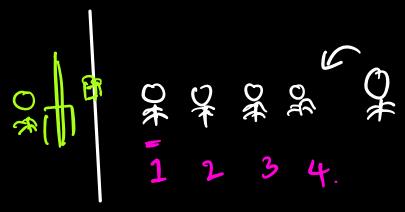
Today's content.

* Introduction to queues
* Queue Implementation using LL
* Reverse first k elements in queue
* Queue Implementation using stacks
* Generate k^{th} number in series using 1 4 2.

Queue Introduction



FIFO → First in first out.

Operations:

O(1), TC.

(i) Enqueue(x) : insert at the rear end of queue.

(ii) deque() : delete an ele from the front of queue.

(iii) front() : Return the ele at the front.

(iv) rear() : Return the ele at the rear.

Ex:

rear() → 15

enqueue(10) → add.

dequeue().

front() → 9.

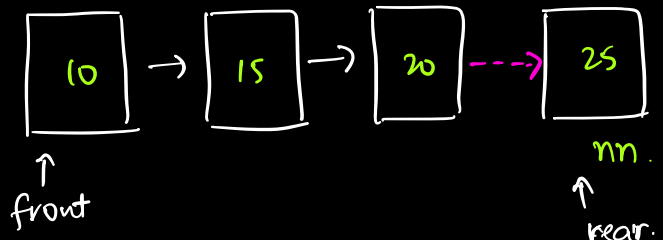
Ex from quiz:

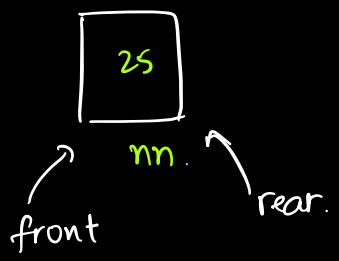
3 7 12 8 3.

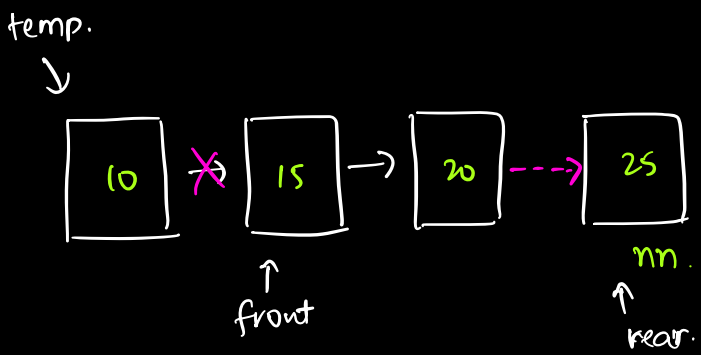
Q1: Implement a custom queue class using linked list.

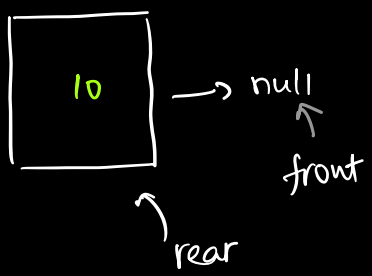
```
class Node
    int data
    Node next
```

```
class Queue
    Node front
    Node rear
    int size

    boolean isEmpty()
        return front==null

    Void enqueue(data)
        Node nn = new Node(25)
        size = size+1
        if (rear=null)
            front=nn , rear=nn
            return
        rear.next=nn
        rear=nn

    int dequeue()
        if (isEmpty())
            return -1
        size = size-1
        Node temp=front
        front = front.next
        if (front==null)
            rear=null
        temp.next=null
        return temp.data
```



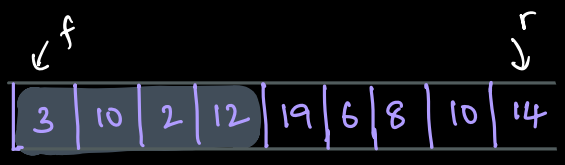




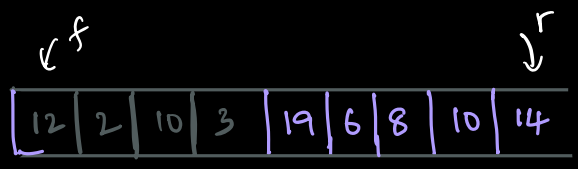


Q2: Given a queue, **Reverse its first k elements** using **1 stack**.

Ex1:

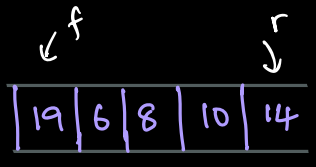


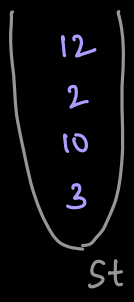
, k=4



Ideas:

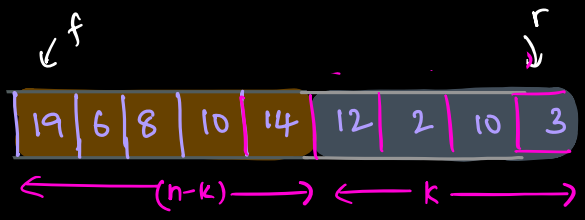
Step 1: Take first 'k' elements from queue and push to stack.

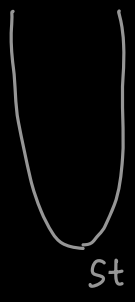




O(k)

Step 2: Pop the elements from stack, add to rear of 'q'.





O(k)

Step 3: Dequeue and enqueue (n-k) elements from q.

O(n-k)

12 2 10 3 19 6 8 10 14

TC: $O(k) + O(k) + O(n-k) \simeq O(n+k) \simeq O(n)$ ($\because k \leq n$).

SC: $O(k)$.

38. Implement queue using stacks.

Queue operations

i) Enqueue(ele)
ii) Deque()
iii) front()
iv) rear()

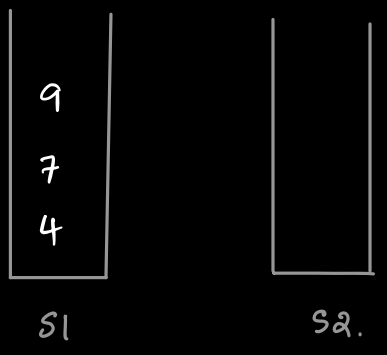
use only stacks.
Expected TC : O(1) in avg case. ✗

stack.

i) push()
ii) pop()

Attempt 1:

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21
5



idea:

✓(i) enqueue(x) → add x to S1; O(1)

✓(ii) dequeue(); O(N). (Issue!).

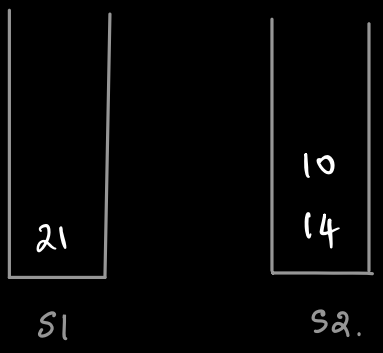
✓ * push all elements from S1 → S2.

✓ * pop the element from S2.

* move all elements from S2 → S1.

Attempt 2:

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21
5 4 7 9 8



(i) enqueue(x) → add x to S1; O(1)

(ii) dequeue(); TC: O(N), Avg TC: O(1).

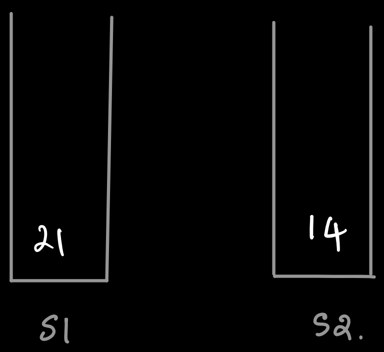
* if (S2.isEmpty())

* push all elements from S1 → S2.

* pop the element from S2.

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21 [deq, deq

(deq() results: 5, 4, 7, 9, 8)



1^{st} deq : All elements from S1 → S2 + Read top

4 operations | 1 operation.

2^{nd} deq : Top of S2

1 operation.

3^{rd} deq : Top of S2

1 operation.

4^{th} deq : Top of S2

1 operation.

Total deque = 4 , Total operations = 8.

On an average, for a single deque = 2 operations.

Avg time complexity of deque = O(1).

Amortized time complexity: Avg time complexity for a function.

Doubts.

5^{th} deq → 4 operations (3 + 1).

6^{th} deq → 1 operation

7^{th} deq → 1 operation.

3 deq = 6 operation, 1 deq = 2 operation.

avg deq = O(1).

When to use avg time complexity?

When your function is working in O(1) time for every input, except for a corner case.

Prompt for queue.

Q1: Generate k^{th} number in series using digits 1 and 2 only.

k = 5, Series : [1, 2, 11, 12, 21]

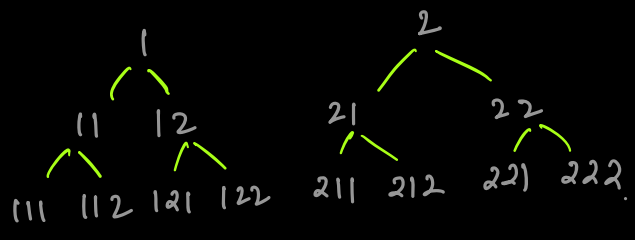
k = 7, series : [1, 2, 11, 12, 21, 22, 111]

k = 10,

1 digit :

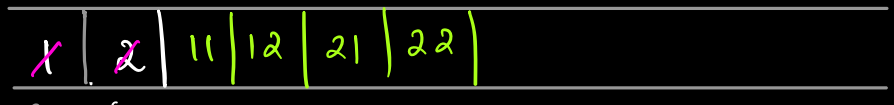
2 digit :

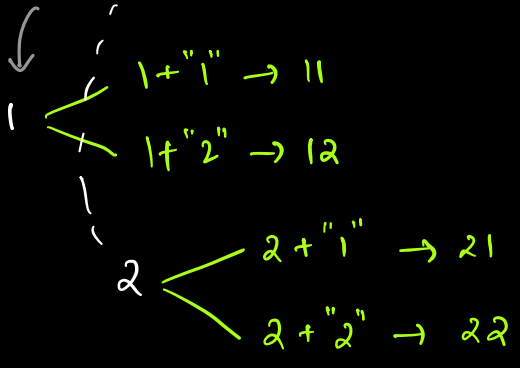
3 digit :



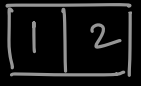
k^{th} level is derived from $(k-1)^{th}$ level ⇒ Always use queues.

queue →



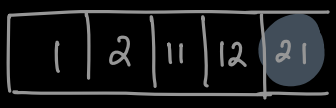


How many times enque & deque?

Que : 

→ For (k-1) times, delete an element & append 2 elements.

→ Ans is at the front of queue.



,

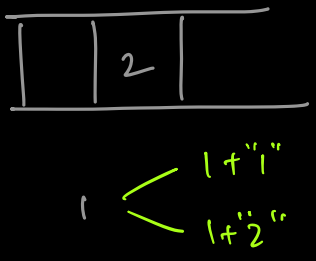
k = 1, ans = 1, no. of deque = 0

k = 2, ans = 2, no. of deque = 1

k = 5, ans = 21, no. of deque = 4.

k-1

```
String  Kth Number ( int  K)
    Queue < String >   q ;
    q.enque("1"), q.enque("2") ;
    for (i= 1 ; i<k ; i++)
        String  s= q.deque() ;
        q.enqueue ( s+"1")
        q.enqueue (s+"2")
    return  q.front()
```



For python,