Queues

Do something today that your future self will thank you for.

Good Morning

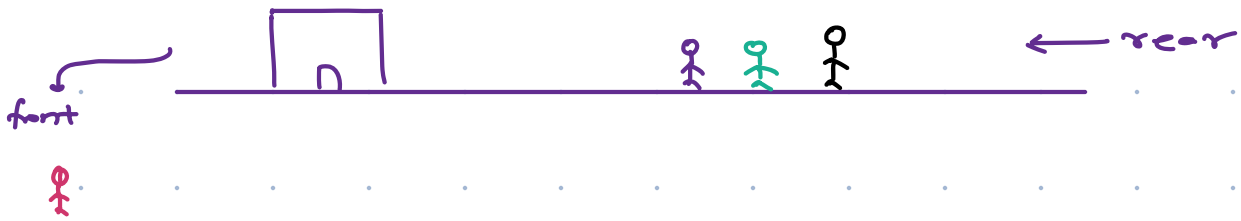
Today's content

01. Introduction to Queues
02. Implementation of Queues
    - → Using Arrays
    - → Using LinkedList
    - → Using stacks
03. Introduction of Deque.
04. Max of every window

Sliding Window Maximum

Queue → Linear data structure where elements are inserted from one end & removed from the other end

⇒ First In First Out



Some real world Examples

01. Ticket counter
02. Message queue
03. Token entry
04. Call center
05 playlist Queue

* Common operations in Queue

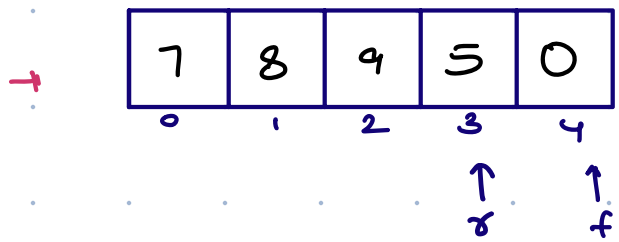
01. Add(x)/Enqueue(x) → Add x inside queue from last
02. Remove()/Deque() → Remove first ele present in queue
03. top() / peek() → return first ele present in queue
04. size() → no. of elements inside queue

Implementation of Queue

01. Array

$f = -1 \Rightarrow$ idx of last ele which was removed

$r = -1 \Rightarrow$ idx of last ele which got inserted



if (f == r) → q is empty

↳ q is full

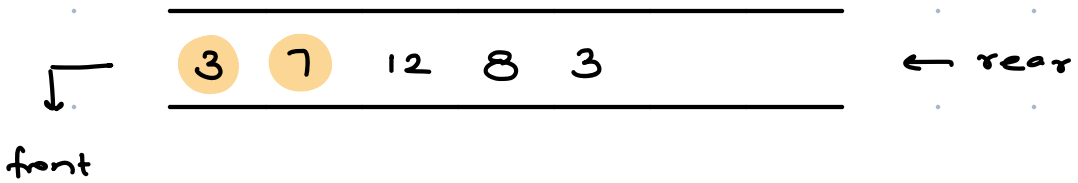
Add(5)	remove() → 5	add(6)	remove() → 6
Add(4)	remove() → 4	add(7)	top() → 7
Add(3)	remove() → 3	add(8)	
Add(2)	remove() → 2	add(9)	
		add(5)	

```
class Queue {
    int sz = 0;
    int [] q;
    int f = -1, r = -1;
    Queue() {
        q = new int [5]
    }
    int size() {
        return sz;
    }
    int top() {
        if (sz == 0) return -1;
        int idx = (f+1) % N
        return A[idx];
    }
}
```

```
void add ( val )
{
    if (sz == N) return;
    r = (r + 1) % N
    A[r] = val;
    sz++;
}
```

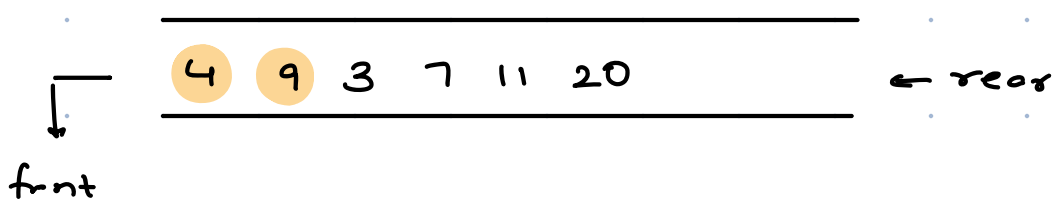
```
int remove ( )
{
    if (sz == 0) return -1;
    f = (f + 1) % N
    int val = A[f]
    A[f] = 0;
    sz = sz - 1;
    return val;
}
```

What will be the state of the queue after these operations enqueue(3), enqueue(7), enqueue(12), dqueue(), dqueue(), enqueue(8), enqueue(3)



print = 12 8 3

What will be the state of the queue after these operations enqueue(4), dqueue(), enqueue(9), enqueue(3), enqueue(7), enqueue(11), enqueue(20), dqueue()



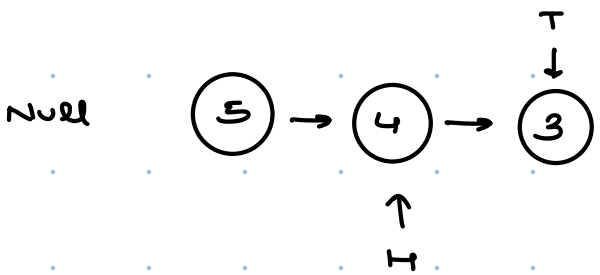
Output = 3 7 11 20

* Using LinkedList

Insert at tail
remove at head
} TC: O(1)

Add(5) Add(3)

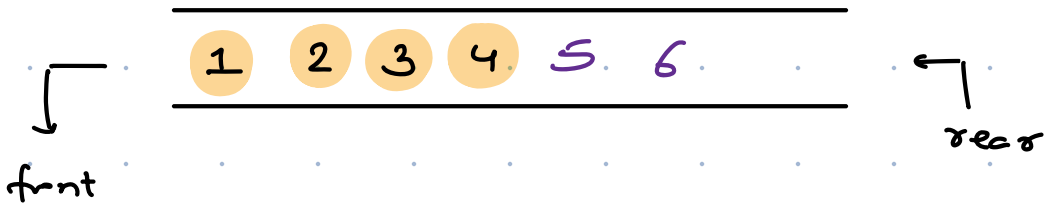
Add(4) remove()



* Implement queue using Stacks

→ push → peek

→ pop → size



Add(1)

Add(2)

Add(3)

Add(4)

remove()

remove()

Add(5)

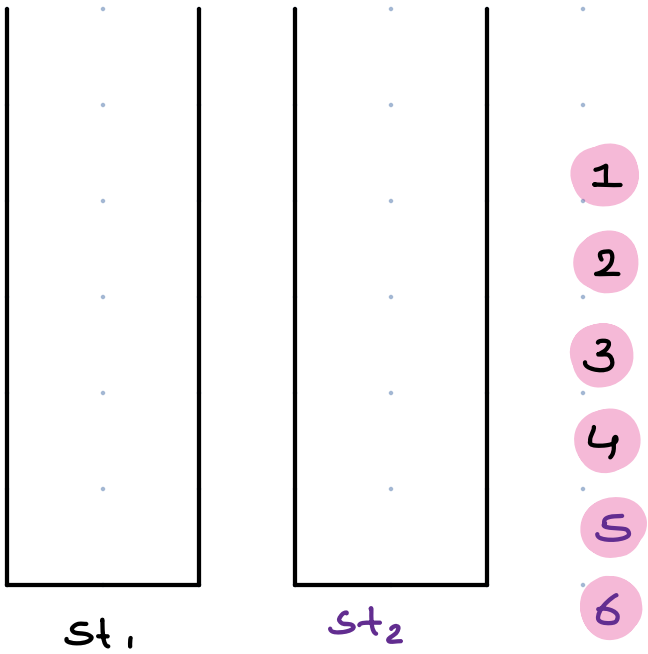
Add(6)

remove()

remove()

remove()

remove()



add(x) → st_1.push(x) → TC: O(1)

remove() → if (st_2.size() == 0)

Yes → move all ele from st_1 to st_2 & then pop

No → st_2.pop(); → TC: O(1)

size() → st_1.size() + st_2.size(); → TC: O(1)

peek() → similar to remove function without popping

Add(1)

Add(2)

Add(3)

Add(4)

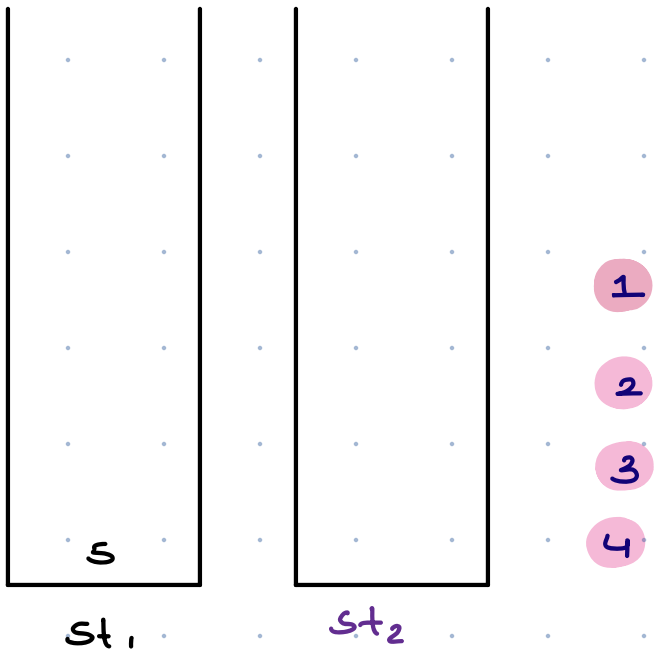
remove()

remove()

Add(5)

remove()

remove()



1st rem = Move all elements from st_1 to st_2 & then $st_2.pop()$;

8 ops + 1 ops = 9 ops

2nd rem = $st_2.pop()$ → 1 ops

3rd rem = $st_2.pop()$ → 1 ops

4th rem = $st_2.pop()$ → 1 ops

4 removals = 12 ops

1 removal = $\frac{12}{4}$ ops ≈ 3 ops

≈ O(K) ⇒ O(1)

Doubly Ended Queue → Doubly Linked List

→ Allows insertion & removal from both the ends

01. Addfirst( )
02. Addlast( )
03. getfirst( )
04. removefirst( )
05. removelast( )
06. getlast( )

TC: O(1)

* Sliding Window Maximum

Given an integer[] A & a window of size K
Find the maximum of all the windows

A[] = {10 1 9 3 7 6 5 11 8} K = 4

Ans = {10 9 9 7 11 11}

A[] = {1 4 3 2 5} K = 3

Ans = 4 4 5

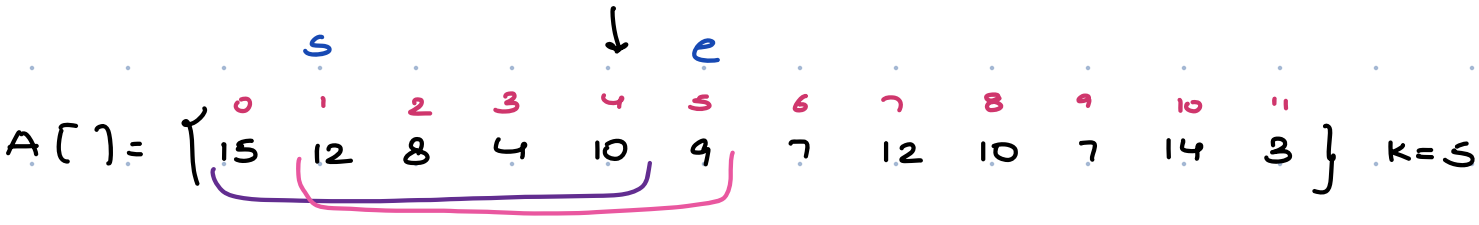
Brute force → Generate all subarrays of size K, iterate on subarr & get maximum

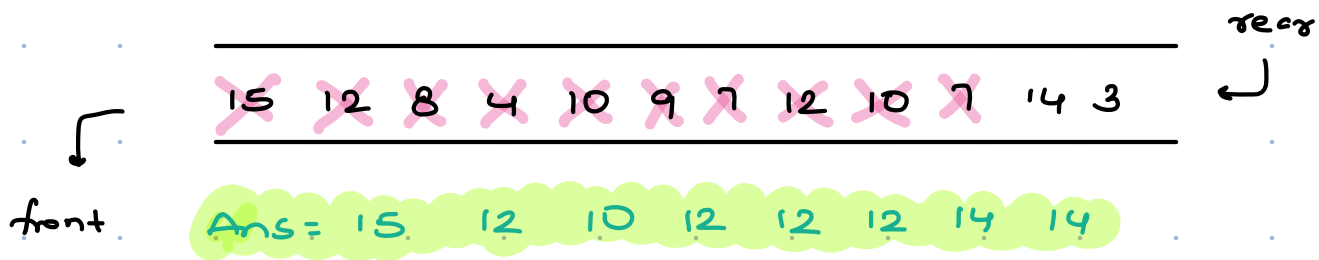
TC : O(N-K+1) * K ⇒ **O(N²)**

⇓

K = n/2

Optimal





→ Insert at last & delete from last

→ deletefirst( )

→ accessfirst( ).

```
int [ ] maxwindow ( int [ ] A , int K )
    int idx = 0
    int [ ] ans = new int [ N-K+1 ]
    Deque < I > dq = new ArrayDeque <>( );
```

```
for ( i=0; i<k; i++ ) {                    // insert first window
    while ( dq.size()>0 && A[i] > dq.getlast() )
        dq.removelast();
    dq.addlast(A[i]);
}
ans[idx++] = dq.getfirst();

s=1 , e=k

while (e<n)
    if ( A[s-1] == dq.getfirst() )
        dq.removefirst();                  } removal
    while ( dq.size()>0 && A[e] > dq.getlast() )
        dq.removelast();
    dq.addlast(A[e]);                      } Insertion
    ans[idx++] = dq.getfirst();
    s++ , e++ ;

return ans;
}
```

Scenario:

At TechTrade Inc., a trading team focuses on day-trading technology stocks. They employ a strategy that involves selling stocks at short-term peaks to maximize profits. The team uses an algorithm to determine the best time to sell stocks based on minute-to-minute price data.

Stock Prices Array A
Consider the minute-by-minute stock prices of a tech company, TechCorp, over a 10-minute interval:

A=[220,215,230,225,240,235,230,245,250,240]
0 1 2 3

```
Queue<I> q = new LinkedList<>();
```