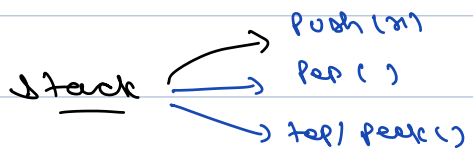
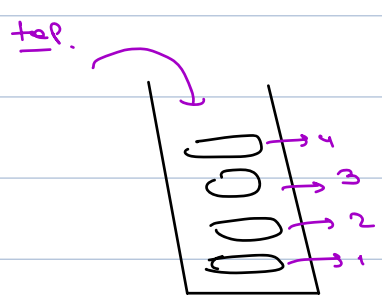
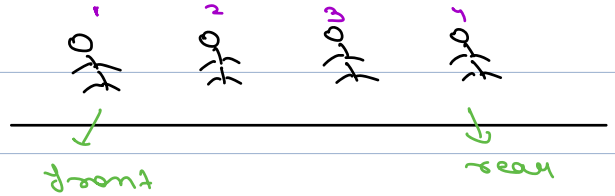
Today's Content

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





Queue



FIFO
↓
first in first out.

Functions of Queue.

1) Enqueue (x) :- x 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() → 14, ↑, rear() → 30, 60, ↑, 5, 10
19, ↑, ↑, ↑

~~8 14 x 20 30 60~~ 5 10 19

0	1	2	3	4	5
~~85~~	~~14~~ 10	~~x~~ 19	20	30	60
y		x			

front = ~~0 x x~~ 3 x 5 0

rear = ~~-1 0 x x~~ 3 x 5 ~~0~~ x 2

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

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

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

Problem :- Queue len is fixed.

- Dynamic array
  - Every time array is full, create a new double len array, copy all the existing elements & use it.
- Linked list

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, ↑, 5 10

14

~~8~~ ~~14~~ 9 20 30

9 → 20 → 30

H T

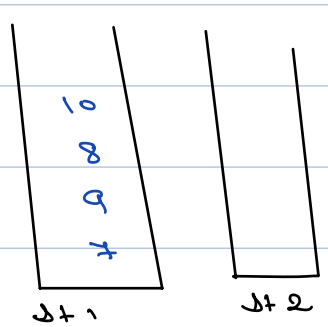
Enqueue
Dequeue

Ques) Implement a Queue using Stack.

Data

5 4 7 9 ↑ 8 10 ↑ ↑ 14 ↑ ↑ 21

~~5~~ ~~4~~ 7 9 8 10



Idea :-

Enqueue(x) :- push x into stack 1

dequeue() :- Transfer all elements from $S_1 \to S_2$

O(n) remove top element from S_2.

Transfer back from $S_2 \to S_1$

Idea 2 :-

5 4 7 9 ↑ 8 10 ↑ ↑ 14 ↑ ↑ 21 ↑ 13 14

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

Idea 1 :-

Enqueue (x) :- push x into stack 1.

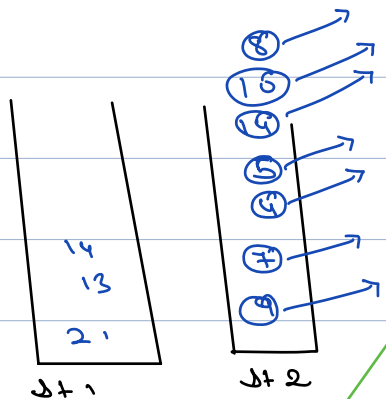
dequeue () :-

if (st2.size == 0) {

Transfer from $s_1 \rightarrow s_2$

s_2.pop();

}



Best case

O(1)

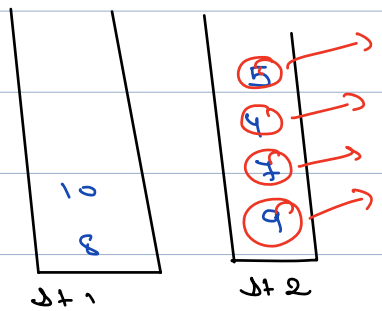
worst case

O(n)

1st deq

5 4 7 9 1 8 10 1 1 14 1 1 21 1 13 14

~~5~~ ~~4~~ ~~7~~ 9 8 10



1st deq → Transfer 4 elements from $S_1 \rightarrow S_2$ + S_2.pop();

9 operations

2nd deq → 1 operation

3rd deq → 1 operation

4th deq → 1 operation

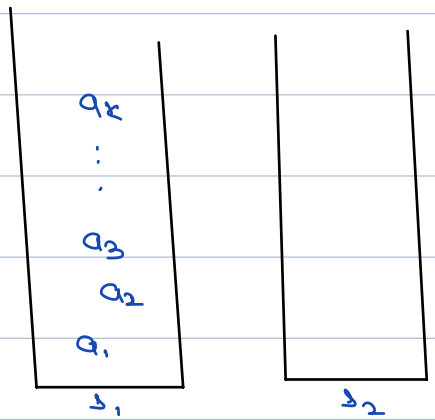
4 deque → $\frac{\text{Total Operations}}{4} = \frac{12}{4} = 3$

Enqueue → O(1).

dequeue → amortized O(1)

// generalized

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



1st deq() → Transfer $s_1 \to s_2$ + s_2.pop() → $2k+1$

2nd deq() → 1 oper

3rd deq() → 1 oper

. . . (k−1)

k^{th} deq → 1 oper

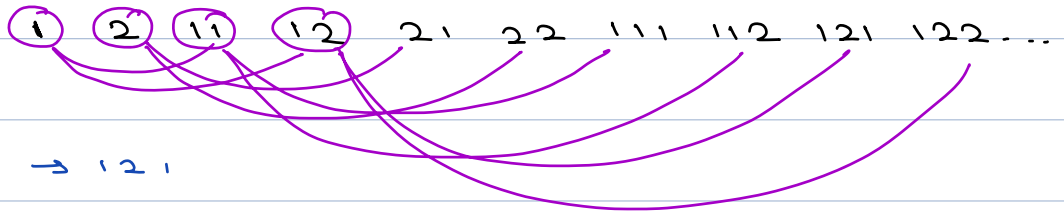
avg. deque cost = $\dfrac{2k+1 + \overbrace{1+1+1\cdots}^{k-1}}{k} \Rightarrow \dfrac{2k+1+k-1}{k}$

$\Rightarrow \dfrac{3k}{k} \Rightarrow 3$

Constant

find nth perfect number

↳ numbers using digit 1 or 2



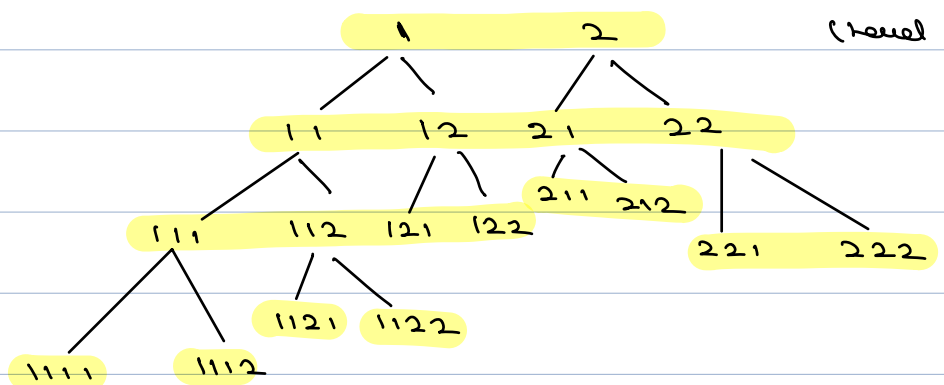
k = 9 → 121

k = 3 → 11

k = 6 → 22

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

BFS (Breadth first search / Level Order Traversal)



k = 5

~~1~~ ~~2~~ ~~11~~ ~~12~~ 21 22 111 112 121 122

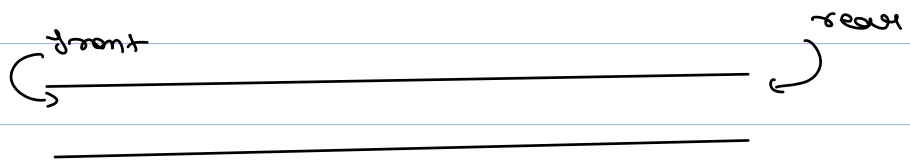
```
String kth number ( int k ){
    Queue <String > q ;
    q. add ( 1 );
    q. add (2);
    for (i=1; i<k; i++) {
        string ele = q.front();
        q. deque();
        q.enque (ele + "1");
        q.enque (ele + "2");
    }
    print (q.front());
}
```

optimise by not adding elements after size becomes k.

T.C → O(k)
S.C → O(k)

→ Doubly 2L.

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 → $(n-k+1) * k$, if $k = n/2$

T.C → $O(n^2)$.

↓

no. of subarrays of size k.

k = 4.

arr[] = 3 15 6 12 4 2 10 9 13 7 2 5 3

(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 3

Ans → 15 15 12 12 10 13 13 13 13 7

removing from end

removing from start

0 1 2 3 4 5 6 7 8

3 2 3 4 5 5 4 5 2 k = 4

~~3~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~5~~ ~~4~~ 5 2

4 5 5 5 5 5

0 1

5 5 x y

0

5 x y

```
Deque → q;
for (i → 0 to k-1) {
    while (!q.isEmpty() && A[q.rear] <= A[i]) {
        q.remove_rear();
    }
    q.insert_rear(i);
}
Print (q.front());
```

```
for (i → k to n-1) {                    // Miding window
    while (!q.isEmpty() && A[q.rear] <= A[i]) {
        q.remove_rear();
    }
    q.insert_rear(i);
    if (q.front() == i-k) {
        q.remove_front();
    }
    Print (q.front());
}
```

T.C → O(n)

S.C → O(k)