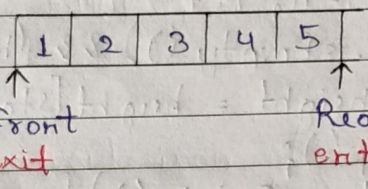
QUEUES

* Meaning of Queue is → एक के एक बाद खड़ा होना
मतलब line लगाना।

* Queue works on FIFO principle.



* Operations :-
1. Enque → Adding → O(1)
2. Dequeue → Remove → O(1)
3. Front → Peek → O(1)

#NOTE: Dequeue & Deque → both are different.
↓
→ Double ended जिसमें दोनों तरफ से entry & exit हो सकती है।

★ QUEUE IMPLEMENTATION USING ARRAY

→ In Array, we can make fixed size of Queue.

adding → O(1)
removing → O(n)

* Time Complexity → O(n)
* Space Complexity → O(1)

⇒
```
static class Queue {
  static int arr[];
  static int size;
  static int rear;
  Queue(int n) {
    arr = new int[n];
    size = n;
    rear = -1;
  }
  public static boolean isEmpty() {
    return rear == -1;
  }
  //Enque
  public static void add(int data) {
    if (rear == size-1) {
      System.out.println("Queue is full");
      return;
    }
    rear = rear+1;
    arr[rear] = data;
  }
  //Dequeue
  public static int remove() {
    if (isEmpty()) {
      System.out.println("Empty Queue");
      return -1;
    }
    int front = arr[0];
    for (int i=0; i<rear; i++) {
      arr[i] = arr[i+1];
    } rear = rear-1;
    return front;
  }
```

```
//peek
public static int peek() {
    if (isEmpty()) {
        System.out.println("Empty Queue");
        return -1;
    }
    return arr[0];
}
} public
public static void main (String args[]) {
    Queue q = new Queue(5);
    q.add(1); q.add(2); q.add(3);
    while (!q.isEmpty()) {
        System.out.println(q.peek());
        q.remove();
    }
}
```

★ CIRCULAR QUEUES USING ARRAYS

* In this, remove → O(1)
* Circular queue uses an array in a circular fashion.
* This mean when we reach reach of the end of array, the next element would be at the start of the array.

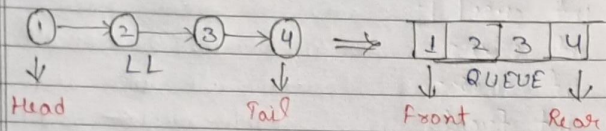
NEXT PAGE

⇒
```
static class Queue {
    static int arr[];
    static int size;
    static int rear;
    static int front;
    Queue(int n) {
        front = -1;
    }
    public static boolean isEmpty() {
        return rear == -1 && front == -1;
    }
    public static boolean isFull() {
        return (rear+1) % size == front;
    }
    public static void add(int data) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear+1) % size;
        arr[rear] = data;
    }
    public static int remove() {
        int result = arr[front];
        if (rear == front)
            rear = front = -1;
        else
            front = (front+1) % size;
        return result;
    }
```

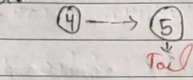
// Enque - O(1)
// Dequeue - O(1)
// front - O(1)

peek() { return arr[front];

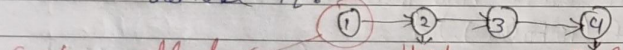
★ QUEUE USING LINKED LIST



* To Add, basically we made an new node & add it from tail.



* To Remove, we simply point out head.next as head then garbage collector will remove it.



⇒
```
static class Node {
    int data;
    Node next;
    Node (int data) {
        this.data = data;
        this.next = null;
    }
}
static class Queue {
    static Node head = null;
    static Node tail = null;
    public static boolean isEmpty () {
        return head == null & tail == null;
    }
```



```
//add
public static void add (int data) {
    Node newNode = new Node (data);
    if (head == null) {
        head = tail = newNode;
        return;
    }
    tail.next = newNode;
    tail = newNode;
}
//remove
public static int remove () {
    if (isEmpty ()) {
        Sysо ("Empty Queue");
        return -1;
    }
    int front = head.data;
    //single element
    if (tail == head) {
        tail = head = null;
    } else {
        head = head.next;
    }
    return front;
}
//peak
public static int peak () {
    if (isEmpty ()) {
        Sysо ("Empty Queue");
        return -1;
    }
    return head.data;
}
```

★ QUEUE USING JCF

* Queue is an type of interface therefore, they can by made using only two classes i.e., → LinkedList
→ ArrayDeque

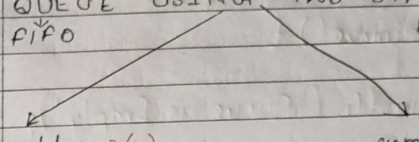
★★★ IN JCF, TIME-COMLEXITY is always O(1).

⇒
```
public static void main (String args[]) {
  Queue <Integer> q = new LinkedList <>();
                    //ArrayDeque <>();
  q.add(1);
  q.add(2);
  q.add(3);
  while (!q.isEmpty()) {
    System.out.println(q.peek());
    q.remove();
  }
}
```

LinkedList	ArrayDeque
* Based on a Doubly LL.	Based on a resizable array
* It allows more efficient insertions & deletions from both the ends of list.	Elements are stored in a contiguous block of memory, & array resizes itself.
* Does allows null element.	Doesn't allows
⇒ Use LL, if you need a list-like structure where frequent insertions & deletions at both ends are important.	Use AD, for more memory efficient & performant queue implementation in most scenarios.

★ QUEUE USING TWO STACKS

FIFO — LIFO



* For addition, S1 is Empty
S1 → S2
S1 push
S2 → S1

* For remove, S1 → pop

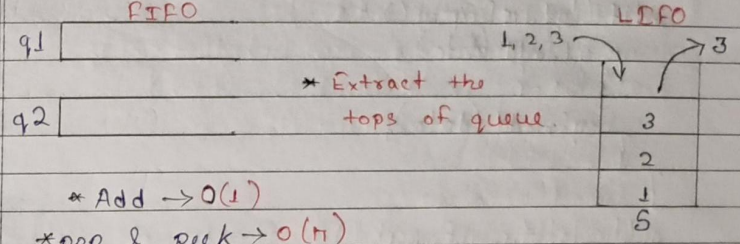
* For peek, S1 → peek

⇒
```
static class Queue {
  static Stack <Integer> s1 = new Stack <>();
  static Stack <Integer> s2 = new Stack <>();
  public static boolean isEmpty() {
    return s1.isEmpty();
  }
  //add
  public static void add (int data) {
    while (!s1.isEmpty()) {
      s2.push(s1.pop());
    }
    s1.push(data);
    while (!s2.isEmpty()) {
      s1.push(s2.pop());
    }
  }
  //Next page
```

```
//remove
public static int remove() {
  if (isEmpty()) {
    System.out.println("Queue is Empty");
    return -1;
  }
  return s1.pop();
}
//peek
public static int peek() {
  if (isEmpty()) {
    System.out.println("Queue is Empty");
    return -1;
  }
  return s1.peek();
}
}
public static void main(String args[]) {
  Queue q = new Queue();
  q.add(1);
  q.add(2);
  q.add(3);
  while (!q.isEmpty()) {
    System.out.println(q.peek());
    q.remove();
  }
}
```

Output → 1
2
3

★ STACK USING TWO QUEUES



⇒

```
static class Stack {
  static Queue<Integer> q1 = new LinkedList<>();
  static Queue<Integer> q2 = new LinkedList<>();
  public static boolean isEmpty() {
    return q1.isEmpty() && q2.isEmpty();
  }
  public static int pop() {
    if (isEmpty())
      Syso("Empty Stack");
      return -1;
    int top = -1;
    if (!q1.isEmpty())
      while (!q1.isEmpty())
        top = q1.remove();
        if (q1.isEmpty())
          break;
        q2.add(top);
    else
      while (!q2.isEmpty())
        top = q2.remove();
        if (q2.isEmpty())
          break;
        q1.add(top)
    return top;
```

```
public static void push(int data) {
    if (q1.isEmpty()) {
        q1.add(data);
    } else {
        q2.add(data);
    }
}

public static int pop() {
    if (isEmpty()) {
        System.out("Empty stack");
        return -1;
    }
    int top = -1;
    if (q1.isEmpty()) {
        while (q1.isEmpty()) {
            top = q1.remove();
            q2.add(top);
        }
    } else {
        while (q2.isEmpty()) {
            top = q2.remove();
            q1.add(top);
        }
    }
    return top;
}
```

```
public static void main(String[] args) {
    Stack s = new Stack();
    s.push(1); s.push(2); s.push(3);
    while (!s.isEmpty()) {
        System.out.println(s.peek());
        s.pop();
    }
}
```

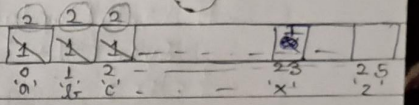

★ FIRST NON-REPEATING LETTER IN A STREAM OF CHARACTERS

Input ⇒ a a b c c x b → (stream)

	Output
a	a
aa	-1
aab	b
aabc	b
aabcc	b
aabccx	b
aabccxb	x

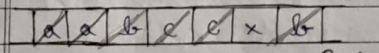
We need FIRST NON-REPEATING LETTER.

S1: Take an Array, freq[26] =



S2: input → a a b c c x b

counting Frequency



// Time Complexity, ⇒ O(n)
// Space Complexity, ⇒ O(1)

Output

① "a" → a
② "aa" → -1
③ "aab" → b
④ "aabc" → b
⑤ "aabcc" → b
⑥ "aabccx" → b
⑦ "aabccxb" → x

* Pseudo Code

① freq[] = [26] 'a' to 'z'
② Queue<Character>

```
for (int i=0; i<str.length(); i++)
    ch = str.charAt(i);
    q.add(ch)
    freq[ch-'a']++
    q.remove → till ; find first → print freq[q.peek()]
                non-repeating      = 1;
    if q.empty → -1
}
```

⇒
```
public static void printNonRepeating (String str) {
    int freq [] = new int [26];
    Queue <Character> q = new LinkedList <>();
    for (int i=0; i<str.length(); i++) {
        char ch = str.charAt(i);
        q.add(ch);
        freq [ch-'a']++;
        while (!q.isEmpty() && freq[q.peek()-'a'] >1) {
            q.remove();
        }
        if (q.isEmpty()) {
            Sys.o(-1+" ");
        } else {
            Sys.o(q.peek()+" ");
        }
    }
    Sys.o();
}
public static void main (String args[]) {
    String str = "aabccxb";
    printNonRepeating (str);
}
```

★ INTERLEAVE TWO HALVES OF QUEUE (even length)

→ Input : 1 2 3 4 5 6 7 8 9 10

→ Output : 1 6 2 7 3 8 4 9 5 10

* size = n = 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Original queue

* size/2 ⇒ 5

1st half — ~~2nd half~~ Original queue

F R | 1 | 2 | 3 | 4 | 5 | 1st half

F R | 6 | 7 | 8 | 9 | 10 | Original Queue

F | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | ~~5~~ | 1 | 6 | 2 | 7 | 3 | 8 | 4 | 9 | 5 | 10 | R

Original Queue

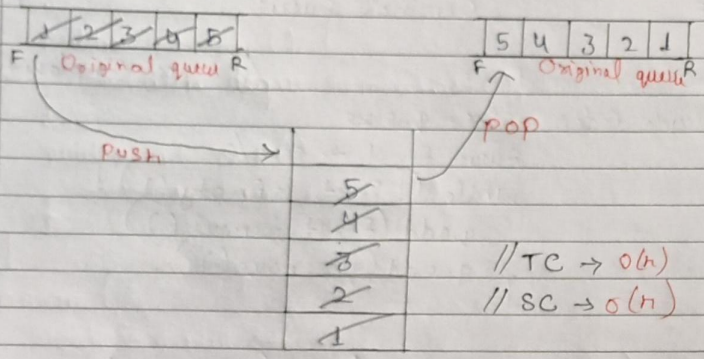
// Space Complexity, O(n)

* Pseudo Code:
```
size = q.size
Queue first → size/2
while (!first.isEmpty()) {
    q.add(first.remove())
    q.add(q.remove())
}
```

⇒
```
public static void interleave (Queue<Integer> q) {
    Queue<Integer> firstHalf = new LinkedList<>();
    int size = q.size();
    for (int i=0; i<size/2; i++) {
        firstHalf.add(q.remove());
    }
    while (!firstHalf.isEmpty()) {
        q.add(firstHalf.remove());
        q.add(q.remove());
    }
}
public static void main (String args[]) {
    Queue<Integer> q = new LinkedList<>();
    q.add(1); q.add(2); q.add(3); q.add(4); q.add(5);
    q.add(6); q.add(7); q.add(8); q.add(9); q.add(10);
    interleave(q);
    while (!q.isEmpty())
        Sys.o(q.remove()+" ");
    Sys.o();
}
```

★ QUEUE REVERSAL

Input: 1 2 3 4 5
Output: 5 4 3 2 1

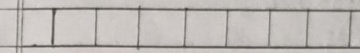


// TC → O(n)
// SC → O(n)

⇒
```
public static void reverse(Queue<Integer> q){
    Stack<Integer> s = new Stack<>();
    while(!q.isEmpty()){
        s.push(q.remove());
    }
    while(!s.isEmpty()){
        q.add(s.pop());
    }
}

public static void main(String args[]){
    Queue<Integer> q = new LinkedList<>();
    q.add(1); q.add(2); q.add(3); q.add(4);
    q.add(5);
    reverse(q);
    while(!q.isEmpty()){
        Sysout(q.remove()+" ");
    }
    Sysout();
}
```

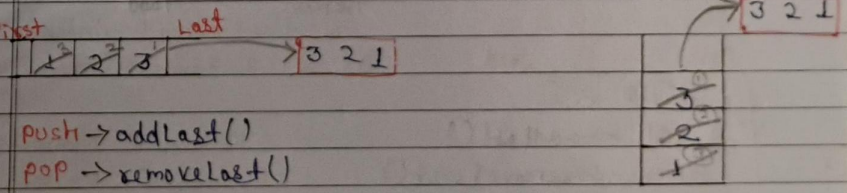

★ DEQUE – Double ended queue : allows insertion & deletion of elements from both ends.



← remove	← remove	* addFirst()
← first	← ~~First~~Last	* addLast()
← peek	← peek	* removeFirst()
← add	← add	* removeLast()
		* getFirst()
		* getLast()

⇒
```
public static void main(String args[]){
    Deque<Integer> deque = new LinkedList<>();
    deque.addFirst(1);
    deque.addFirst(2);
    deque.addLast(3);
    deque.addLast(4);
    Syso(deque);
    deque.removeLast();
    Syso(deque);
    Syso("first el = " + deque.getFirst());
    Syso("Last el = " + deque.getLast());
}
```

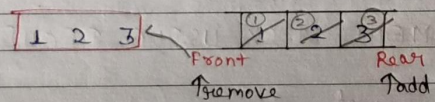
★ IMPLEMENT STACK USING DEQUE

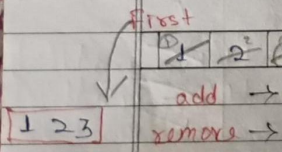


push → addLast()
pop → removeLast()
peek → getLast

⇒
```
static class Stack {
  Deque<Integer> deque = new LinkedList<>();
  public void push(int data) {
    deque.addLast(data);
  }
  public int pop() {
    return deque.removeLast();
  }
  public int peek() {
    return deque.getLast();
  }
}
psvm {
  Stack s = new Stack();
  s.push(1);  s.push(2);  s.push(3);
  Syso("peek = " + s.peek());
  Syso(s.pop());
  Syso(s.pop());
  Syso(s.pop());
}
```

★ IMPLEMENT QUEUE USING DEQUE





⇒
```
static class Queue {
  Deque<Integer> deque = new LinkedList<>();
  public void add(int data) {
    deque.addLast(data);
  }
  public int remove() {
    return deque.removeFirst();
  }
  public int peek() {
    return deque.getFirst();
  }
}
psvm {
  Queue q = new Queue();
  q.add(1);  q.add(2);  q.add(3);
  Syso("peek = " + q.peek());
  Syso(q.remove());
  Syso(q.remove());
  Syso(q.remove());
}
```

QUE1: Generate Binary Numbers – Given a number N. The task is to generate & print all binary numbers with decimal values from 1 to N.

* Sample Input 1: N=2
* Sample Output 1: 1 10

* Sample Input 2: N=5
* Sample Output 2: 1 10 11 100 101

Sol →
```
public void generatePrintBinary (int n) {
    Queue <String> q = new LinkedList <String>();
    q.add ("1");
    while (n -> 0) {
      String s1 = q.peek();
      q.remove();
      Syso (s1);
      String s2 = s1;
      q.add (s1 + "0");
      q.add (s2 + "1");
    }
}                          // Time Complexity, => O(n)
psvm {                     // Space Complexity, => O(n)
   int n = 10;
   generatePrintBinary (n);
}
```

QUE 2: Connect n ropes with minimum cost - Given are N ropes of different lengths, the task is to connect these ropes into one rope with minimum cost, such that the cost to connect two ropes is equal to the sum of their lengths.

* Sample Input 1 : N = 4, arr = [4, 3, 2, 6]
* Sample Output 1 : 29

* Sample Input 2 : N = 2, arr = [1, 2, 3]
* Sample Output 2 : 9

// Time Complexity, => O(n)
// Space Complexity, => O(n)

Sol →
```
public static int minCost (int arr [], int n) {
    PriorityQueue < Integer > pq = new PriorityQueue <Integer>();
    for (int i = 0; i < n; i++) {
       pq.add (arr [i]);
    }
    int res = 0;
    while (pq.size() > 1) {
       int first = pq.poll();
       int second = pq.poll();
       res += first + second;
       pq.add (first + second);
    }
    return res;
}
public static void main (String args[]) {
    int len [] = {4, 3, 2, 6};
    int size = len.length;
    Syso ("Total cost for connecting" + "ropes is" +
          minCost (length, size));
}
```

QUE 3: Job Sequencing Problem - We have an array of jobs where every job has a deadline & associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1.
Maximize the total profit if only one job can be scheduled at a time.

* Sample Input : JobID Deadline Profit

a	4	20
b	1	10
c	1	40
d	1	30

* Sample Output: c, a

Sol →
```
Static class Job {
    char job_id;          // TC - O(n log n)
    int deadline;         // SC - O(n)
    int profit;
    Job (char job_id, int deadline, int profit) {
        this.deadline = deadline;
        this.job_id = job_id;
        this.profit = profit;
    }
}
static void printJobScheduling (ArrayList<Job> arr) {
    int n = arr.size();
    Collections.sort (arr, (a, b) -> {
        return a.deadline - b.deadline;
    });
    for (int i=n-1; i>=-1; i--) {
        int slot_available;
        if (i==0) {
            slot_available = arr.get(i).deadline;
        } else {
            slot_available = arr.get(i).deadline -
                             arr.get(i-1).deadline;
        }
        maxHeap.add (arr.get(i));
```



```
    while (slot_available > 0 && maxHeap.size() > 0) {
        Job job = maxHeap.remove();
        slot_available --;
        result.add (job);
    }
}
Collections.sort (result, (a, b) -> {
    return a.deadline - b.deadline;
});
for (Job job : result) {
    Sysp (job.job_id + " ");
}
Syo ();
}
public static void main (String[] args) {
    ArrayList <Job> arr = new ArrayList <Job>();
    arr.add (new Job ('a', 2, 100));
    arr.add (new Job ('b', 1, 19));
    arr.add (new Job ('c', 2, 27));
    arr.add (new Job ('d', 1, 25));
    arr.add (new Job ('e', 3, 15));
    Syso ("Following is maximum " + profit sequence of Jobs);
    printJobScheduling (arr);
}
```

QUE4: Reversing the first K elements of a Queue – We have an integer k & a queue of integers, we need to reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

* Sample Input 1 : Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], K = 5
* Sample Output 1: Q [50, 40, 30, 20, 10, 60, 70, 80, 90, 100]

Sol →
```
static class cell {
    int x, y;
    int dis;
    public cell (int x, int y, int dis) {
        this.x = x;
        this.y = y;          //TC → O(n+k)
        this.dis = dis;      // SC → O(k)
    }
}
static boolean isInside (int x, int y, int N) {
    if (x >= 1 && x <= N && y >= 1 && y <= N) {
        return true;
    }
    return false;
}
static int minStepToReachTarget (
    int knightPos [], int targetPos [], int N) {
    int dx [] = {-2, -1, 1, 2, -2, -1, 1, 2};
    int dy [] = {-1, -2, -2, -1, 1, 2, 1, 2};
    Vector <cell> q = new Vector <>();
    q.add (new cell (knightPos [0], knightPos [1], 0));
    cell t;
    int x, y;
    boolean visit [][] = new boolean [N+1] [N+1];
    visit [knightPos [0]] [knightPos [1]] = true;
    while (!q.isEmpty()) {
        t = q.firstElement;
        q.remove (0);
        if (t.x == targetPos [0] && t.y == targetPos [1])
            return t.dis;
        for (int i=0; i<8; i++) {
```



```
            x = t.x + dx [i];
            y = t.y + dy [i];
            if (isInside (x, y, N) && !visit [x] [y]) {
                visit [x] [y] = true;
                q.add (new cell (x, y, t.dis + 1));
            }
        }
    }
    return Integer.MAX_VALUE;
}
public static void main (String [] args) {
    int N = 30;
    int knightPos [] = {1, 1};
    int targetPos [] = {30, 30};
    Syso (minStepToReachTarget (knightPos, targetPos, N));
}
```

Que 5: Maximum of all subarrays of size k – We have an array arr [] of size N & an integer K. Find the maximum for each & every contiguous subarray of size K.

* Sample Input 1: N=9, K=3, arr = 1 2 3 1 4 5 2 3 6
* Sample Output 1: 3 3 4 5 5 5 6

Sol →
```
public static void printMax (int arr [], int n, int k) {
    Dequeue <Integer> Qi = new LinkedList <Integer> ();
    int i;
    for (int i=0; i<k; ++i) {
        while (!Qi.isEmpty () && arr [i] >= arr [Qi.peekLast])
            Qi.removeLast ();
        Qi.addLast (i);
```

```
for (; i<n; ++i) {
    System.out.print (arr [Qi.peek()] + " ");
    while ((!Qi.isEmpty()) && Qi.peek() <= i-k){
        Qi.removeFirst();
    while ((!Qi.isEmpty()) && [arr [Qi.peekLast()])
        Qi.removeLast();
    Qi.addLas(i);
}
System.out.print(arr [Qi.peek()]);
}
psvm {
    int arr [] = {12, 91, 78, 90, 57, 89, 56};
    int k = 3;
    printMax (arr, arr.length, k);
}
```