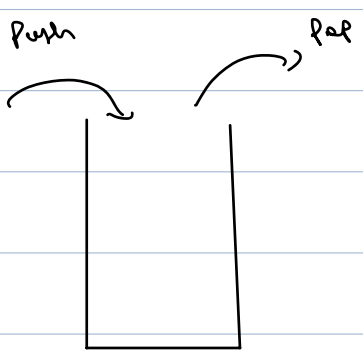
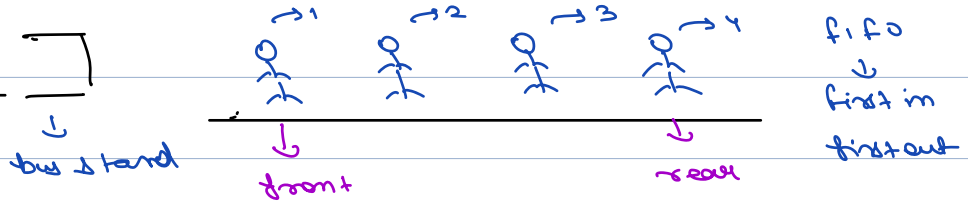
Today's Content

- Queue
- Implementation of the queue using array
- Implementation of the queue using stack
- Perfect Number Question
- Doubly ended queue
- Sliding Window Maximum



Stack → push(x)
→ pop()
→ peek()
→ isEmpty()

Queue



functions of Queue.

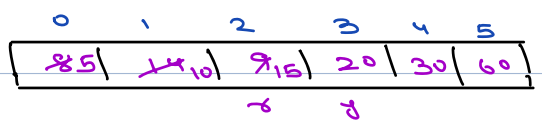
1) Enqueue (x) :- it will enter at the rear end.
2) Deque () :- remove an element from front end.
3) front () :- gives you element at front end.
4) rear () :- gives you element at rear end.

Implementation of Queue ()

1) Arrays :-

8, 14, 9, 20, ↑, 30, front(), ↑, rear(), 60, 9, 5, 10
14 15

8 14 9 20 30 60 5 10 15



front = -1 0 1 2 3

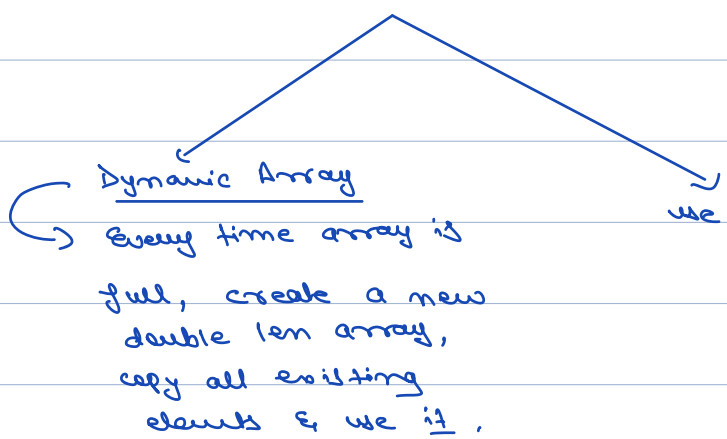
rear = -1 0 1 2 3 4 5 6 7 8

```
int front = -1;
int rear = -1;
int size = 0;
```

```
enqueue(x) {
    if( rear==-1 ) { rear=0; front=0; arr[rear]=x; size++; return; }
    if ( size == arr.len ) { (or (r+1)%n == front)
        return Queue is full;
    }
    rear++; rear = rear%n; size++;
    arr[rear] = x;
}
```

```
dequeue() {
    if (size==0) { return "Queue is Empty" }
    temp = arr[front];
    front++;
    front = front%n ; size--;
    return temp;
}
```

Problem :- Queue len is fixed.



Implementation via linked list :-

LL → ① addfirst() → O(1)

(assuming tail)

② addlast() → O(1)

③ removefirst() → O(1)

④ removelast() → O(n)

8 14 9 20 ↑ 30 front() ↑ rear() 60 7 5 10

8 14 9 20 30

9 → 20 → 30

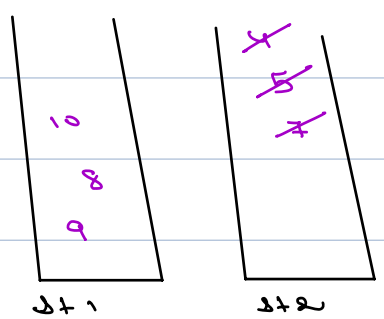
Ques) Implement a Queue using Stack.

Enqueue
Dequeue

Data

5 4 7 9 ↑ 8 10 ↑ ↑ 14 ↑ ↑ 21

5 4 7 9 8 10



idea 1 :- O(1)

Enqueue(x) :- Push x in stack 1

Dequeue() → Transfer all elements from st1 → st2

O(n)

remove top element of st2

Transfer all elements back, st2 → st1

Idea 2 :-

5 4 7 9 7 8 10 7 7 14 7 7 21 7 13 14

5 7 7 9 8 10 14

idea :- → O(1)

① Enqueue (x) :- push x in stack 1

② Dequeue () :-

if (S2.size() == 0) {

Transfer all S1 → S2 .

}

S2.pop();

Best case

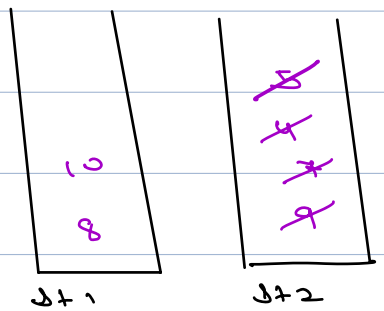
O(1)

Worst case

O(n)

5 4 7 9 ? 8 10 ? ? 14 ? ? 21 ↑ 13 14

5 * * 9 8 10



1st deq() → Transfer 4 elements from st1 → st2
st2.pop() → 9 oper.

2nd deq() → 1 oper
3rd deq() → 1 oper
4th deq() → 1 oper

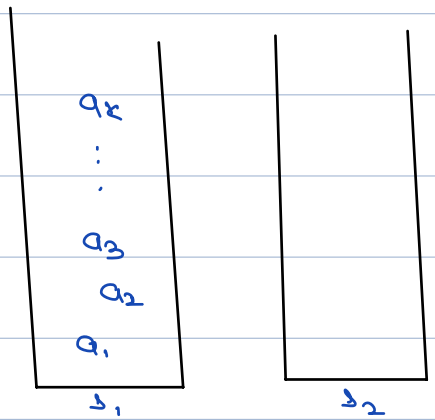
4 deq() → $\frac{\text{Total operations}}{4} = \frac{12}{4} =$ 3 oper.

Enqueue → O(1).

dequeue → amortized O(1).

// generalized

$q_1, q_2 \ldots q_k$, deq()



Transfer $s_1 \to s_2$
s_2.pop()

1st deq() → $2k+1$

2nd deq() → 1 oper

3rd deq() → 1 oper

⋮

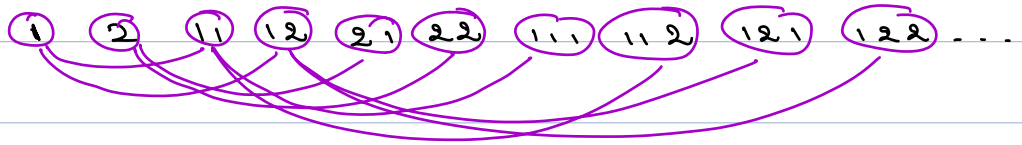
kth deq() → 1 oper

$$\text{avg. dequeue cost} = \frac{\overbrace{2k+1+1+1\ldots}^{k}}{k} \Rightarrow \frac{2k+k}{k}$$

$$\Rightarrow \frac{3k}{k} \Rightarrow 3 \text{ opers}.$$

find nth perfect number

↳ numbers using digit 1 or 2



k = 9 → 121

k = 3 → 11

k = 6 → 22

Brute force :- ∀ natural numbers, check if it has digit 1 or 2. → find kth no.

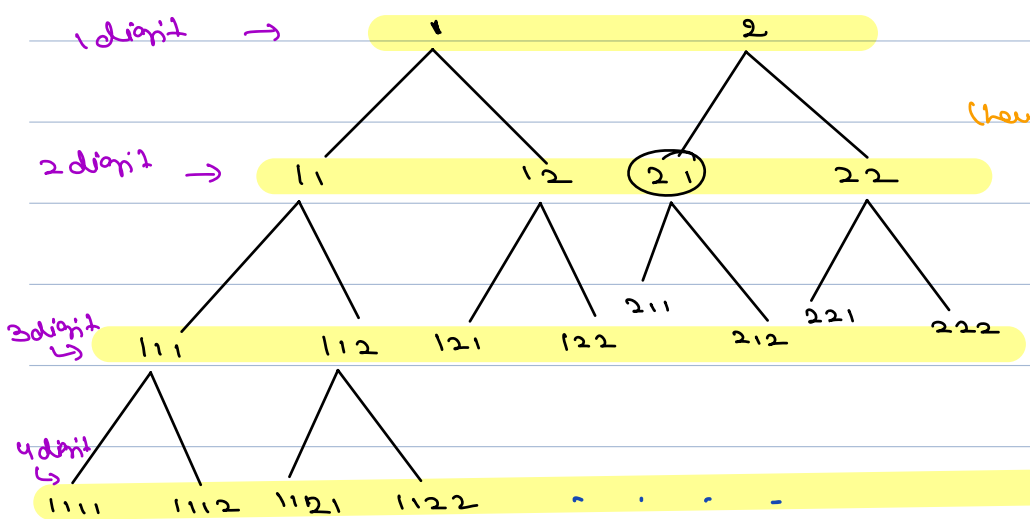
Recursion → DFS.

(BFS) (Breadth first search)

↓

Queue.

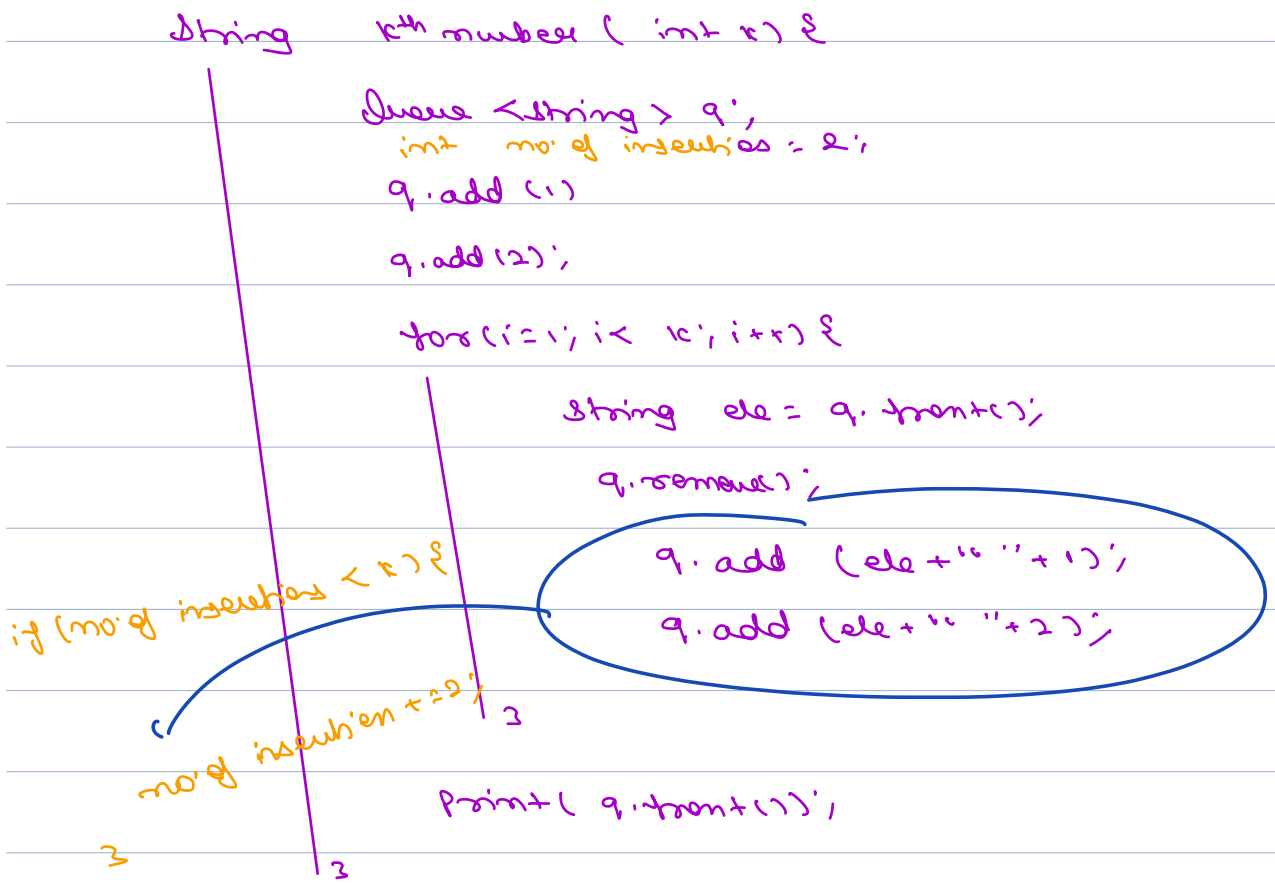
(Level Order Traversal)



k = 6

no. of iteration = 2 × 6

✓ ✗ ✗ ✗ ✗ ✓

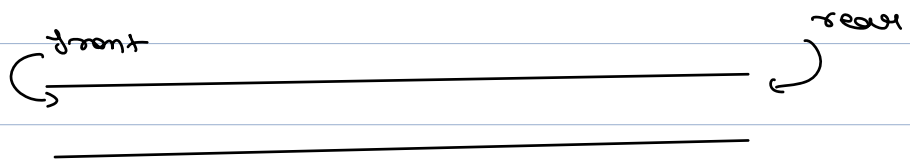

String kth number (int k) {
Queue <String> q;
int no. of insertions = 2;
q.add (1)
q.add (2);
for (i=1; i< k; i++) {
String ele = q.front();
q.remove();
q.add (ele + "" + 1);
q.add (ele + "" + 2);
if (no. of insertions < k) {
no. of insertion += 2;
}
}
Print (q.front());
}

T.C → O(k)

S.C → O(k)

→ Doubly Ended Queue (Deque)

Doubly Ended Queue (Deque)



insert_rear();

remove_rear();

rear();

insert_front();

remove_front();

front();

Ques **Sliding Window Maxm.**

Given arr[n] & k, print max element in every window of size k>1

arr[9] = 10 1 9 3 7 6 5 11 8, k=4

Ans → 10 9 9 7 11 11

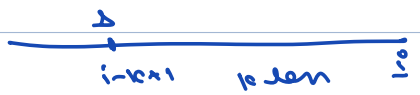
Brute force :- ∀ window of size k, Travel & find Maxm.

T.C → O(n-k+1) * k

↳ when k = n/2 → O(n²).

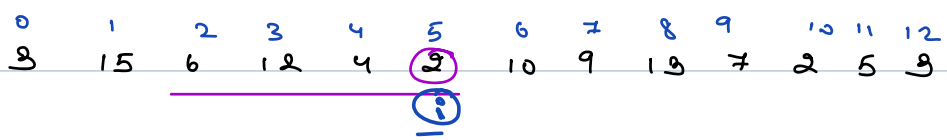
↓ no. of subarrays of size k,

s → i = k ⇒ i - s + 1 = k ⇒ s = i - k + 1



5 - 4 = 1

i - k



15 12 4 2
1 3 4 5

k = 4.

arr[] =

0	1	2	3	4	5	6	7	8	9	10	11	12
3	15	6	12	4	2	10	9	13	7	2	5	9

15 15 12 12 10 13 13 13 13 7

3 15 6 12 4 2 10 9 13 7 2 5 9

removing from start

removing from back

8 - 4 = 4

(i - 10)

0	1	2	3	4	5	6	7	8
3	2	3	4	5	5	4	5	2

k = 4.

(1)

3 2 3 4 5 5 4 5 2

4 5 5 5 5 5

5

5 4 3

Dequeue();

```
for (i → 0 to k-1) {
    while (!q.isEmpty() && A[q.rear()] != A[i]) {
        q.removeRear();
    }
    q.insertRear(i);
}
print(q.front());
```

```
for (i → k to n-1) {
    while (!q.isEmpty() && A[q.rear()] != A[i]) {
        q.removeRear();
    }
    q.insertRear(i);
    if (q.front() == i-k) {
        q.removeFront();
    }
    print(q.front());
}
```

T.C → O(n)
S.C → O(k).