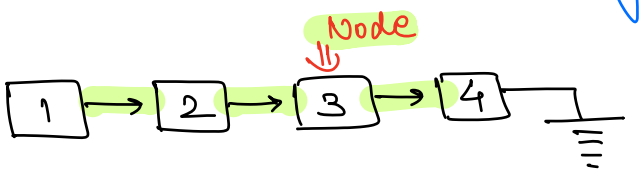
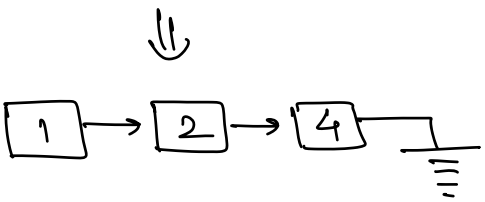
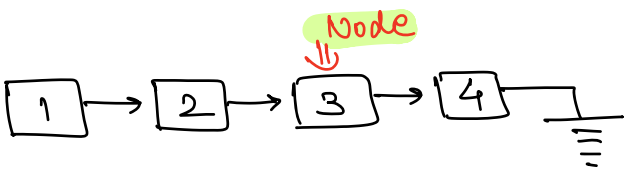
Q Given a node of a L.L, Delete this node from the L.L.

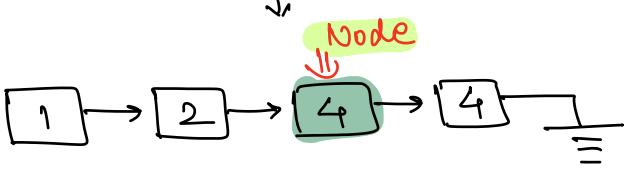
Google

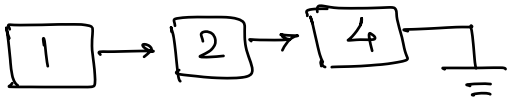
Will never be the last node of the L.L



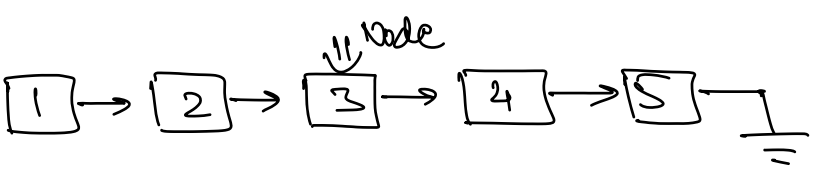


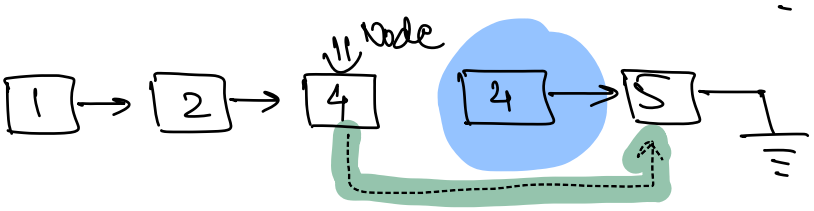


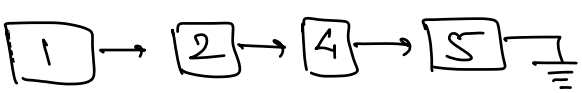




#





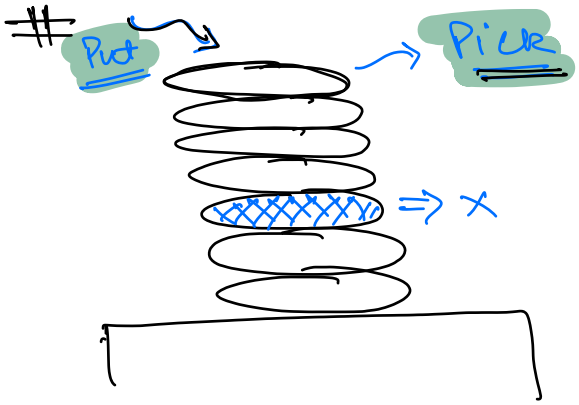


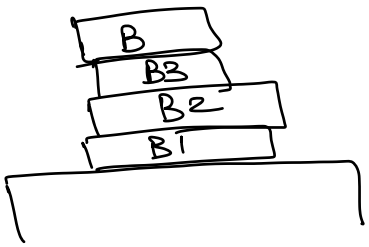
```
void delete(node) {
    node.data = node.next.data;
    node.next = node.next.next;
}
```

TC : O(1)

SC : O(1)

⇒ If given node is the first Node, then this code will work ✓





Last In, First Out (LIFO)

⇒ Stack

Stack → Push(x) | Insert(x)
→ pop() | delete | removal.

} Main Operations

⇒ top() | peek()

⇒ isEmpty()

⇒ size()

Abstract Data Type (ADT)

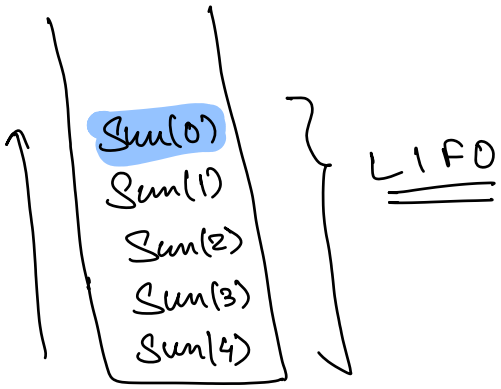
↳ Stack
↳ Queue

Applications

1) function | call Stack

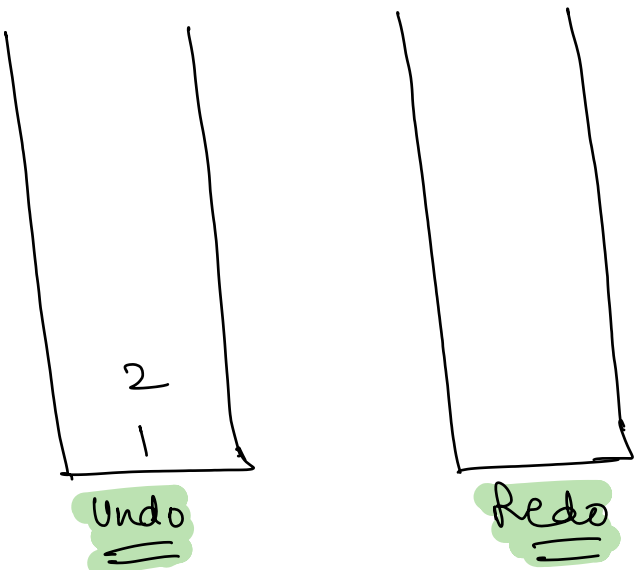
Sum(N) = N + Sum(N-1)

Sum(4)



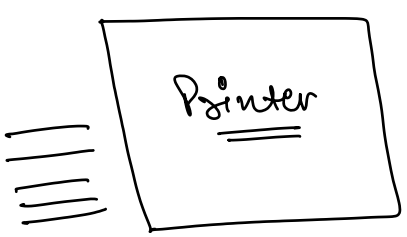
2) Undo | Redo Operation

1
2
3
4



3) Back button on web browsers

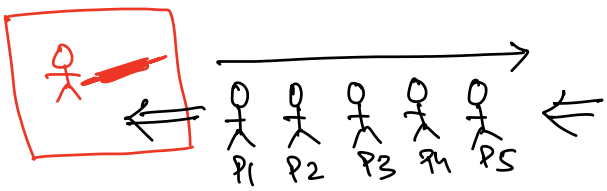
#



Vijay: Doc I ①
Gaurav: Doc II ②
Pratik: Resume.pdf ③

First In, First Out

⇒ Queue



FIFO

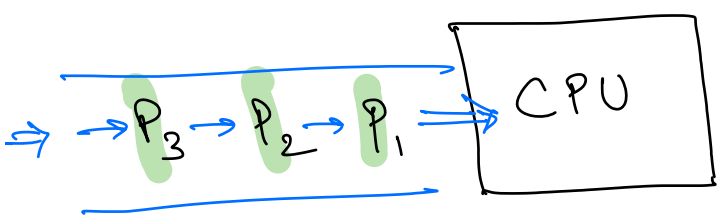
Queue (ADT)

- Enqueue (Push | Insert)
- Dequeue (Pop | delete | remove)
- ⇒ size()
- ⇒ isEmpty()

⇒ ⇒

Applications

1) Kafka | RabbitMQ | ActiveMQ (HLD)
   - → Implementation of Queue
2) Scheduling Algo's



3) Printer :

Implementations

1) STACK

→ 1) Arrays
2) Dynamic Arrays (Array List / Vector)
3) Linked List

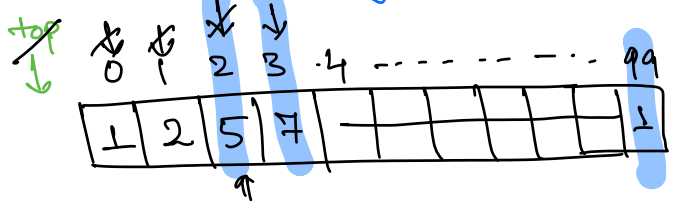
⇒

```
int arr[100];
```

→ Max size of the stack.

```
int top = -1;
```

// If top == -1, Stack is Empty.




```
void push(int x){
  if(top < 99)
    top++;
    arr[top] = x;
  }
}
```

⇒ TC : O(1)

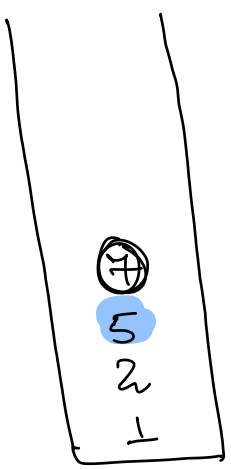
```
Void pop(){
  if(top >= 0)
    top--;
}
```

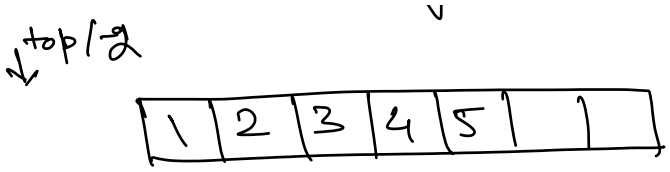
TC : O(1)

for underflow.

To overcome the issue of overflow.

```
push(1)
push(2)
push(5)
push(6)
pop()
push(10)
pop()
push(7)
```





pop() →
pop()
pop()
pop()
pop()
pop() X
push(10) ⇒ top = -1
a[top] ≠ 10

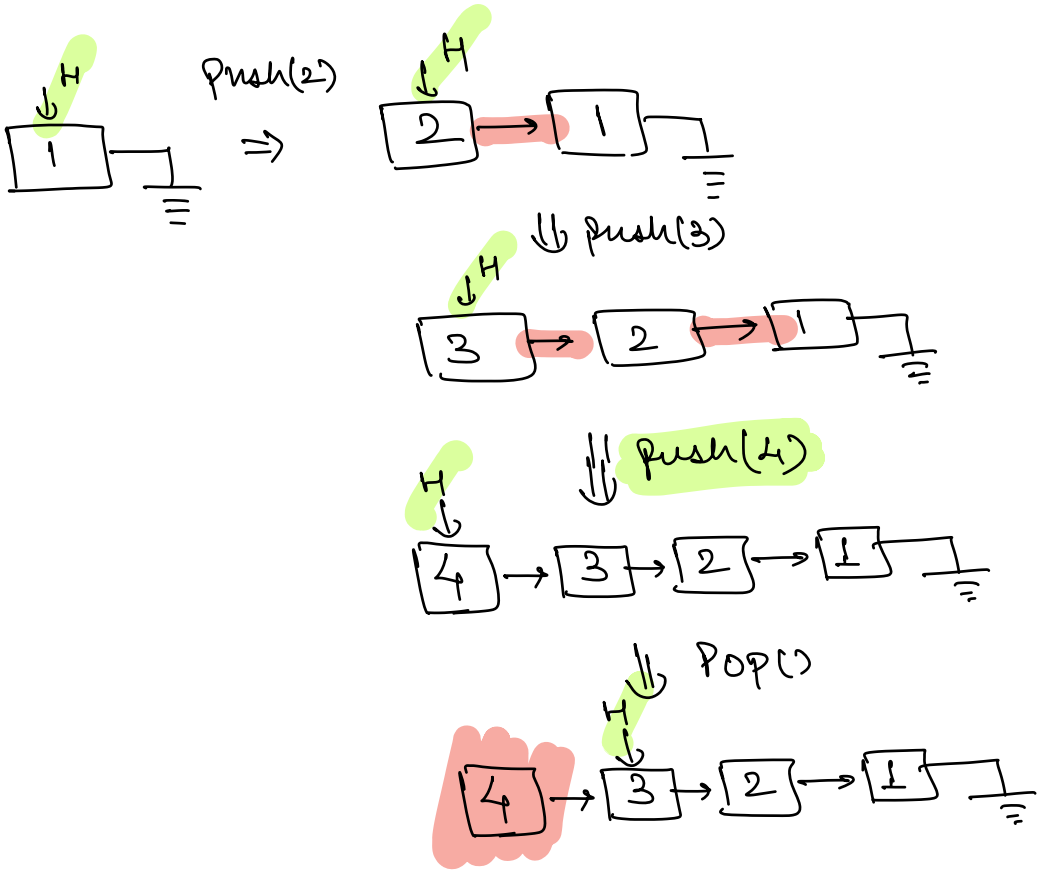
⇒ If we use Dynamic Arrays
↳ No overflow
⇒ Underflow can happen

Stack using L.L

```
Class Node {
    int data;
    Node next;
    Node (d) {
        this.data = d;
        this.next = null;
    }
}
```

this → current Object.

push(x) ⇒ Add at front.
pop() ⇒ Delete from front
} O(1)



Java ⇒ Collections

```
Stack<Int> st = new stack<>();
st.push(5)
st.push(1)
st.push(2)
st.pop()
```

C++

```
Stack<Int> st;
```

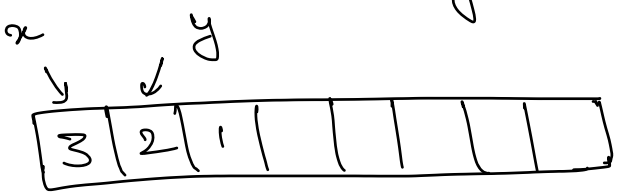
Queue Implementation

1) Arrays

Insert at front ⇒ O(N)

Delete from last ⇒ O(1)

y--



equeue(1)
equeue(2)
equeue(3)

⇒ 3 2 1 ⇒