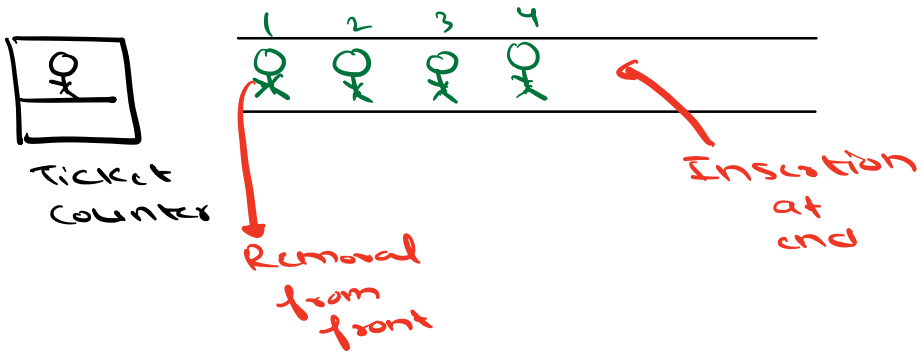
Agenda

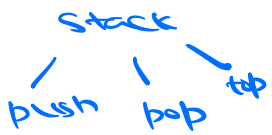
Queue
Implementation of queue using array
Implementation of queue using stack
Ques: Perfect No.
Doubly ended Queue
Sliding Window Maximum

Queue

A linear data structure where items are added at the end and removed from other end. Queue follows First In First Out (FIFO) principle.



Functions of Queue

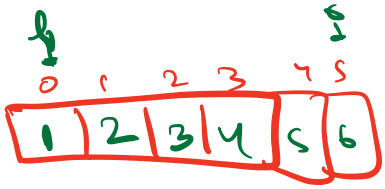


1) Enqueue (x)
   Add element x at rear (or last) of queue

2) Dequeue ()
   An element is removed from front of queue and it is returned.

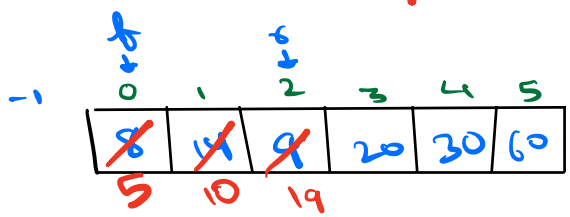
3) Front() → Element in front of queue
4) Rear() → Element at rear of queue
5) isEmpty() → Queue is empty
6) size()

Implementation of Queue



1) using arrays

8 14 9 20 30 60 5 10 19



sz = 6

f=0, r=-1

E(8)	r++	a[r]=8	r=0
E(14)	r++	a[r]=14	r=1
E(9)	r++	a[r]=9	r=2
E(20)	r++	a[r]=20	r=3
D()	f++		f=1 r=3
E(30)	r++	a[r]=30	r=4
Front()	14	a[f]	
D()	f++		f=2 r=4
Rear()	30	a[r]	
E(60)	r++		r=5
D()	f++		f=3 r=5
E(5)	r++ → r%N	a[r]=5	r=6 → 6%6=0
E(10)	r++	a[r]=10	r=1
E(19)	r++	a[r]=19	r=2

If (r+1)%N == f
Queue is full

N=6
idx → 0 to 5
no. %6

D( ) f++ f = 4

D( ) f++ f = 5

D( ) f = (f+1) % n f → 6 % 6 → 0

r → 2

n → arr.len

f = 0 r = -1 size = 0

enqueue (x) {

if ((r+1) % n == f) {

return Queue is full

}

r = (r+1) % n

a[r] = x size++

}

r = -1 ||

dequeue() {

if (size == 0)

return Queue is empty

temp = a[f]

size--

f = (f+1) % n

return temp

}

Problem: Array size is fixed

↙ Dynamic Array / Arraylist

↘ Linked List

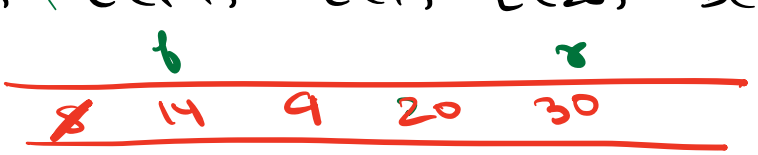
Everytime array is full, a new array gets created of double the size, all elements get copied to new arr

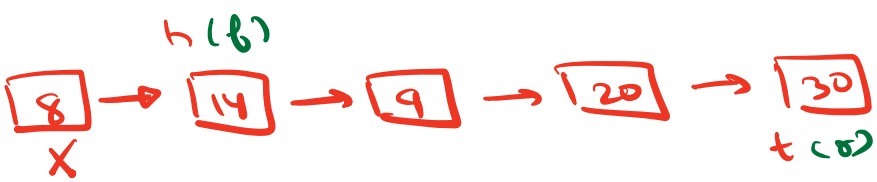
2) Using Linked List

LL (head and tail)	head ↓ Front	tail ↓ rear	head ↓ rear	tail ↓ front
	deletion from front ↓ O(1)	insert at rear ↓ O(1)	insert at rear ↓ O(1)	delete from front (tail) ↓ O(N)

head → front
tail → rear

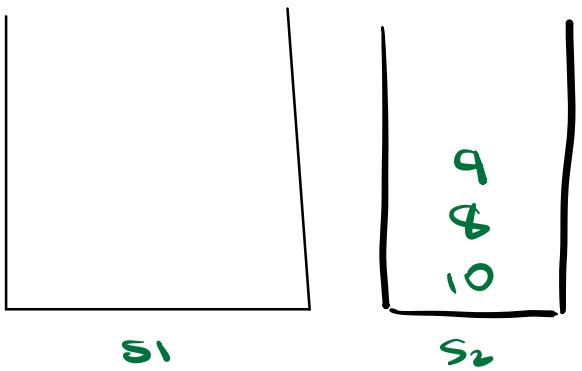
E(8) E(14) E(9) E(20) D() E(30) F()





3) Using stack

E(5), E(4), E(7), E(9), D(), E(8), E(10), D(), D()
E(14), D(), D(), E(21)



Enqueue(x) → Push x into stack (S1)

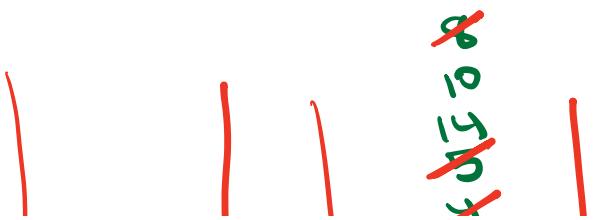
Dequeue() → Transfer all elements from $S_1 \rightarrow S_2$

Remove top el from S_2

Transfer elements back from $S_2 \rightarrow S_1$

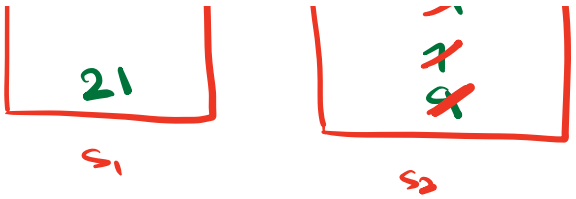
E(5), E(4), E(7), E(9), D(), E(8), E(10), D(), D()
E(14), D(), D(), E(21)

Enqueue(x) → Push x into stack (S1)



Dequeue()

if (s2.size == 0) {

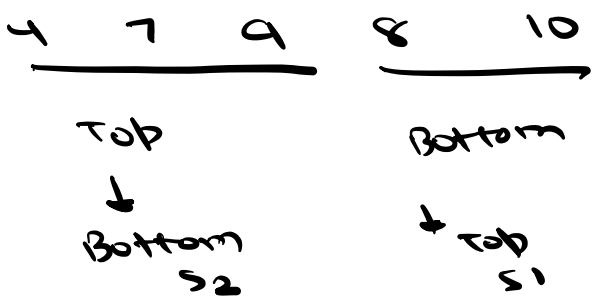


Transfer S1 → S2

S2.popC)

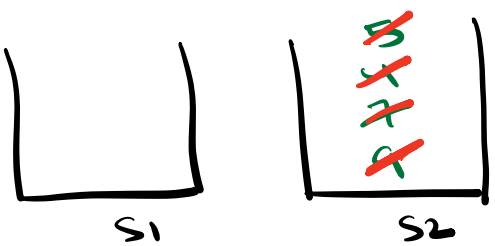
Best case → O(1)

Worst case → O(n)



~~5~~ ~~4~~ ~~7~~ ~~9~~

5 4 7 9 D() D() D() D()



1st dequeue (del 5)

↓

Transfer 4 elements from S1 → S2

+ S2. pop()

9 operations

2nd dequeue

↓

1 operation

3rd → 1 operation

4th → 1 operation

4 del → 12 oper

1 del → $\frac{12}{4}$ = 3 operation

1 del on avg takes O(1)

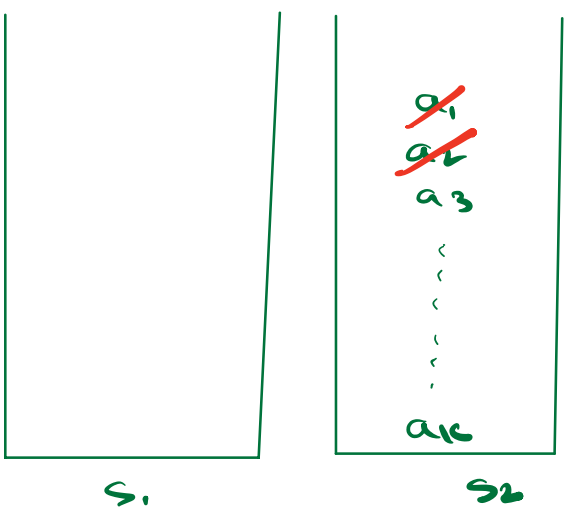
Enqueue → O(1)

Dequeue → O(1) Amortized TC

// generalized

~~a_1~~ a_2 ... a_K

E(a_1) E(a_2) .... E(a_K) D()



1st deletion →

Transfer $S_1 \rightarrow S_2$ +

S_2.pop()

2K+1 operations

2nd deletion → 1 operation

3rd → 1 operation

Kth deletion → 1 operation

K elem → $2K+X+K-X$ → $3K$ operations

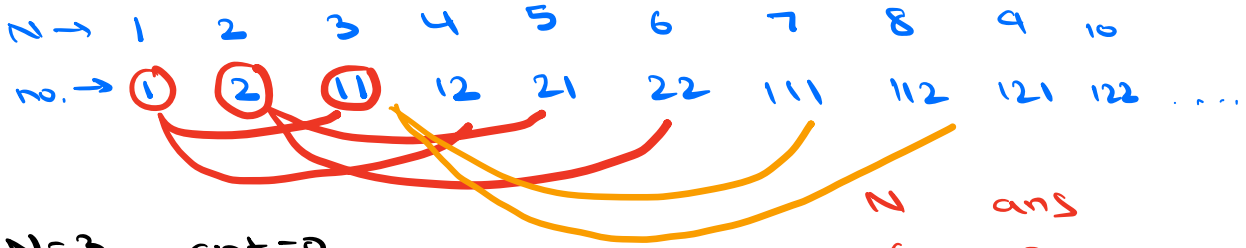
1 elem → $\frac{3K}{3} = 3$ operations

↓

constant

10:38

1. An integer N is input, return N^{th} perfect no. A perfect no. is formed by only digits 1 and 2.

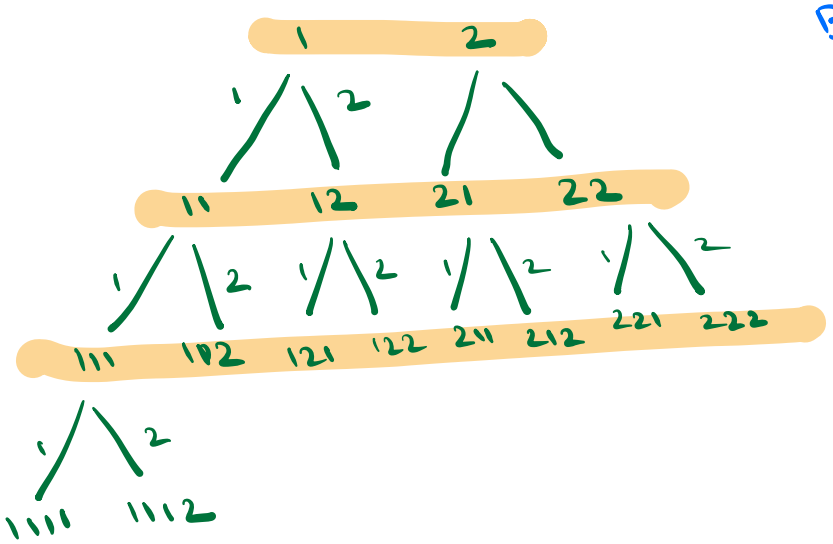


N	ans
6	22
3	11
9	121

N=3 cnt=0

Num			
1	✓	cnt++	cnt=1
2	✓	cnt++	cnt=2
3	X		↓2
4	X		2
5	X		
6			
⋮			
11	✓	cnt++	3

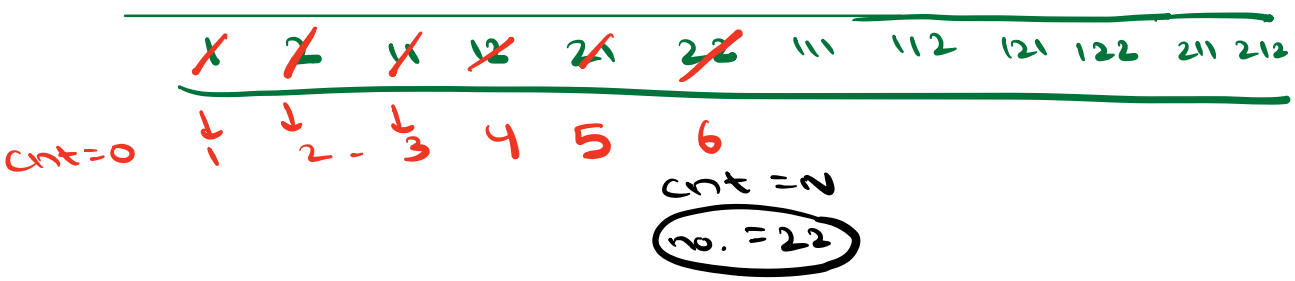
BF :

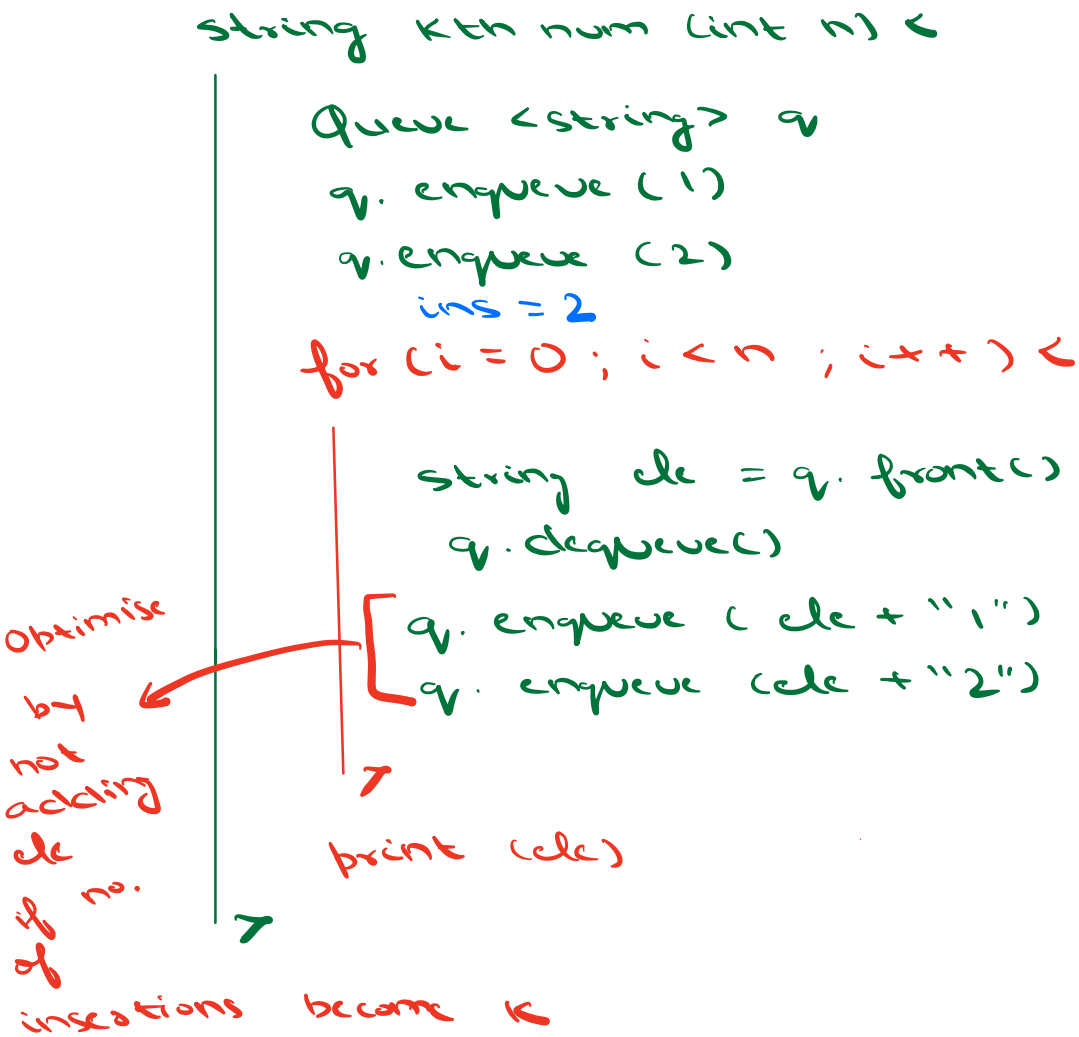


BFS
(Breadth First search)

↓
Queue

N = 6




string kth num (int n) {
Queue <string> q
q. enqueue (1)
q. enqueue (2)
ins = 2
for (i = 0 ; i < n ; i++) {
string ele = q. front()
q. dequeue()
q. enqueue (ele + "1")
q. enqueue (ele + "2")
print (ele)
Optimise by not adding ele if no. of insertions became

TC : O(N)

SC : O(N)

Doubly Ended Queue (Deque)

Data structure that allows elements to be added or removed from both ends

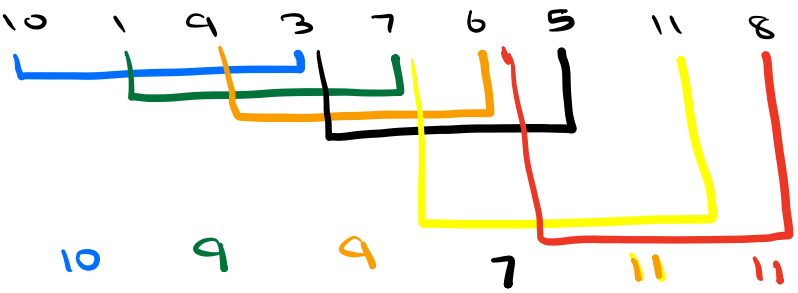
push_front()	push_rear()
pop_front()	pop_rear()
front()	rear()

Deque → Doubly LL

2. Given an integer array A and window size k, find max element in every window of size k.

k = 4

a[9] = 10 1 9 3 7 6 5 11 8



ans → 10 9 9 7 11 11

How many windows ? 6

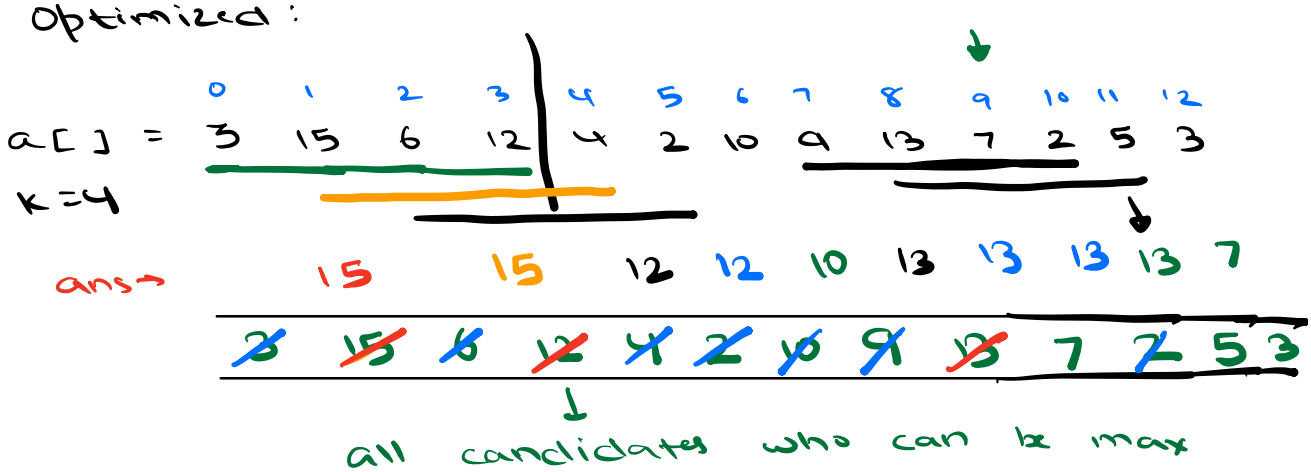
How many subarrays of size K are in an N size array ? N-K+1

BF : For every subarray of size k, traverse and find max.

$(N-K+1) * K$ $\quad K = N/2$

$N/2 * N/2 \rightarrow TC : O(N^2)$

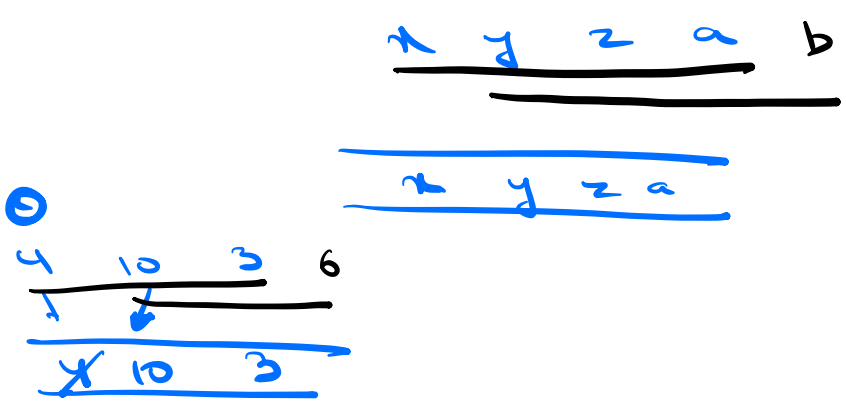
Optimized :

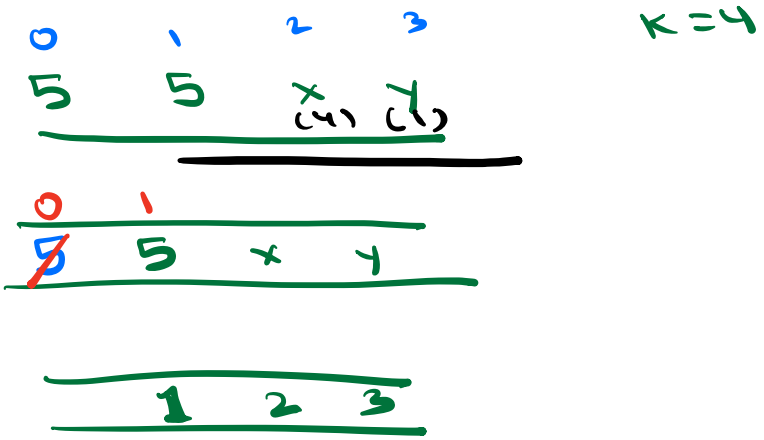


① Data is maintained in descending order

② Max de → front()

③ Outgoing de → del from front





Deque → q

```
for ( i = 0 ; i < k ; i + + ) {
    while (!q.empty() && A[q.rear()] <= A[i])
        q.pop-rear()
    q.insert-rear(i)
}
// Max of 0 → k-1
print (q.front())
```

0 → k-1
1 → k

```
for (i = k; i < n; i++) {
    while (!q.empty() && A[q.rear()] <= A[i])
        q.pop-rear()
    q.insert-rear(i)
    if (q.front() == i-k)
        q.remove-front()
    print(q.front)
}
```

TC: O(N)

SC: O(K)