→ Knight tours → Jumps → multiple calls

wrong order less calls base → last value

r-2, c-1 →

r-2, c+1 →

→ Time Complexity in recursion.

Queue → _____ 4 four

DS
- → LinkedList
- → Hashmap
- → Two pointers
- → Sliding window / Prefix - Suffix
- → Bit manipulation
- → Trees
  - → D.P
  - → Graph

Class | Object | Constructor.

↳ Saturday → Trees → Binary Tree, Binary Search Tree
↳ Sunday ↗ ↑

Queue

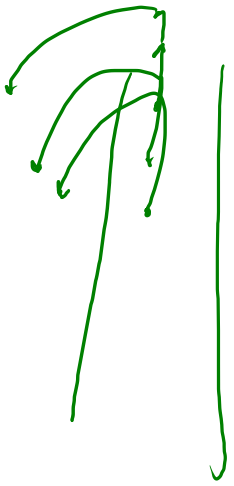
↳ FIFO

(First In First Out)

Stack.

↳ LIFO

(Last In First out)



⇒ Syntax :-

Queue < datatype > queue_name = new Queue <>();

↑
Interface

⇒ ~~Queue < datatype > queue_name = new LinkedList<>();~~

or

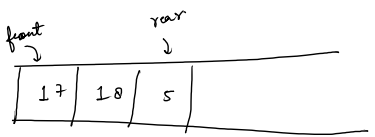
⇒ Queue < datatype > queue_name = new ArrayDeque <>();

Ex :- Queue < Integer > q = new LinkedList <>();

* front → removing the elements → dequeue

rear → adding the elements ⇒ enqueue

→ 12, 15, 17, 18 , dequeue, dequeue, 5



null

⇒ peek → element present at the front position.

⇒ (17)

add → add

q.add(ele); → rear

peek

q.peek()

remove / poll

q.remove() → front

q.poll() → front

size →

q.size()

Using ArrayDeque

```java
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN. Print outp
    Queue<Integer> q = new ArrayDeque<>();

    q.add(1);
    q.add(3);
    q.add(7);

    System.out.println(q);
    q.remove();
    System.out.println("Size of queue: " + q.size());
    System.out.println(q);
    System.out.println("Peek element is: " + q.peek());
    q.poll();
    System.out.println("Size of queue: " + q.size());
    System.out.println(q);
}
```

Interface

5 - 6 min's

Using LinkedList

```java
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN. Print outp
    Queue<Integer> q = new LinkedList<>();

    q.add(1);
    q.add(3);
    q.add(7);

    System.out.println(q);
    q.remove();
    System.out.println("Size of queue: " + q.size());
    System.out.println(q);
    System.out.println("Peek element is: " + q.peek());
    q.poll();
    System.out.println("Size of queue: " + q.size());
    System.out.println(q);
}
```

Time Needed to Buy Tickets (Day 37)

Problem | Submissions | Leaderboard | Discussions

There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (n - 1)th person is at the back of the line.

You are given a 0-indexed integer array tickets of length n where the number of tickets that the ith person would like to buy is tickets[i].

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

Return the time taken for the person at position k (0-indexed) to finish buying tickets.

Sample Input 0

```
4
5 1 1 1
0
```

Sample Output 0

```
8
```

n = 4

tickets = [5, 1, 1, 2] at indices 0 1 2 3

1 sec → 1 buy

[4, 1, 1, 4] K = 3

[3, 1, 2, 3] at indices 0 1 2 3 → 1 sec

[1, +, +, +]

+ + + + + + + + = 8

[3, 1, 1, 0] at indices 0 1 2 3 k = 0

8

ans = 0

9

5 - 1 = 4

1 - 1 = 0 4 - 1 = 3

1 - 1 = 0 3 - 1 = 2

1 - 1 = 0 2 - 1 = 1

1 - 1 = 0 = = 0

⇒ 8

tickets [0, 1, 2, 3, 6] at indices 0 1 2 3 4

k = 2

ans = 7

return ans.

7

index = 0 tickets[0] = 2 - 1 = 1

index = 1 tickets[1] = 3 - 1 = 2

index = 2 tickets[2] = 2 - 1 = 1

index = 3 tickets[3] = 4 - 1 = 3

index = 4 tickets[4] = 7 - 1 = 6

index = 0 tickets[0] = 1 - 1 = 0

index = 1 tickets[1] = 2 - 1 = 1

index = 2 tickets[2] = 1 - 1 = 0

9

[3, 1, 0, 1, 2, 3, 4, 1]

5-6 wins

```
public static int tntbt(int tickets[], int n, int k){
    Queue<Integer> q = new LinkedList<>();

    for(int i=0;i<n;i++){
        q.add(i);
    }

    int ans = 0;
    while(q.size()>0){
        int index = q.poll();

        tickets[index]--;
        ans++;

        if(tickets[index] == 0 && index == k){
            return ans;
        }

        if(tickets[index] != 0){
            q.add(index);
        }
    }

    return ans;
}
```

```
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN.
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int tickets[]= new int [n];
    for(int i=0;i<n;i++){
        tickets[i] = scn.nextInt();
    }

    int k = scn.nextInt();

    System.out.println(tntbt(tickets,n,k));
}
```

232. Implement Queue using Stacks

Easy 6.6K 376

Companies

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

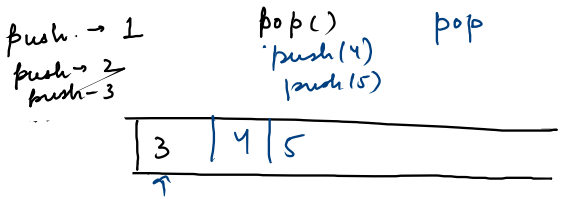
- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

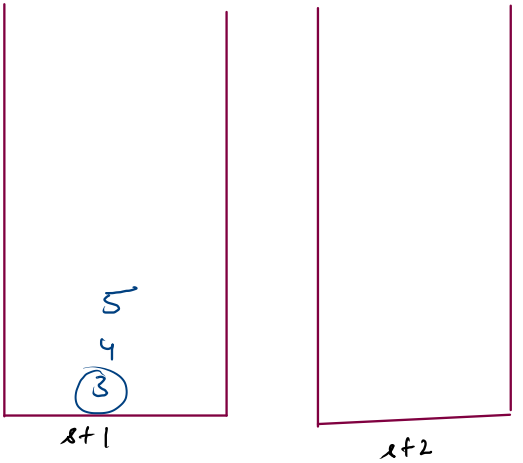
Notes:

- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]
```





push → st1.push

pop() → 1) push all the elements into the st2

2) st2.pop()

3) push all the elements into the st1.

peek() → similar to pop()

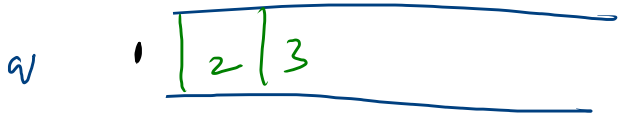
empty() → st1.size() > 0 →

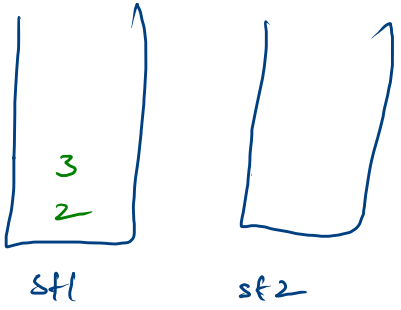
```java
class MyQueue {
    Stack<Integer> st1;
    Stack<Integer> st2;
    public MyQueue() {
        st1 = new Stack<>();
        st2 = new Stack<>();
    }

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

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

public int pop() {
    while(st1.size()>0){
        st2.push(st1.pop());
    }
    int val = st2.pop();

    while(st2.size()>0){
        st1.push(st2.pop());
    }

    return val;
}
```

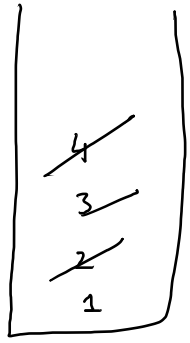
```java
public int peek() {
    while(st1.size()>0){
        st2.push(st1.pop());
    }
    int val = st2.peek();

    while(st2.size()>0){
        st1.push(st2.pop());
    }

    return val;
}

public boolean empty() {
    if(st1.size()!= 0) return false;
    else return true;
}
```

remove | pop()





Implement stack using queue.



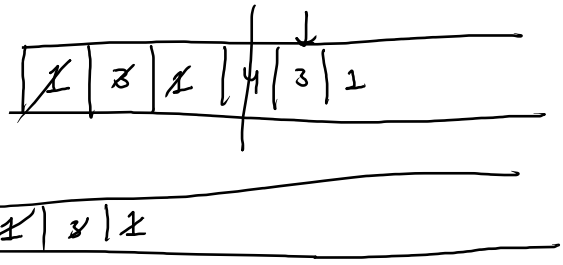
st

push → 1
→ 2

push → 3
push 4

pop()

front



peek()

empty.

q1.size() > 0 → false / true

Implement stack using queue

```
Queue<Integer> q1;
Queue<Integer> q2;
public MyStack() {
    q1 = new LinkedList<>();
    q2 = new LinkedList<>();
}

public void push(int x) {
    while(q1.size() >0){
        q2.add(q1.poll());
    }

    q1.add(x);

    while(q2.size()>0){
        q1.add(q2.poll());
    }
}

public int pop() {
    return q1.poll();
}
```

```
public int top() {
    return q1.peek();
}

public boolean empty() {
    if(q1.size()>0) return false;
    return true;
}
```