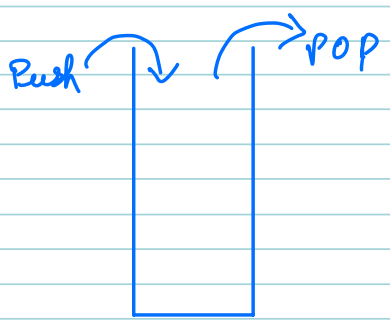
Queues : Implementation & Problems

TODAY'S CONTENT

- Queues
- Implementation of the queue using Arrays
- Implementation of the queue using Stack
- Perfect Number Question
- Double ended queue
- Sliding Window Maximum

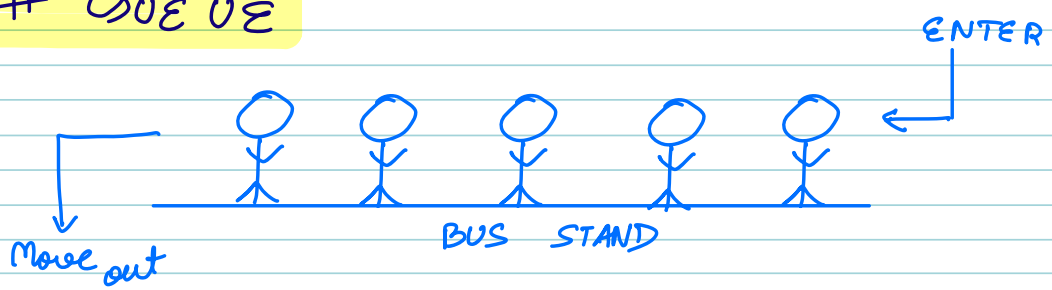
=> STACK



operation
- Push(x)
- Pop()
- Peek()
- isEmpty()

LIFO Principle

QUEUE



FIFO Principle

⇒ Function of Queue

① Enqueue(x) :- x will enter to the rear end

② Dequeue() :- 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 Using ARRAYS

8, 14, 9, 20, ↑, 30, front(), ↑, rear(), 60, ↑, 5, 10, 15, 16

~~8~~ ~~14~~ ~~9~~ 20 30 60 5 10 15

QUEUE

queue is full

$A =$

0	1	2	3	4	5
~~5~~ ~~8~~	10 ~~14~~	15 ~~9~~	20	30	60

r f

circle back

Front = ~~0~~ ~~1~~ ~~2~~ 3

Rear : ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ [0] ~~1~~ 2

PSEUDO CODE

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

Deque ( ) {
    if ( size == 0) {
        return queue is empty;
    }
    temp= arr [ front ];
    front = (front +1) % n;
    size --;
    return temp;
}
```

FIXED LENGTH

⇒ Problem with Arrays Implementation.

Dynamic Array

Use Linked List

⇒ Implementation via Linked List

operation of LL

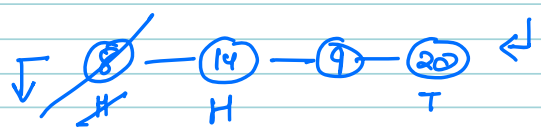
(Tail is maintained)

- addfirst() → O(1)
- addLast() → O(1)
- Removefirst() → O(1)
- RemoveLast() → O(N)

will use this Combination.

Ex

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



* We can simply use LL & use the above addLast(), removeFirst(). To simulate queue [FIFO]

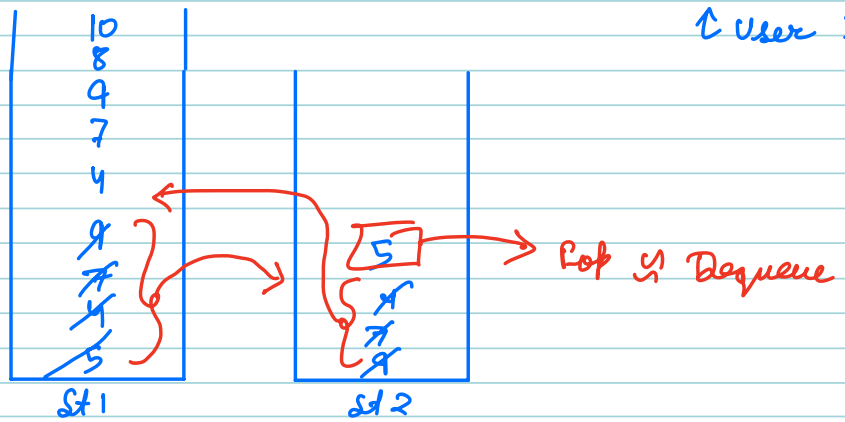
Question :- Implement a Queue (→ Enqueue, → Dequeue) using stack

Data

5 4 7 9 ↑ 8 10 ↑ ↑ 14 ↑ ↑ 21

5 4 7 1

↕ User Imagination



IDEA !

Enqueue → Push x in Stack 1 — O(1)

Dequeue → ↳ Transfer all elements from St 1 → St 2

↳ Remove top element of St 2

↳ Transfer all elements from St 2 → St 1 — O(N)

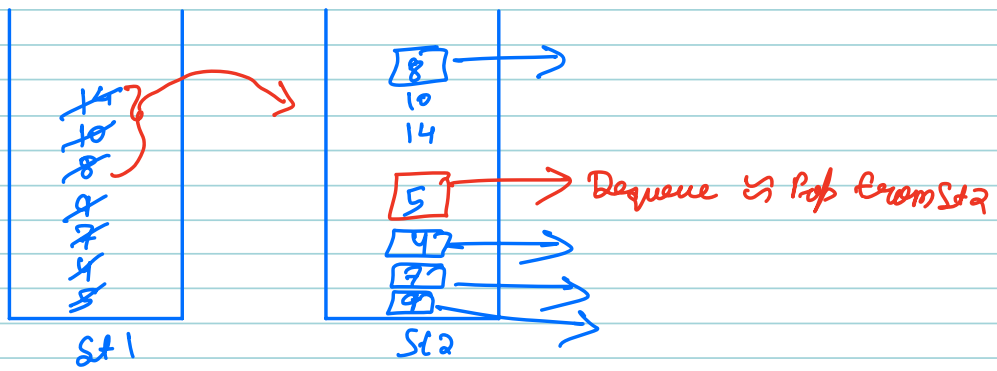
IDEA 2

5 4 7 9 ↑ 8 10 ↑ ↑ 14 ↑ ↑ 2 ↑ ↑ 13, 14

↑

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

Queue



→ Removing 5

1st deq() ⟶ Transfer all element St 1 → St 2 then pop
⇒ 9 operations

2nd deq() → 1 oper

3rd deq() → 1 oper

4th deq() → 1 oper

4 dequeues :- $\frac{\text{Total operations}}{4} = \frac{12}{4} = 3$ operation

* 1 Dequeue is making dequeue cheap for other operation.

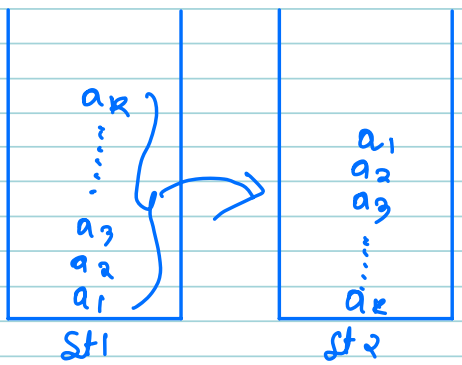
* where we have seen behaviour in past
↳ Arraylist.

Enqueue :- O(1)

Dequeue :- O(1) Amortized

// Generalization

$a_1, a_2, a_3, \ldots\ldots a_k$, deq()



→ transfer is required.

1st deq() :- 2K+1

2nd deq() :- 1 oper

3rd deq() :- 1 oper

⋮

K^{th} deq() :- 1 oper

Now, Average for 1 deq :- $\dfrac{\overbrace{(2K+1) + 1 + 1 + 1 \ldots\ldots + 1}^{K}}{K}$

$= \dfrac{2K + K}{K}$

$= \dfrac{3K}{K} \approx 3$ operations

Hence we can say No matter how many Enqueue & Dequeue is performed the average cost per Dequeue will remain 3.

10:26 ← Resume time.

Question :- Find n^{th} Perfect Number

↳ Numbers formed using Digit 1 or 2

Ex :-

0	1	2	3	4	5	6	7	8	9
1	2	11	12	21	22	111	112	121	122 .....

if, $K = 9 \longrightarrow 122$

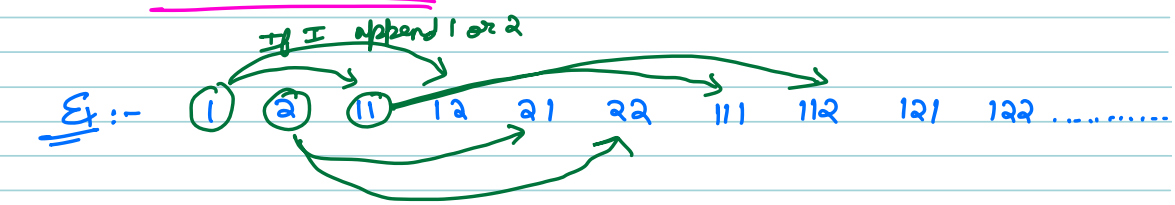
$K = 3 \longrightarrow 12$

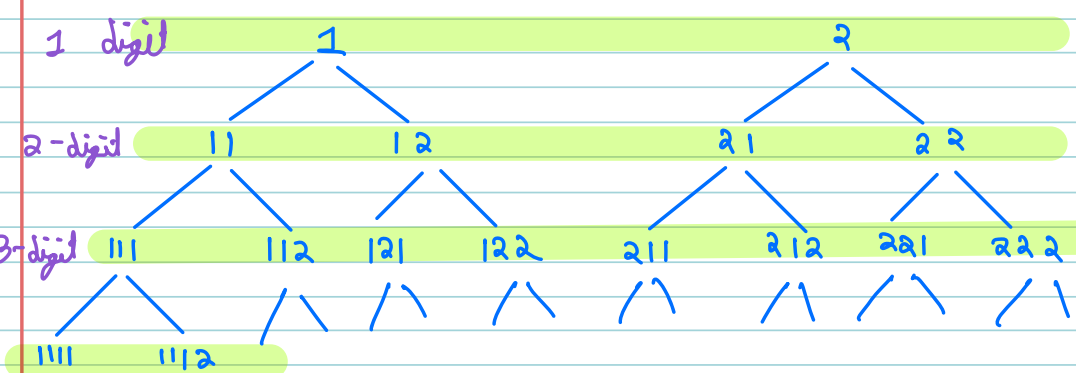
$K = 6 \longrightarrow 111$

Brute force

↳ For all $\forall$ Natural numbers, check if it has digit 1 or 2 ⟶ find K^{th} No.

↳ **OPTIMIZATION**





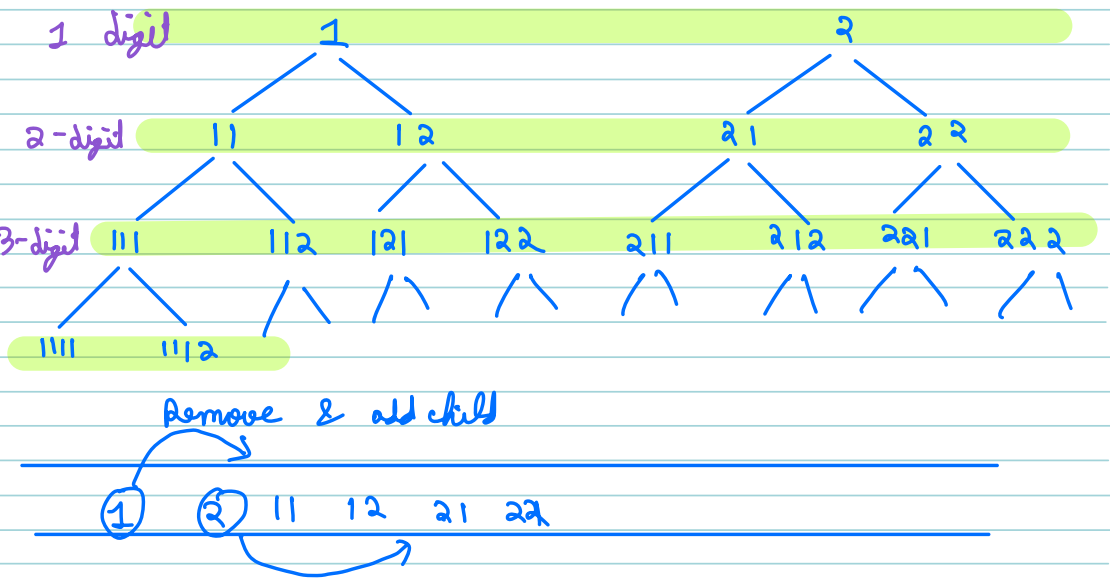
* This is known as BFS (Breadth first search)

Brief Idea

DFS (Depth first search) → In Recursion, we use DFS

BFS, we go over levels one by one.

DRY RUN [BFS]



* we use Queues to do BFS.

* BFS is also known as Level order traversal

PSEUDO CODE

```
String   Kth Number ( int K ){
         queue < String > q ;
         q. add ("1");
         q. add ("2");

         for (i = 1;  i < K ; i++){
              String num = q. front();
              q. remove();
              q. add ( num + " " + "1");
              q. add ( num + " " + "2");
         }
         return q. front();
}
```

STUPIDITY

0	1	2	3	4	5	6	7	8	9
1	2	11	12	21	22	111	112	121	122 .....

Ex :-

K = 5

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

Do I need this

↳ No

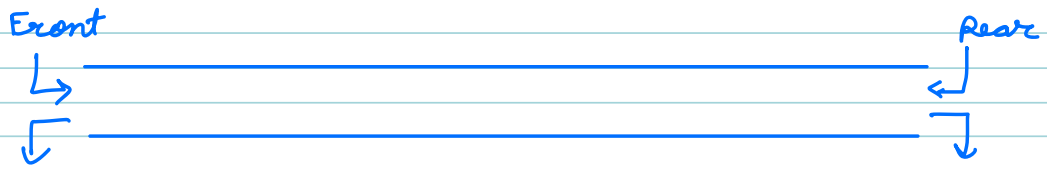
PSEUDO CODE

```
String   Kth Number ( int K ){
    queue < String > q ;
    q . add ("1");
    q . add ("2");
    insertions = 2;
    for (i = 1;  i < K ; i++) {
        String num = q . front ();
        q . remove ();
        if ( insertions < K) {
            q . add ( num + " " + "1");
            q . add ( num + " " + "2");
            insertions += 2
        }
    }
    return q . front ();
}
```

TC → O(K)

SC → O(K)

Double Ended Queue [Deque]



insert_rear()	insert_front()
remove_rear()	remove_front()
rear()	front()

* Implementation :- Using DLL.

Question :- Sliding Window Maximum

Given arr[N] & K, Print max element in every window of size K > 1

Ex arr[9] = [10, 1, 9, 3, 7, 6, 5, 11, 8] → SLIDE

K = 4

Ans = [10, 9, 9, 7, 11, 11]

Brute force :- # window traverse & check max.

TC → O[(N-K-1) * K] → For Find Max

(N-K-1) → No. of Windows.

Worst Case, $K = \frac{N}{2}$

$TC \rightarrow O(N^2)$

⇒ OPTIMIZATION

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr[] =	3	15	6	12	4	2	10	9	13	7	2	5	3

IDEA

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr[] =	3	15	6	12	4	2	10	9	13	7	2	5	3

Slide

Queue

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

↳ when 15 comes 3 can't be ans

Ans = [15, 15, 12, 12, 10, 13, 13, 13, 13, 7]

⇒ Points of Observation

① when window move ahead from element on front we need to remove it.

② while Inserting remove element from end which can never be Ans.

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

K = 4

3, 2, 3, 4, 5, 5, 4, 5, 2

Ans = [4, 5, 5, 5, 5, 5]

★ Instead of elements store indexes in Deque.

PSEUDO CODE

```
Deque q;
for ( i → 0 to k-1){                          // First Window
    while( !q.isEmpty() && A[q.rear()] <= A[i]){
        q.removeRear();
    }
    q.addLast( i );
}
Print (A[q.front()]);
for (i → k to N-1){
    if ( q.front() == i - k) {
        q.removeFront();
    }
```

```
while( !q.isEmpty() && A[q.rear()] <= A[i]){
        q.removeRear();
}
q.addLast(i);
Print(A[q.Front()]);
}
```

TC → O(N)

SC → O(N)