Stack Data Structure

Q. Why to study so many data structures?

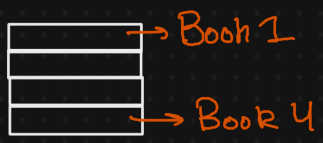
Choice of data structure ⇒ efficient problem solving

Stack

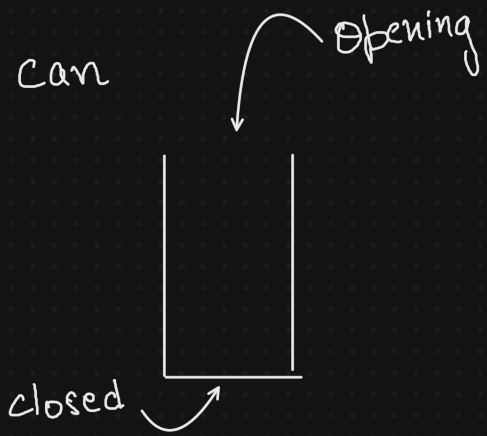
→ Linear Data Structure

example of stack

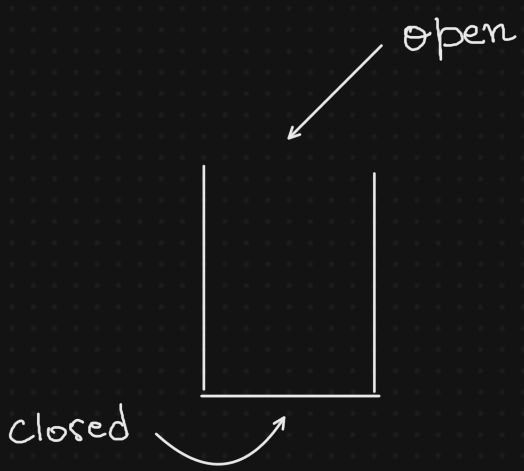
1. Stack of Plates
2. stack of chairs
3. stack of books



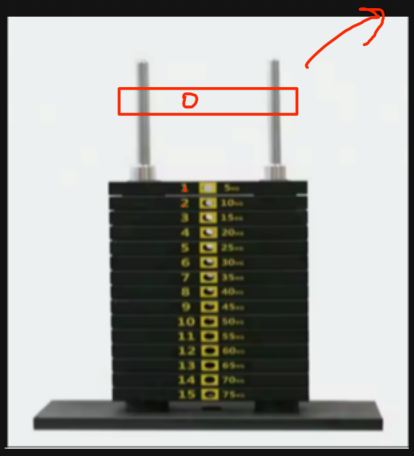
Pringles can



Bread Packet



More examples of Stack



Tower of Hanoi

LIFO / FILO : element which is put up last will be the first one out eg. plate O

Abstract Data Type : Abstract data type is a type for objects whose behaviour is defined by set of operation. as abstract, no knowledge about how they are performed.

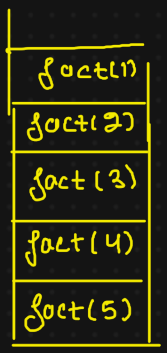
Array list Linked list

Industry example of stack

→ undo/redo operation

→ call stack in programming

→ matching html tags / brackets

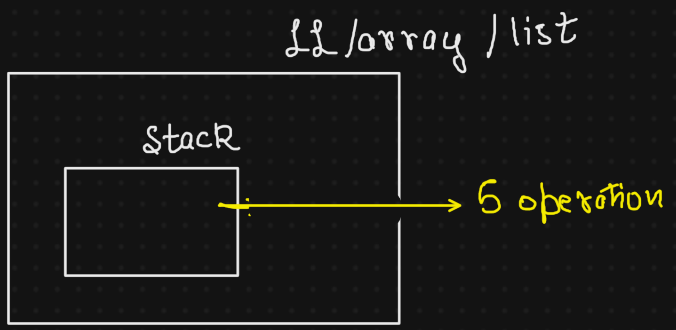


5 → 4 → 3 → 2 → 1

1 → 2 → 3 → 4 → 5

Operation in Stack

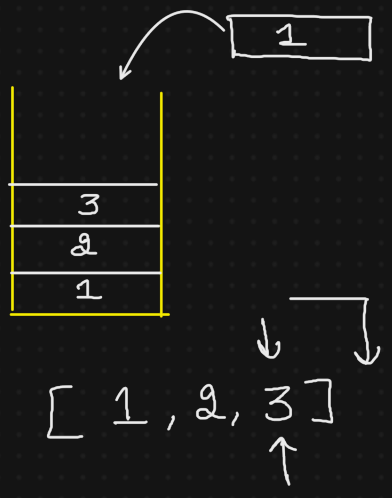
1. Top / peek → See the top element
2. Pop → removing the top element
3. Push → insert an element on top
4. Size → returns no. of elements in our stack
5. Empty → if our Stack is empty or not



We will make sure that these 5 operation are optimized.

Stack Implementation : Using Inbuilt array

[ ]



1. Push

[1, 2, 3]

2. Top

```
          0  1  2  3  4
l1 = [ 1, 2, 3, 4, 5]
len(l1) = 5   -3 -2 -1
```

how to access the last element of list ?

l1 [4] ⇌ l1 [-1]

l1 [len(l1) - 1]

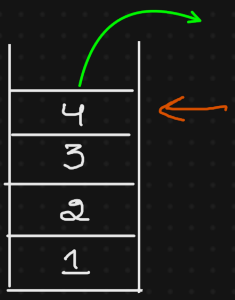
3. Pop() → getting the top element and removing it as well from our stack

Stack Implementation : Using Linked List

1. push()

It will be having $O(n^2)$ complexity.

Let us maintain a tail variable $\longrightarrow O(1)$



2. size()

will be O(n) in complexity.

size / len will be maintained

3. is empty()

4. Peek / Top

5. Pop()

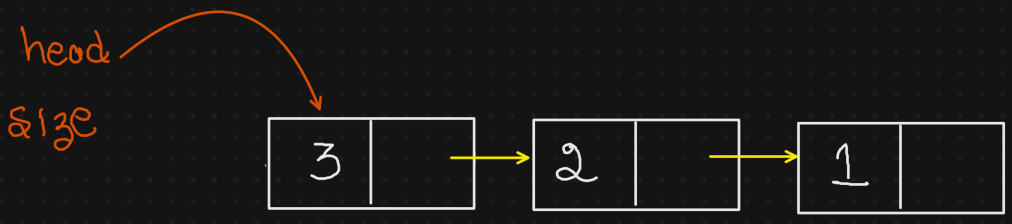
O(n) Use a temp to travel

Due to pop() function, we cannot do it in better complexity than O(n), which is double.

Would like to reduce it as well if we are implementing stack using Linked List

Stack using Linked List : Optimized

When we insert/delete from ll head, complexity is O(1).



push() → insert at end O(1)

pop() → we can just move head to next element O(1)

```
head = head.next
```

top()/peek → head.data

size() → we can maintain size ⇒ O(1) approach

isempty() → size == 0