Stack & queues basics

Stack

Last in first out

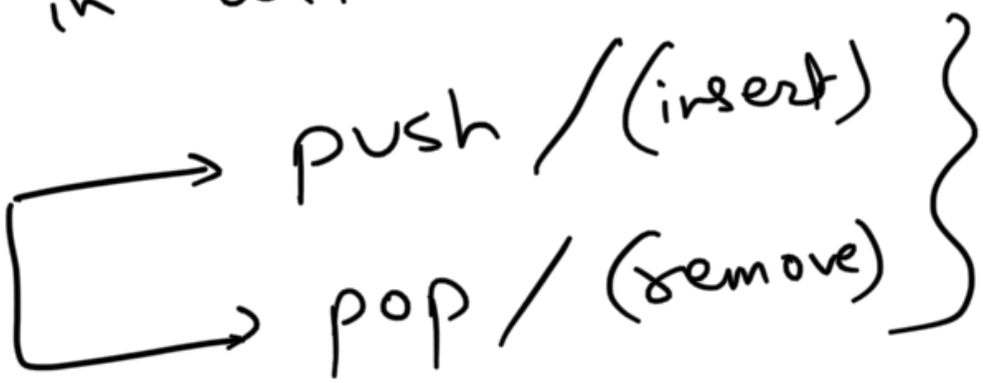
Stack → Abstract data type

↓

Logical definition is known

(Implementat^{n} can be done in different ways)

stack



top() / peek

isEmpty()

size() etc.

Applications

1. Recursion / call stack
2. undo - redo.
3. back button in browser.

Ctrl + T

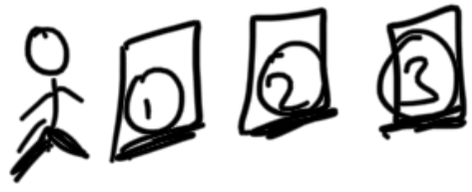
Calculator (1, W)

3 + 2 * 5 = 7

sum(N)

N + sum(N-1)

(N-1) + sum(N-2)



First in first out.

FIFO

Abstract

enqueue (insert/push/add)

dequeue (pop/delete)

data type

size
isEmpty()
front()

Applications?

1. Kafka / Rabbit MQ / Active MQ (HLD classes)

me

2. Scheduling
   - process
   - download

enque(1)
enque(5)
enque(3)
dequeue() → 1
dequeue() → 5

Implementations

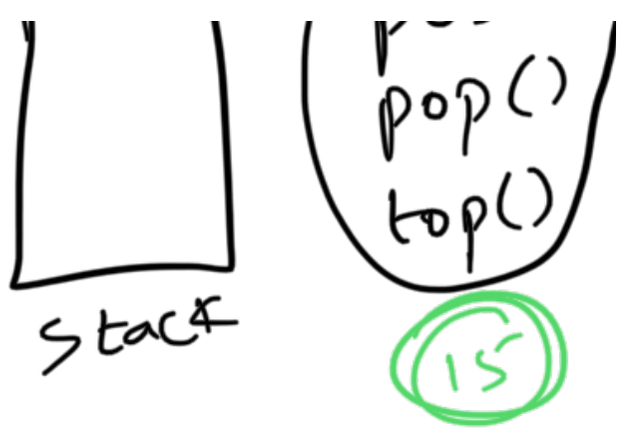
push(a)

1. Stack

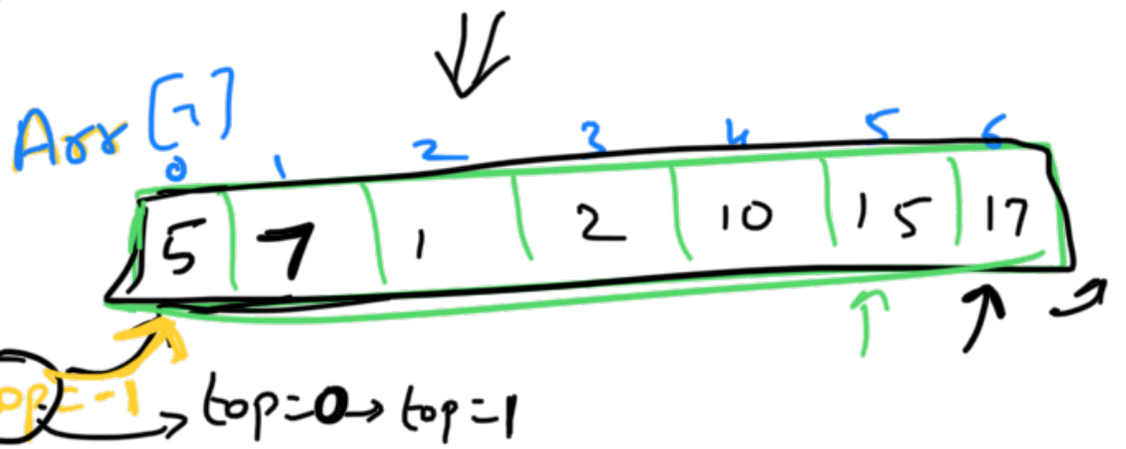
- array
- (H.W.)
- Linked list ✓✓

(Walmart, Vmware, SAP Labs, MMT etc.)

Array → (size is fixed).



Fix the maximum size




```
{
    int size;
    int arr[size];
    int top = -1;
```

// Always points to the topmost element

```
void push(int a)
{ if(top != size-1)
  {
    top++;
    arr[top] = a
  }
```

Bug

st.push(5)

st.push(7)

push(1)
(2)
(10)

} else throw exc()

}

```
void pop()
{
    if (top >= 0)
        top--;
}
```

```
int top()
{
    if (top != -1)
        return arr[top];
    else return -1;
}
```

15
(17)
st.push(3)

Arr[5]

Buy

push(12)

top++
arr[top] = 12

Buy?

T.C. => O(1)
S.C => O(N)

static

C++ → STL

Python / JS / ruby
a = []

Java
⇒ Collection :

Stack<INT> ST = new Stack< >()
st.push(i)
st.pop()

Stack<int> st
st.push(i)
st.pop()

a.append()
a.pop()

push
pop
top / peck
isEmpty
size()

push(a,b)?

push(5)
(6)
(10)
(15)

int top()

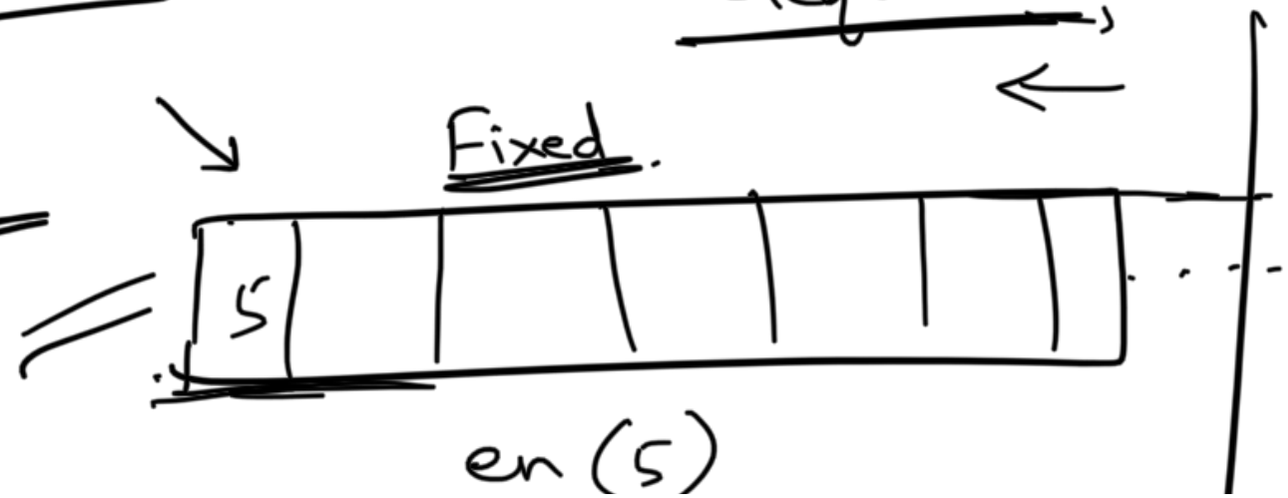
Queue (using array)

Think

copy & shift

enque()
dequeue()

Fixed.



en(10)

en(16)

deq() → 5

T.C => enqueue() => O(N)

| 10 | 5 | . . . . .

| 16 | 10 | 5 | . . . . . .

en(20)

| 20 | 16 | 10 | 5 |

dequeue() => top

O(1)

en(30)

| 30 | 20 | 16 | 10 | |

en(17)

| n | 30 | 20 | 16 | 10 |

back

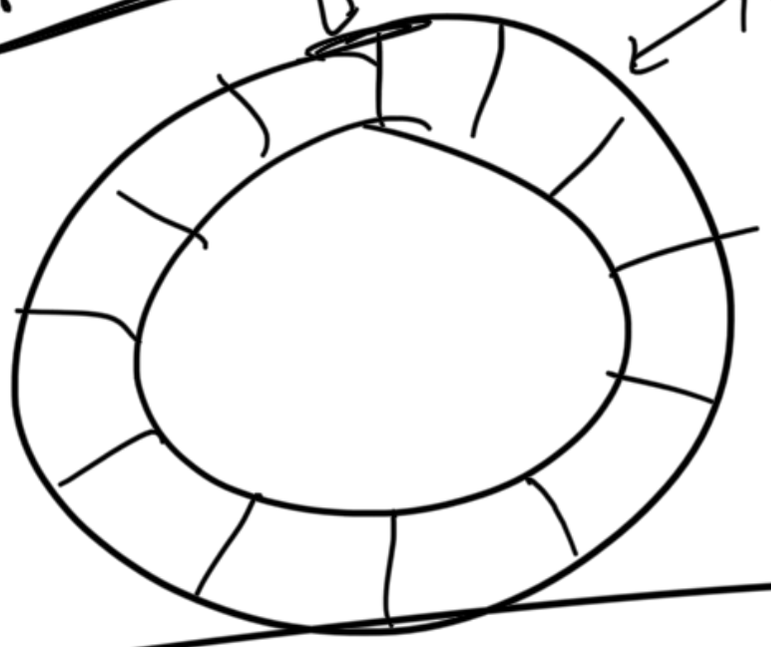
front

back

| 7 | 10 | 15 | 13 | 1 | 2 | 18 |

→ n-1 → n%n

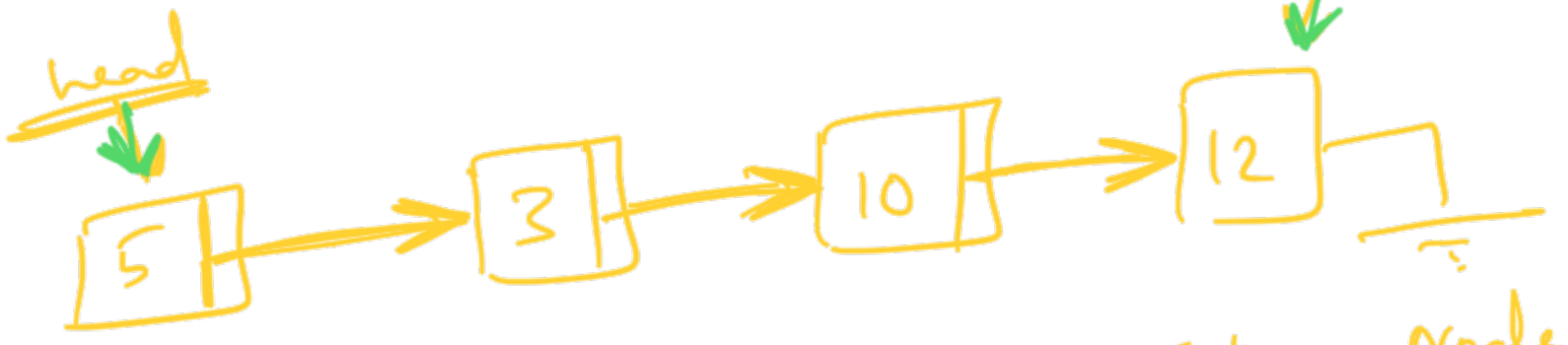
Circular



% mod

enq(5) → O(1)
en (7)
en (10)
en (15)
deq() → O(2)

Stack (using LL)



Class Node

Node

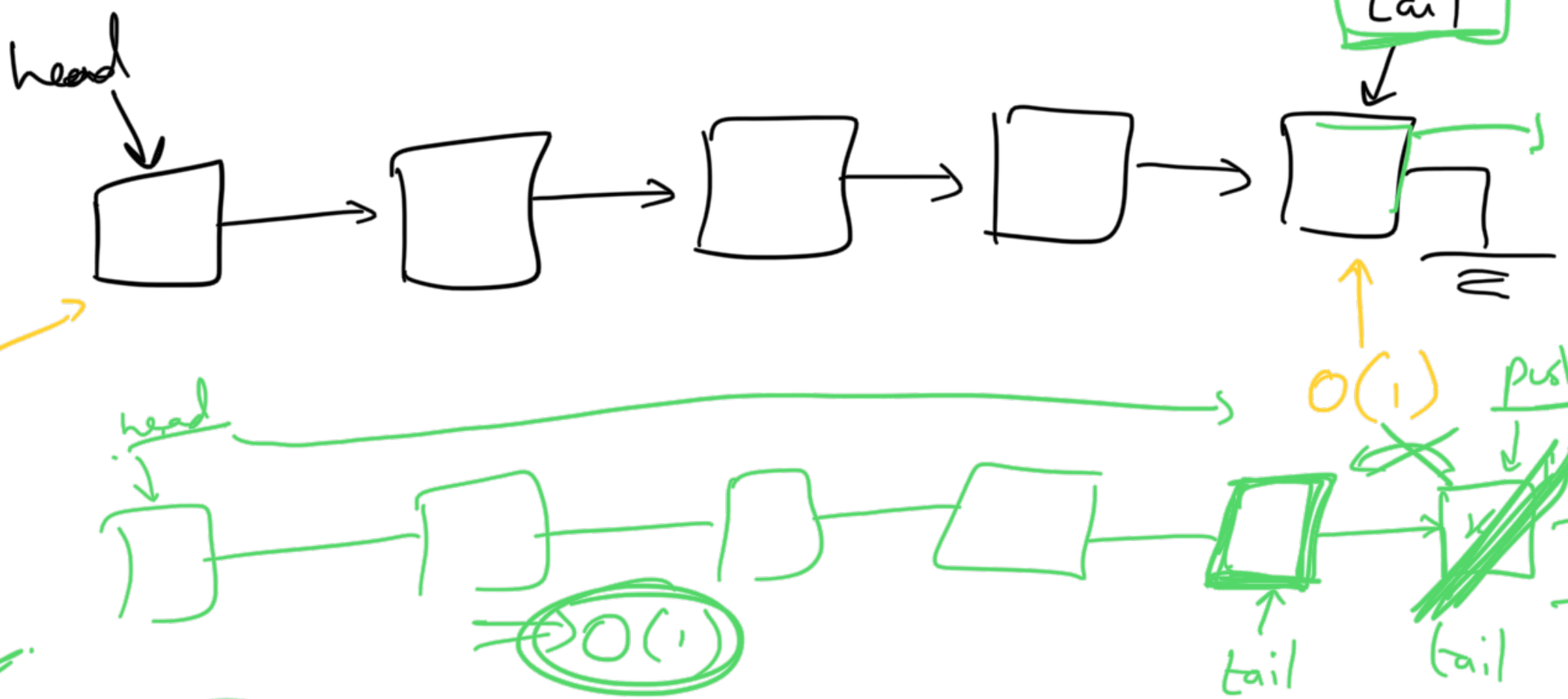
push (5)
push (8)
push (1)
pop()

pop()

push(2)

pop()

push(10)



push(x).

Insert at tail.

O(N)

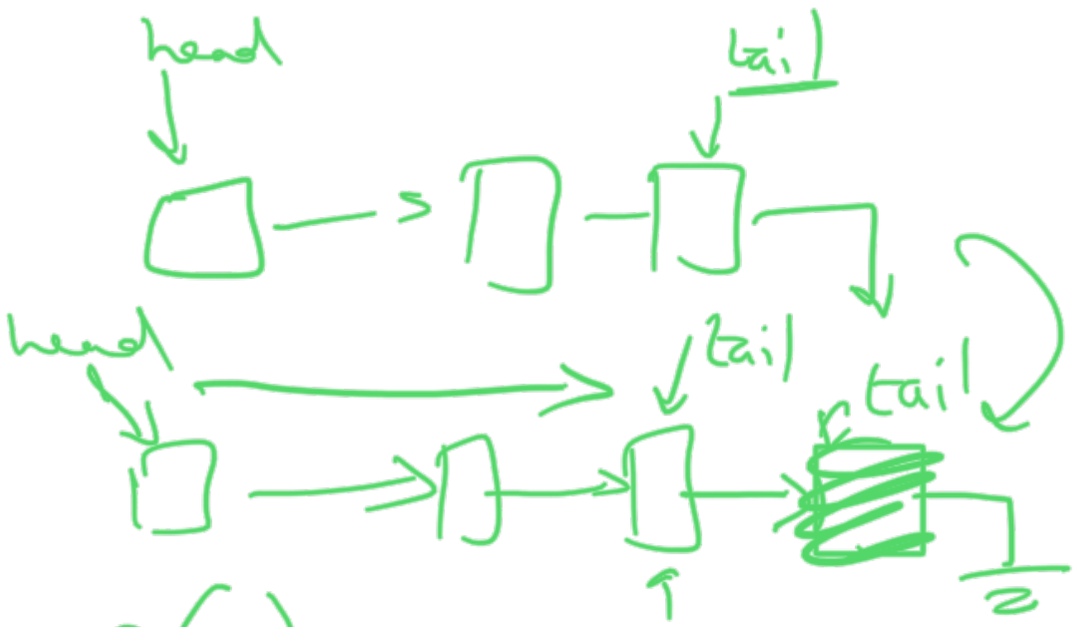
pop() removing at tail

head

head
insert at head
push()
O(1)


head = hea
head
deletion at head
pop()
O(1)


LL
(head, tail)
Class Node
{
int data
Node next;
O(1) Insert at head ( )
O(1) delete at head ( )
O(1) Insert at tail ( )
delete at tail ( )



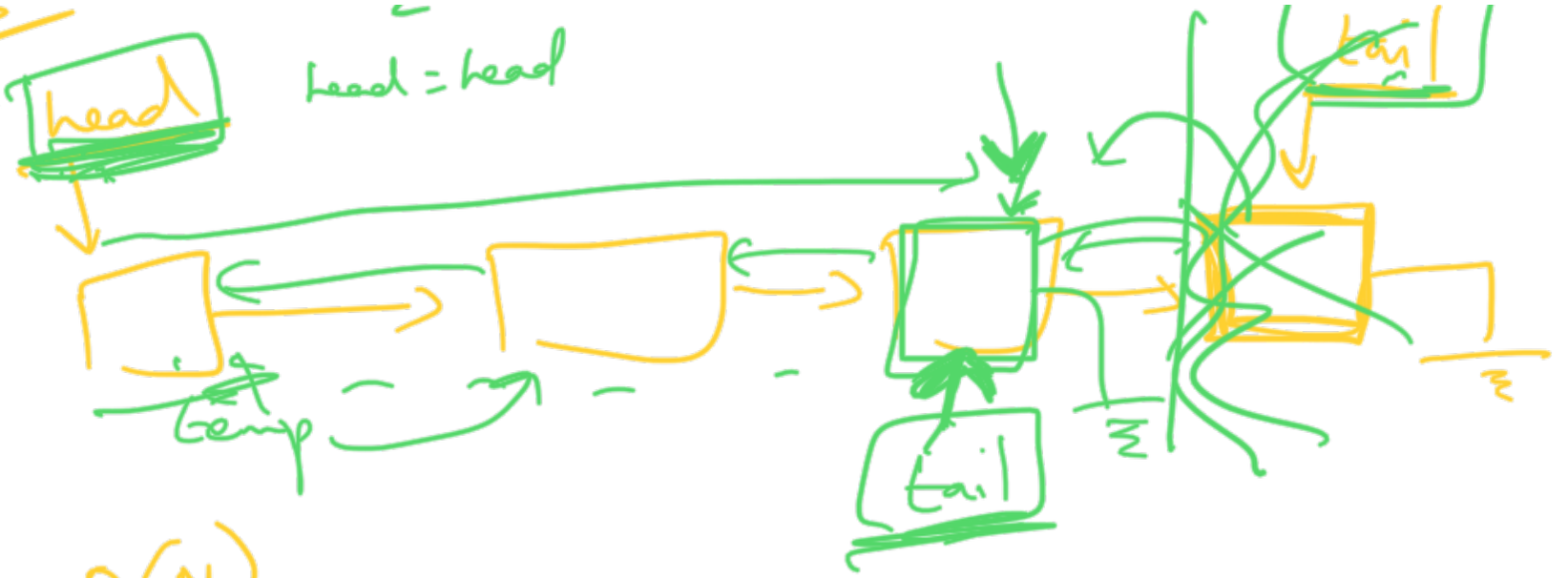
Insert at head → O(1)

delete at head → O(1)

Break → 10:30

2 variables

Doubt>



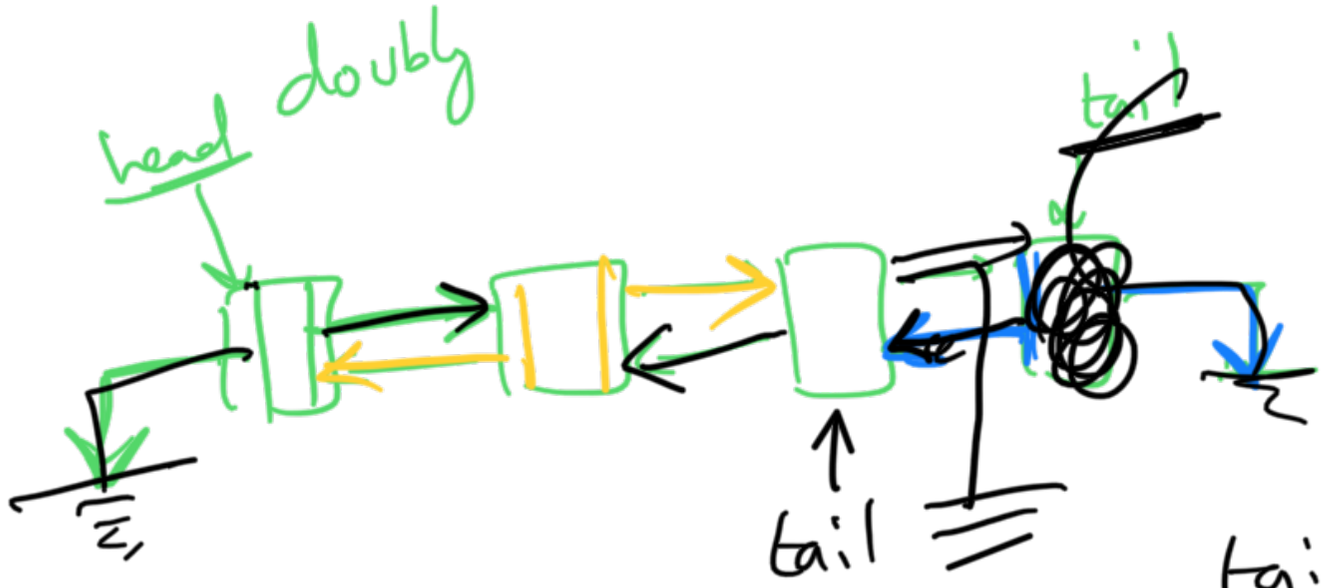
tail.prev

removing for tail → O(N)

while (temp != ten

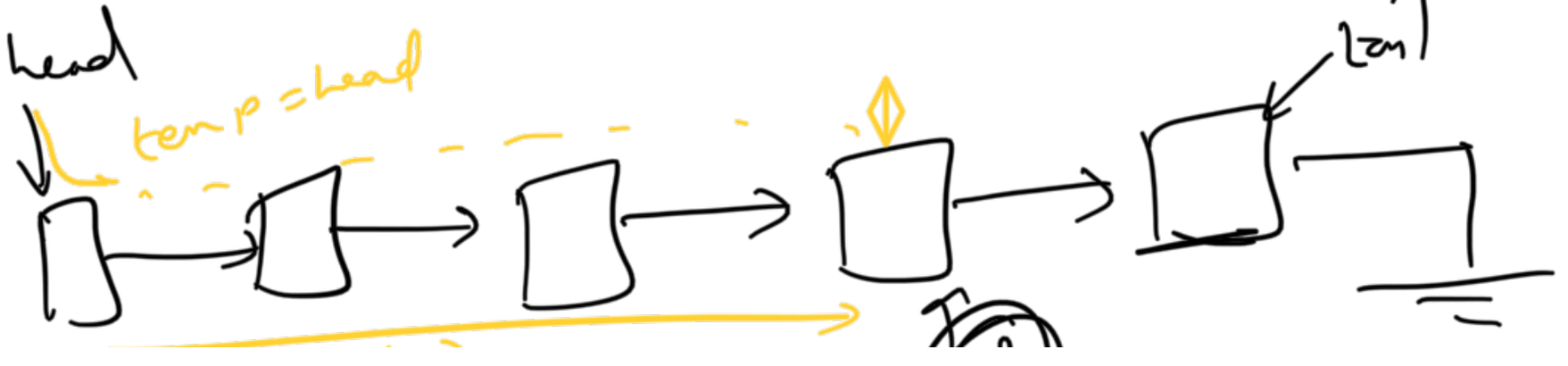
temp = NULL)

tail = ten



tail = tail.prev

tail.next = NULL



O(N)

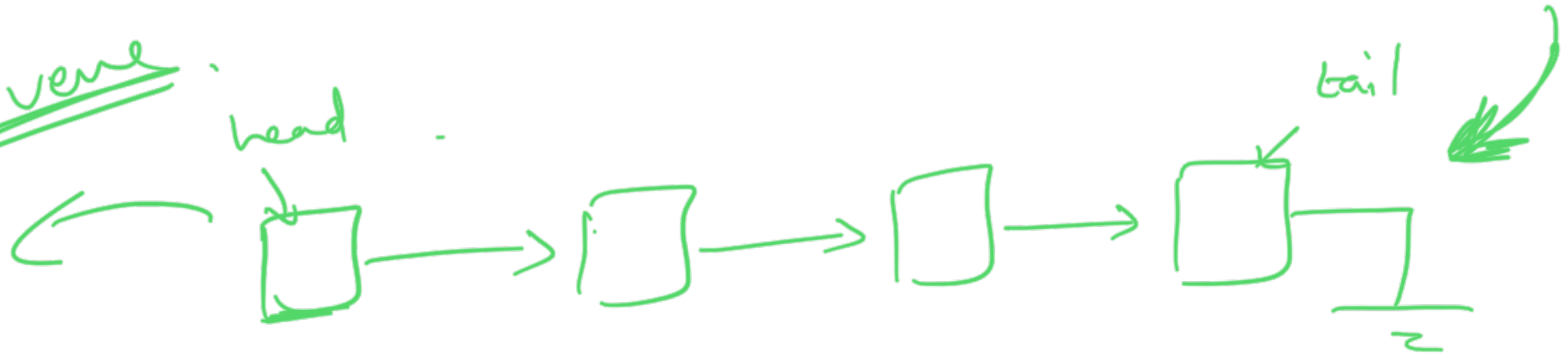
While(temp.next!=...)

Break over.

Queue

head

tail



remove at head

pop() → O(1)

en(5)
en(7)
en(11)
en(2)
de()
de()

en(8)

de()

5

head = head.next

head

head

2

8

head

insert from back / tail

insertion at tail => O(1)

tail

tail

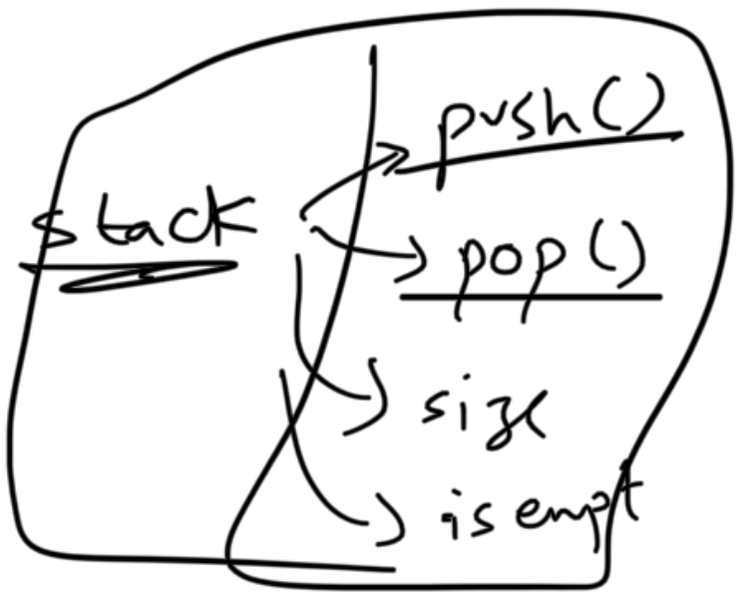
Queue

T.C => O(1).

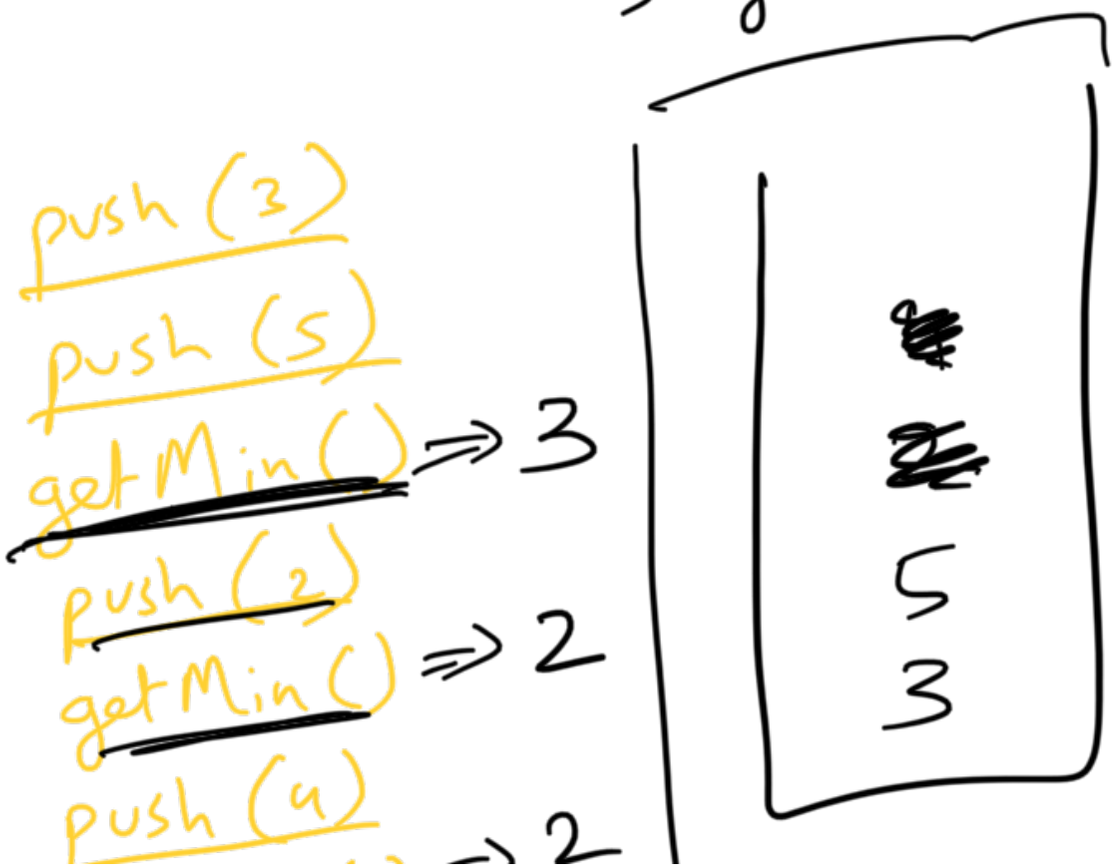
S.C => O(N) -> dynamic

Q. (Flipkart)

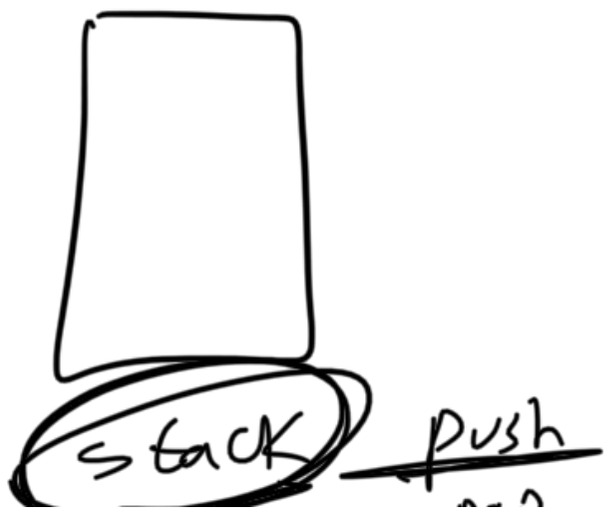
Implement a stack which supports 'getMin()' function.
(can use extra space)



class NewStack
- push()
- pop()
- getMin() ⇒ curr min in stack.



int.



getMin()
pop()
pop()
getMin ⇒ 3

pop

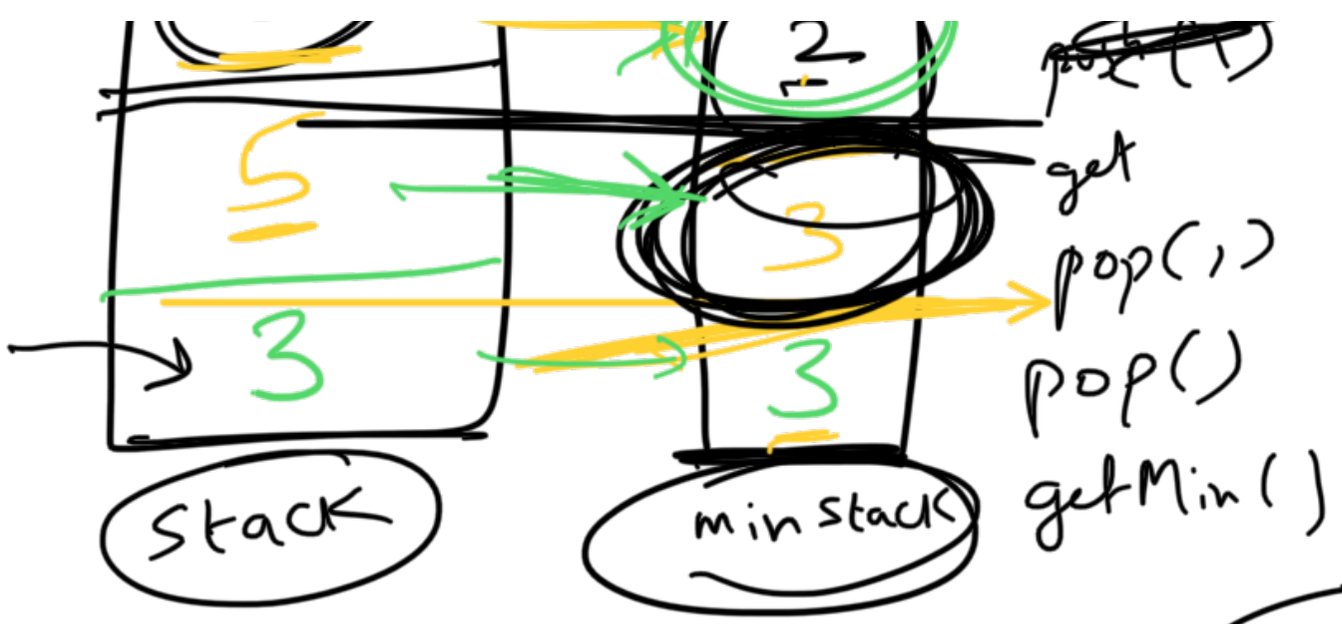
min ⇒ O(N)

glob min = 2 → 3

push(1)
push(3)
push(5)
push(2)
getmin()
pop()
getMin()

O(N)

obs.



Use 2 stack.

push(a)
pop()
getMin()

Class newStack

{ stack<int> st, min_st;

void push(int a)

{ st.push(a)

min_st.push(min(a, min_st.top()))

}

void pop()

{ st.pop()

O(1)

min_st.pop()

}

int getMin()
{
return min_st.top()
}

}

push(3)

push(5)

push(3)

getMin()

push(2)

getMin() ⇒ 2

push(8)

getMin() ⇒ 2

pop()

pop()

getMin() ⇒ 3

st.push(a)

Min_st min 5 min_stack.top()

min(5,3)

int main()

{

newStack()

max St. push(+int_max)

for(

push

}

getMax()

max_st

Doubts.