Queue

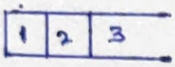
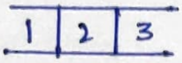
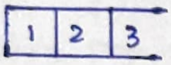
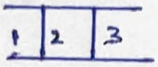
→ It is also a Linear Data Structure which follows FIFO.

→ FIFO (First In First Out), for ex: If we are in a queue, the first person to come in, he will go out first.

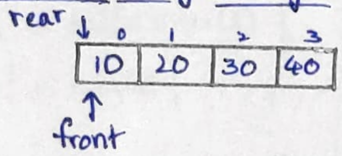
Syntax:- → Interface, not or a class

Queue<DataType> queue = new LinkedList<>();

Operations:

Stack	Queue
1. push() 	1. add() 
2. pop()  ans: 3, 2, 1	2. poll() → the first inserted element will be returned  ans: 1, 2, 3.
3. peek() last element → 3	3. peek(). first Element → 1
4 isEmpty() → True/false	4. isEmpty() → true/false

Queue Implementation Using Array:



rear = points to the last element

front = points to the first element

→ Treat the array as Circular Array, even though the array is not circular array, rear will increment like it is circular.

→ Take 2 pointers, front and rear, initially pointing to same location.

→ Insert /add() / enqueue()

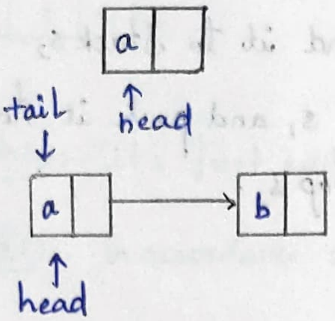
a) Check overflow condition ⇒ insert

→ Delete / Poll() / dequeue()

b) check Underflow condition ⇒ f++

```
void Enqueue (int number){
        rear ++,
        rear= (rear +1) % arr.length
        if (rear == front)
        { s.o.pln ("Overflow");
        // we need to reset our rear pointer.
            if (rear ==0)
                rear = arr. length -1;
            else
                rear = ~~arr.leng~~ rear -1;
        }
      else
          {
            queue[rear] = number;
          }
   }
int dequeue (){
   if (front ==rear)
   {
     s.o.pln("Underflow");
     return -1;
   }
   else {
       front=(front +1)%arr.length,
       number= queue[front];
       return number;
     }
}
```

Queue Implementation Using Linked List:



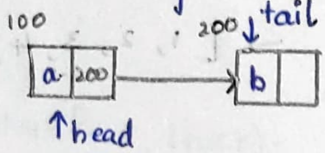
add:

we need to add at the Last of the list

① → move the tail pointer to new node

② → Before step ①, tail.next point to new node.

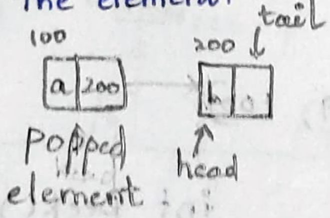
So, now tail points to b node.



Pop():-

→ popping the head ; → pointing the head to head.next;

→ when pop the element.



```
void push (int n){
 Node node = new Node(n);
 if (head == null || tail == null) {
   head = = tail = node;
 }
 else{
   tail.next = node;
     tail = node
   }
 }
```

```
int pop(){
   if(head= = null) {
     return -1; // queue is empty
   }
  Node node = head;
    head = head.next;
   if (head ==null) {
     tail = null; → reseting the tail if queue becomes empty.
   }
   return node.data;
}
```

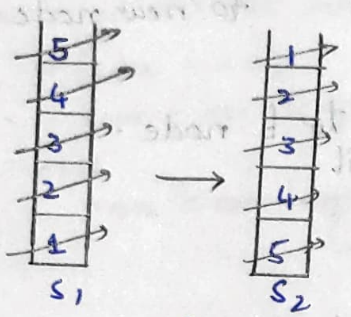
Using Stacks:

→ Pushing all the elements in Stack s_1

→ Popping all elements froms; and it to Stack s_2.

If s_2 is empty ; empty s_1 and push it to s_2 and then pop().

If s_2 is not empty ; pop s_2.



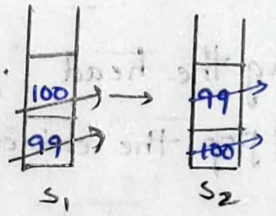
→ [1, 2, 3, 4, 5]

→ add, more elements to s_1

→ s_2 is empty, so empty s_1 and Push to s_2.

→ pop s_2.

[1, 2, 3, 4, 5, 99, 100]



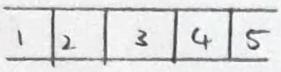
```
void push(int n){
    s1.push(n);
}

int pop() {
    Peek();
    return s2.pop();
}
```

```
int peek(){
    if (s2.isEmpty()){
        while (s1.isEmpty())
        {
            s2.push(s1.pop());
        }
    }
}
```

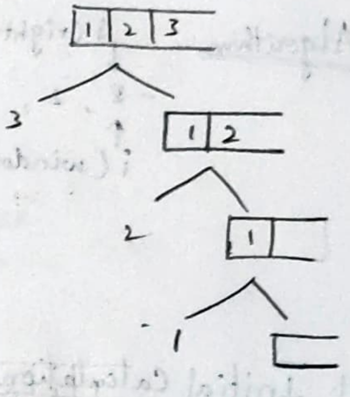
```
boolean empty(){
    // check both s1 and s2
    return s1.isEmpty() && s2.isEmpty();
}
```

Using One Stack: Reverse a Stack Using Recursion

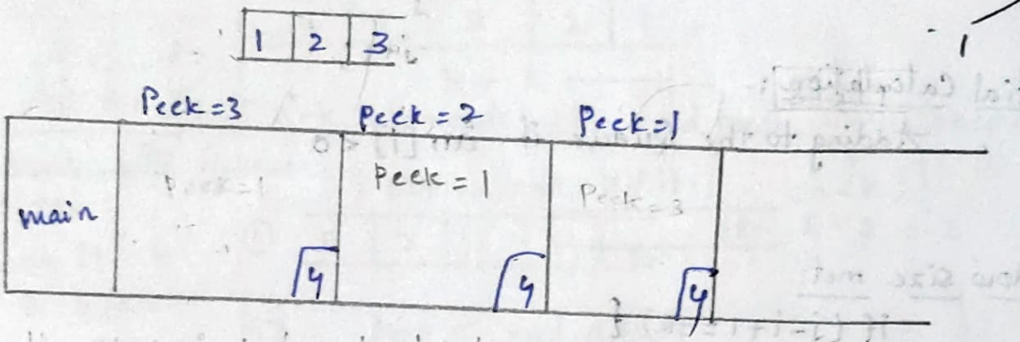


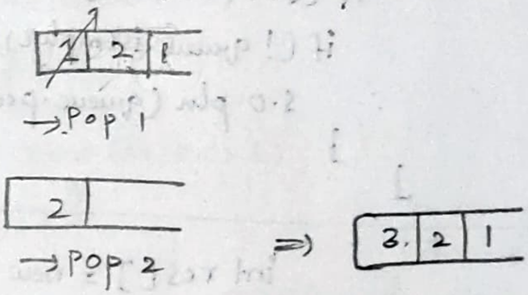
Push():- It's just adding elements into stack

Pop():. in accordance with FIFO



```
1. int stackPeek = stack.peek();
2. stack.pop();
3. rev(stack);
4. insertAt bottom (stackPeek, stack).
```





```
insertAt Bottom (int stackPeek)
{
   if (stack.isEmpty())
   {
     stack.push (stackPeek);
   }
   else {
       int peek = Stack.peek();
       stack.pop();
       inserAtBottom (stackPeek);
       Stack.push (peek);
     }
}
```

** First -ve Integer in Every Window of Size k:

I/P: -8, 2, 3, -6, 10 , k=2

O/P: -8, 0, -6, -6
0 1 2 3

→ For each window, check if there is any negative number.

→ If negative number is available get the first negative number.

→ If not result is zero.

Algorithm:-

```
j (right end)
↓
-8, 2, 3, -6, 10
↑
i (window start)
```

```
while(j<arr.length)
{
    //initial calculation
    //If window hits k, other calculation
    i++;
}
j++;
```

1. Initial Calculation:-
Adding to the Queue if arr[i] < 0

| -8 |

2. Window size met:

```
if (j-i+1==k) {
    if (! queue.isEmpty()) {
        s.o.pln (queue.peek());
    }
}
```

K = 2

```
 0   1   2    3    4
-8 , 2 , 3 , -6 , 10
     ↑   ↑    ↑    ↑
         i    i    j
```

| -8 |

j++, ⇒ j is at 2

j-i+1 ⇒ 1-0+1 ==2

res[0] = -8

arr[0] = queue peek
= -8 ✓

So queue.remove

| |

j-i+1 ⇒ 2-~~1~~+~~1~~==2

arr[1] = 0 ✓

| -6 |

arr[2] = -6 ✓

arr[i] != peek, so

| -6 |

arr[j] != 0

arr[3] = 6 ✓

```
int res[] = new int[n-k+1];
int i=0; j=0;
while(j<n){
    if (arr[j] < 0){
        queue.add (arr[i]);
    }
    if(j-i+1 = =k){
        if(! queue.isEmpty()){
            res[i] = queue.peek();
            if(arr[i] = = queue.peek()){
                queue.remove();
            }
        else{ res[i] = 0; }
        } i++;
} j++;
```

add to the queue if element<0

if window size reaches

If the element at i is peek, remove it

If any element is present add add it to array

else add '0' to the array

* Reverse First K Elements from Queue:

I/P:

1	2	3	4	5

k = 3

O/P:

5 4 3 2 1

Algo:

1. Push first k elements to the stack.

3
2
1

2. Push the stack elements back to the Queue.

← n →

4	5	3	2	1

← k →

3. For the first n-k elements pop and push simultaneously back into queue.

(n-k)
5 - 3 = 2

① Pop 4 and add it

5	3	2	1	4

② Pop 5 and add it

3	2	1	4	5

```
Stack<Integer> stack = new Stack<>();
for(int i=0; i<k; i++)
{
    Stack.add(q.poll());
}
```

adding first k elements to Stack

```
while(!stack.isEmpty()){
    q.add(stack.pop());
}
```

adding elements from Stack to Queue

```
for(int i=0; i<q.size()-k; i++)
{
    int element = q.poll();
    q.add(element);
}
return q;
```

Polling & adding back upto (n-k) to the Queue.

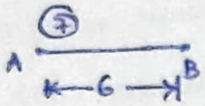
First Circular Tour: (Petrol & Distance)

A B C D E F

Petrol [] ⇒ 7, 8, 5, 11, 7, 6   o/p: D

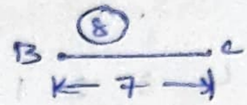
distance[] ⇒ 6, 7, 8, 9, 7, 5

(a)



extra = 1L

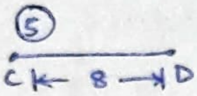
b)



Petrol available = 1 + 8
= 9

extraFuel = 9 - 7
= 2L

(c)



Petrol available = 2 + 5
= 7

extra Fuel = 7 - 8 = -1 < 0.

→ We need to find the point from which we can start such that we can start and cover the entire route.

→ Start at point D:

extra Fuel = 11 - 9 = 2L

→ At point E: Petrol Available = 2 + 7 = 9

extral Fuel = 9 - 7 = 2L

→ At point F: Petrol Available = 2+6 = 8

extra Fuel = 8 - 5 = 3L

→ At point A: Petrol available = 3 + 7 = 10

extra Fuel = 10 - 6 = 4L

→ At point B: Petrol: 4 + 8 = 12L

extra Fuel = 12 - 7 = 5

→ At point c: Petrol: 5 + 5 = 10L

extra Fuel: 10 - 8 = 2L

→ We can each Reach point D:

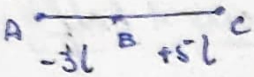
Brute Force:

7, 8, 5, 11, 7, 6

6, 7, 8, 9, 7, 5

We need to check at each point, so

O(n^2)



-2l Storage if we start at B.

At A:
extra = -3
required = -3

At B:
next
extra = 0+5

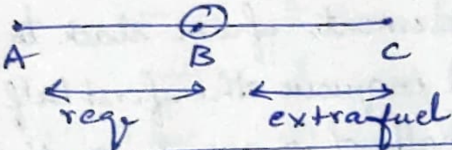
	A	B	C	D	E	F
P	7,	8,	5,	11,	7,	6
d	6,	7,	8,	9,	7,	5

$$eF = (7-6) + (8-7) + (5-8)$$

$$= 1 + 1 - 3$$

$$= -1 < 0$$

Start a Fresh



$\boxed{\text{required} <= \text{extraFuel}}$

```
int start = 0;
int requiredPreviousFuel = 0;
int extraFuel = 0;
for(int i=0; i< petrol.length; i++){
    extraFuel+=petrol[i]- distance[i];
    if(extrafuel<0) {
        Start = i+1;
        required Previous Fuel + = extra Fuel;
        extraFuel = 0;
    }
}
    int ans = -1;
    if (extra Fuel >= Math.abs(required PreviousFuel){
        ans = start;
    }
    return ans;
```

**Interleave the First Half of the Queue With Second half:

→ Given a queue of integers of even length, rearrange the elements by interleaving the first half of the queue with 2nd half.

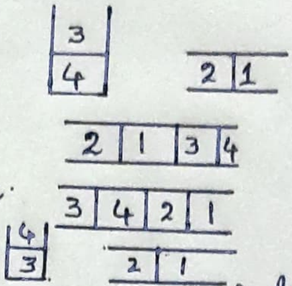
I/P: | 4 | 3 | 2 | 1 | | 1 | 2 | 3 | 4 | 5 | 6 |

| 4 | 3 | | 2 | 1 | | 1 | 2 | 3 | | 4 | 5 | 6 |

| 4 | 2 | 3 | 1 | | 1 | 4 | 2 | 5 | 3 | 6 |

Algorithm:-

1. add first half of the queue to a stack.
2. Enqueue the elements from stack to queue.
3. Dequeue and enqueue the first half of queue.
4. add first half of queue to a stack.
5. add elements to the list from stack and then queue each time.



| 4 | 2 | 3 | 1 |

```
Stack<Integer> st = new Stack<>();
① for (int i=0; i<N/2; i++){
      st.push(q.poll());
  }
② while(!st.isEmpty()){
      q.add(st.pop());
  }
③ for(int i=0; i<N/2; i++){
      int front = q.poll();
      q.add(front);
  }
④ for (int i=0; i<N/2; i++){
      St.add(q.poll());
  }
⑤ while(!st.isEmpty() && !q.isEmpty()){
      list.add(st.pop());
      list.add(q.poll());
  }
```