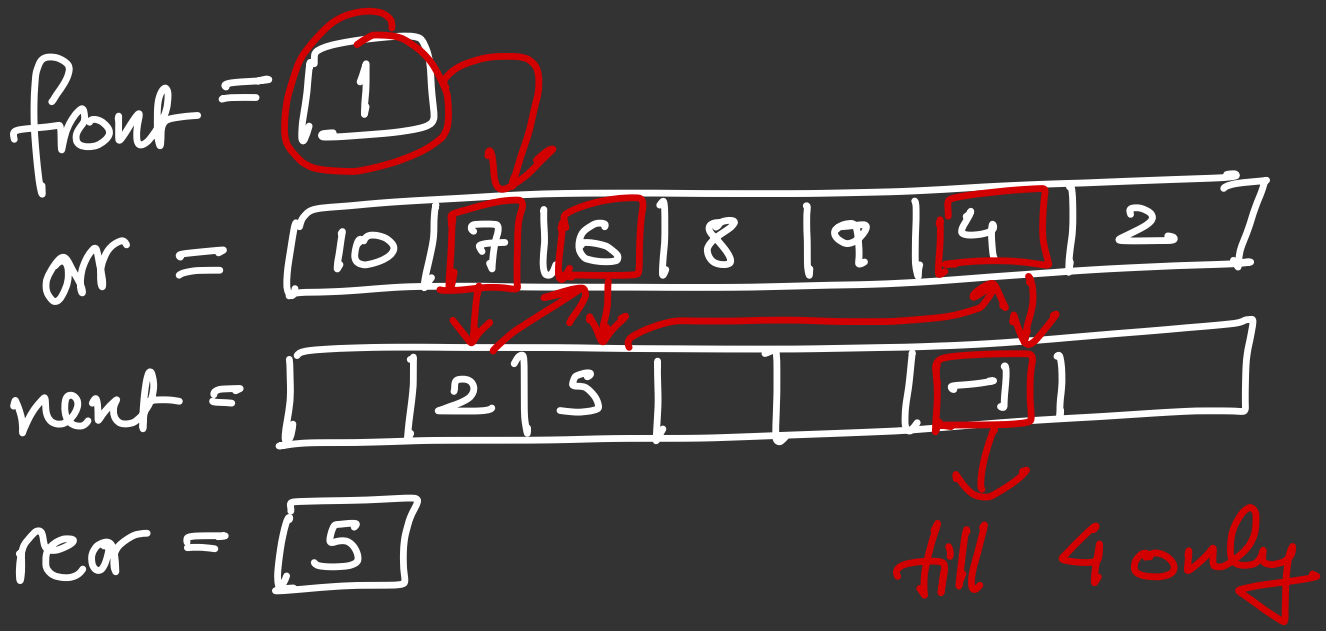

```
push(x, m)
    if free == -1
        cc "Overflow"

    int index = free;
    free = next[index];
    arr[index] = x;

    if (front[m-1] == -1)
    {
        front[m-1] = index;
    }
```

```
else
{
    next[rear[n-1]] = index;
}
next[index] = -1;
rear[n-1] = index;
```

⚠ We have to connect front → next not from (top, next) to till bottom as in stack



pop(int m)

→ Underflow

```
index = front[m-1];
front[m-1] = next[index];
next[index] = free
free = index;
```

→ N_QueueInArray.cpp

Sum of Min and Max of SubArrays of size K

→ 2 dequeue to store max and min

1. Do it for first K elements
2. Now add sum
3. Now

for (i=K — i<N)

→ pop while front elements are out of range of current subarray

→ Check the ith element

→ SumMinMax_subArrays.cpp