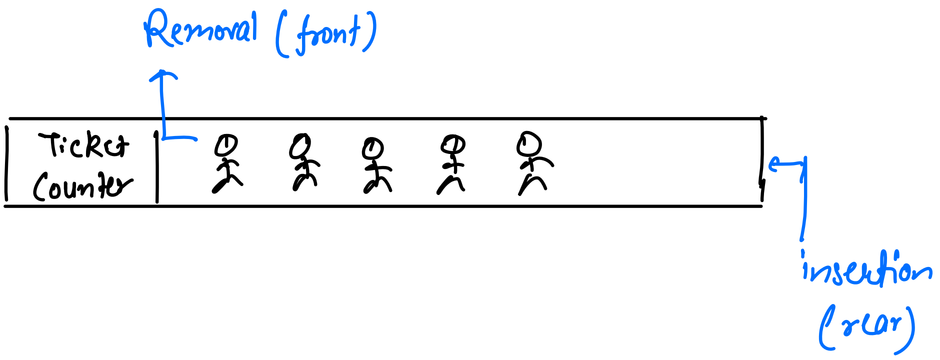
Queue



FIFO → First In First Out.

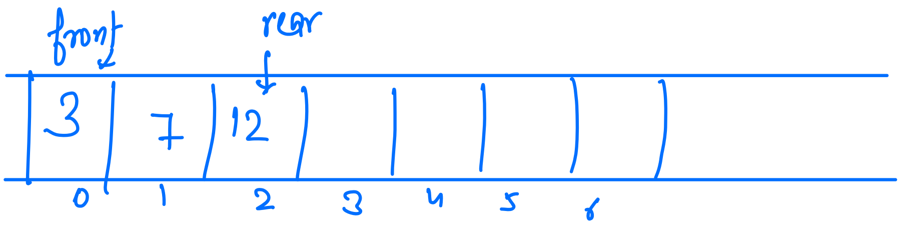
- Call centre
- Printer
- Message Queue

Operations by Queue →

O(1)

- Enqueue(x) → insert data at rear end
- Dequeue() → Remove data from front
- front() / peek() → return the data present at front
- isEmpty() → check if queue is empty or not.

Eq(3), Eq(7), Eq(12), dq(), dq(), eq(8), eq(3), front()

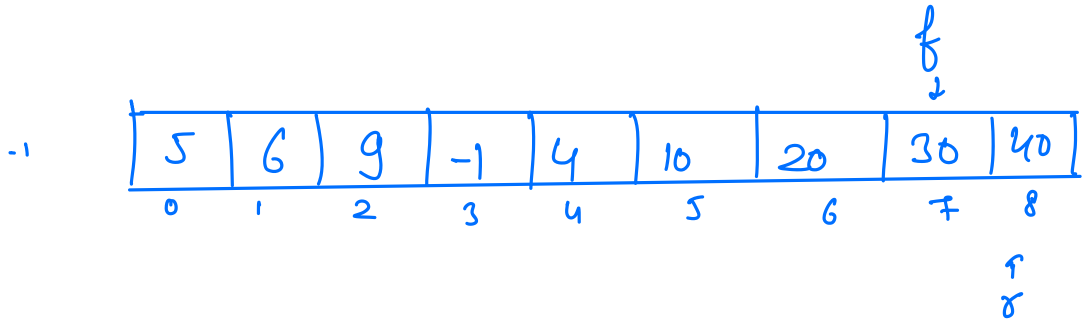


Array Implementation of Queues

front = -1 → idx of last element which was just removed

rear = -1 → idx of last element which was just added.

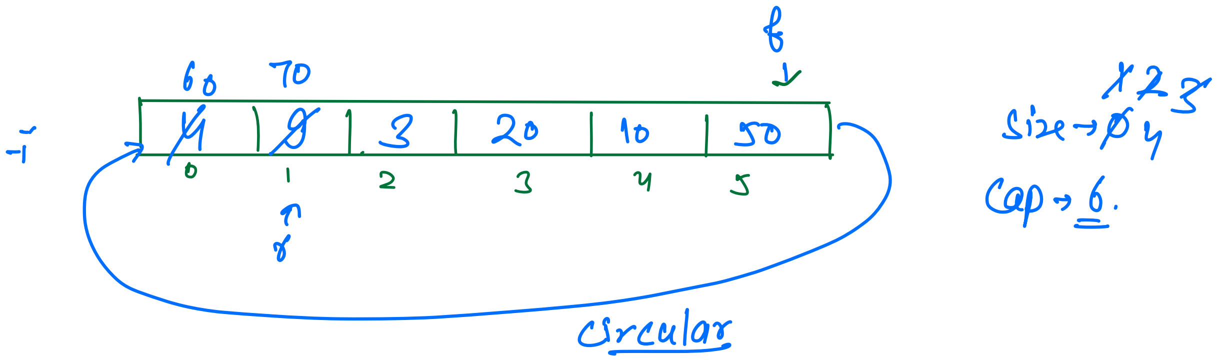
Eq(5)✓ Eq(6)✓ Eq(9)✓ Eq(-1)✓ Dq()✓ Dq()✓ front()✓ Eq(4)

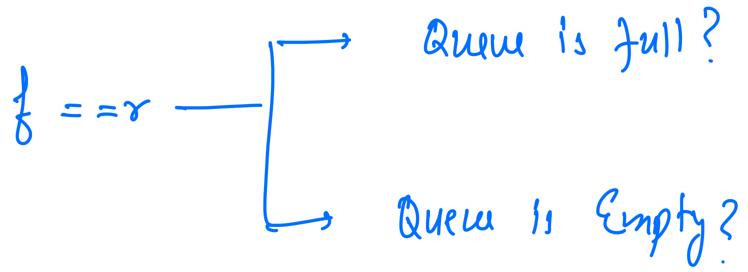


```
void enqueue( int x){
    if(r==N-1) {return }
    r++;
    arr(r]=x;
}
```

```
int dequeue( ){
    if( f ==r) { //Queue is empty
                  return -1}
    f++;
    return arr(f];
}
```

```
int front( ){
    if( f ==r) { //Queue is empty, return -1}
    return arr[f+1];
}
```





f == r

Queue is full?

Queue is Empty?

```
void  enqueue( int x){
      if ( size = = cap) { //Queue is full}
       r = (r+1) % N
       arr(r] = x ;
        size ++;
}

int  deque ( ){
      if ( size ==0) { // Queue is Empty ,return -1 }
       f = (f+1) % N;
        size --;
      return arr[f];
}

int  front( ) {
        if ( size ==0) { // Queue is Empty ,return -1 }
        return arr[(f+1) % N];
}
```

→ O(1)

Implementation of Queue using linked-list →

enqueue(x) → insertion → head / tail

dequeue() → deletion → head / tail

✗ [insert at Head
~~delete from tail~~]

↓

O(N)

✓ [insert at tail
delete from head]

code → todo.

Implementation of Queue using Stack → [Directi]

O(1) { → dequeue()
→ enqueue()
→ front() }

→ pop
→ top
→ push

~~10~~	~~20~~	~~3~~	~~9~~	~~7~~	30	40	50

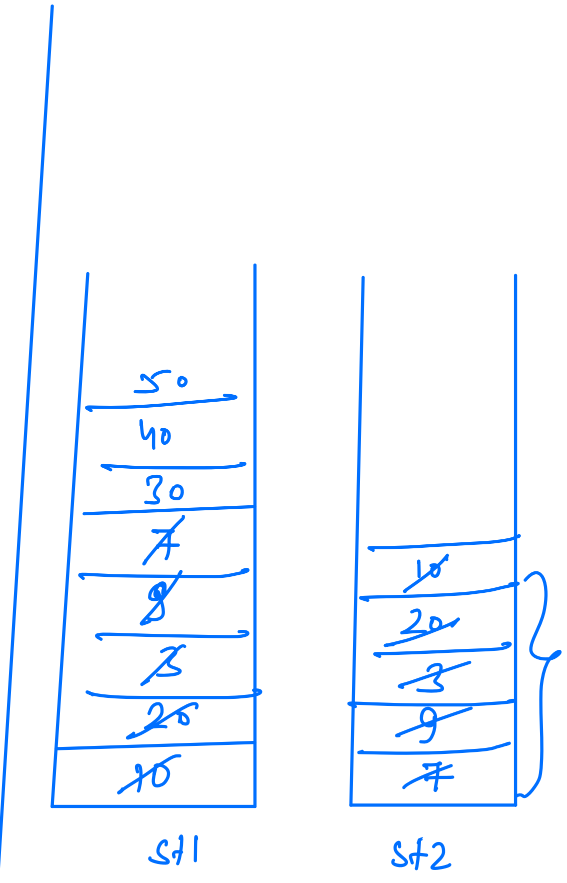
1 dq() → N operations

dq() → 1

dq() → 1

dq() → 1

dq() → 1



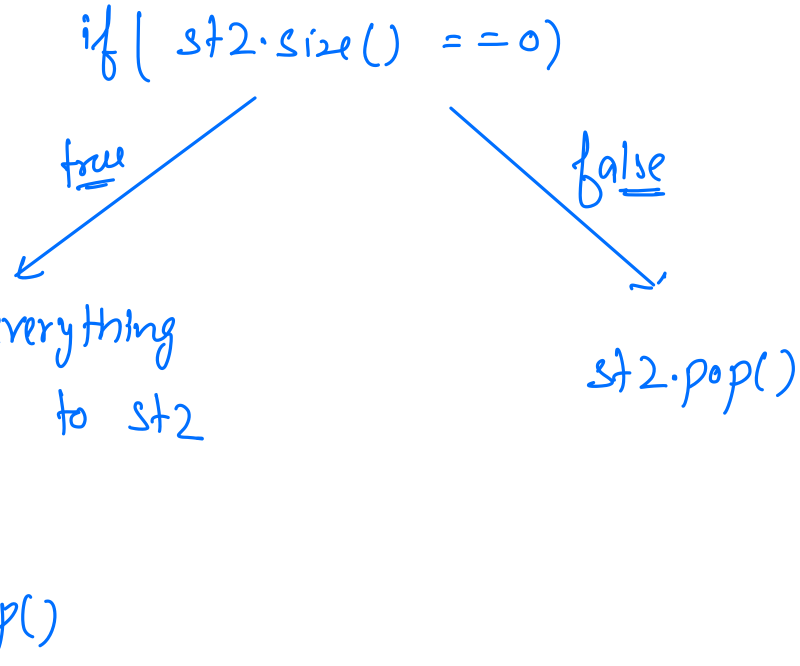
N dequeues → 2N operations

1 dequeue → $\frac{2\cancel{N}}{\cancel{N}}$ operations

↳ $\underline{\underline{O(1)}}$

Enqueue() → st1.push(x)

Dequeue() →



if(st2.size() ==0)

true

transfer everything from st1 to st2

↓

st2.pop()

false

st2.pop()

Perfect Numbers

Find A^{th} perfect number. A perfect number has some properties →

(1) it comprises of 1 or 2 or both.

(2) no. of digits in perfect no. must be even.

(3) no. must be palindrome.

9th

11, 22, 1111, 1221, 2112, 2222, 111111, 112211, 121121

[n → a a'] → first half + reverse of first half.

1, 2, 11, 12, 21, 22, 111, 112, 121, 122, 211, 212, 221, 222, - -

~~1~~	2	11	12

count = ~~1~~ ~~2~~ ~~3~~ 4

first half of 9th perfect no. →

A = 9.

9th perfect no = 121 + rev(121)

= 121121

code :-

$1 \leq A \leq 10^5$

```
if (A==1) { return "11" }
if (A==2) { return "22" }
Queue<string> q;
 q.enqueue("1") , q.enqueue("2");
 count = 2;     String ans = "";
 while(  true    ){
        num = q.dequeue();
        first  =  num + "1" ;
        second =  num + "2" ;
        q.enqueue(first);
         count++;
         if (count == A) { ans = first, break }

        q.enqueue(second);
         count++;
         if (count == A) { ans = second, break }
 }

   return   ans + rev(ans);
```

A = 5

~~1~~	2	11	12

count = ~~2~~ ~~3~~ 4

T.C → O(A)
S.C → O(A)

Sliding Window Max

→ find max of every subarray of size K.

[3,	2,	3,	4,	5,	5,	4,	5,	6]	, K=4
0	1	2	3	4	5	6	7	8	

[0-3] →	4
[1-4] →	5
[2-5] →	5
[3-6] →	5
[4-7] →	5
[5-8] →	6

O/p. → [4, 5, 5, 5, 5, 6]

idea.1 → Consider all the subarrays of size K, iterate on that subarray & find maximum element.

T.C → $O(N^2)$

Nearest Greater on r.h.s

				✓	✓	✓	✓	✓	✓	
3	2	9	4	-1	16	1	7	-2	5	-5
0	1	2	3	4	5	6	7	8	9	10

~~3~~	~~2~~	~~9~~	~~4~~	~~-1~~	~~16~~	~~1~~	7	~~-2~~	5	-5

(store indices)

[9, 9, 16, 16, 16, 16, 7, 7]

→ remove all smaller elements

→ insert curr elemt

{* front elemt → max of current window}

→ use linkedlist or doubly ended queue

→ {Revise Recursion/ Basic Trees}