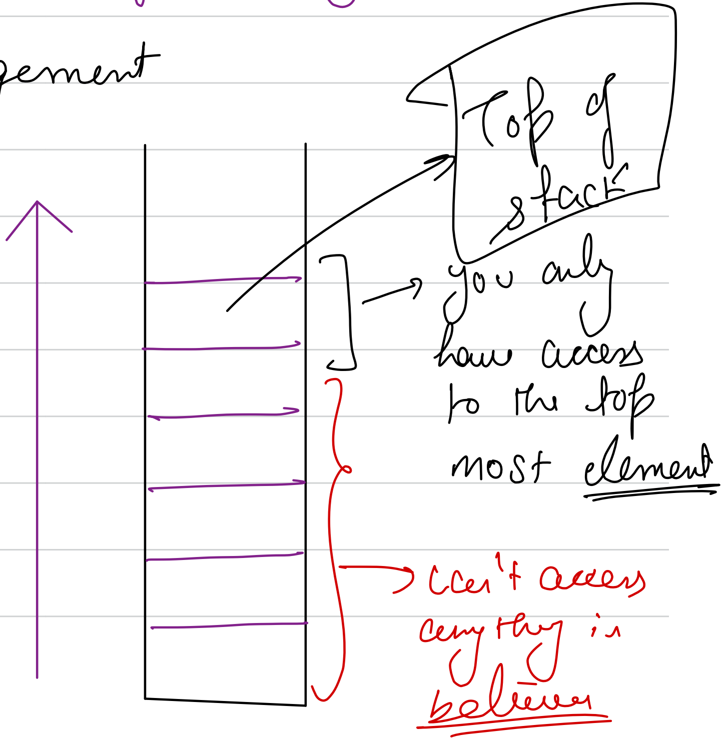
Stacks → Linear Data Structure, that supports LIFO (Last In First Out) kind of usage.

↳ Recursion & memory Management

↳ Android & other OS

↳ OS algorithms



Top of stack
you only have access to the top most element
can't access anything in between

↳ How we can create our own stack.

↳ Some Inbuilt python implementation of stack

↳ Problem solving with stack.

$\hookrightarrow$ ① Add an element $\longrightarrow$ Top of stack

$\hookrightarrow$ push()

② Delete an element $\longrightarrow$ Top of Stack

$\hookrightarrow$ pop()

③ Get the topmost elemet $\rightarrow$ Top of stack

$\hookrightarrow$ peek() / top()

(1) Stacks can be implemented by arrays / lists

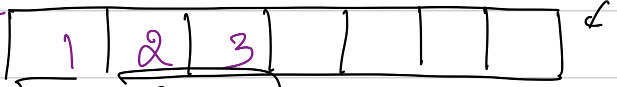
(2) Also via linked list

(3) Using Queue (we will study this later)

Stack Using Arrays (fixed size)

How to stop user from accessing the array

st



underflow

top = 1

Overflow

1, 2, 3, 4, 5, 6, 7

top -= 1

top += 1
st[top] = element

push

pop

peek → return → st[top]

$\boxed{\text{OOP}}$ ⟶ private data member

By default ⟶ public

no one outside the class will be able to access the data member.

→ Access Modifier

push → O(1)
pop → O(1)
top/peek → O(1)

↳ Issue → size is fixed, so we cant grow/shrink

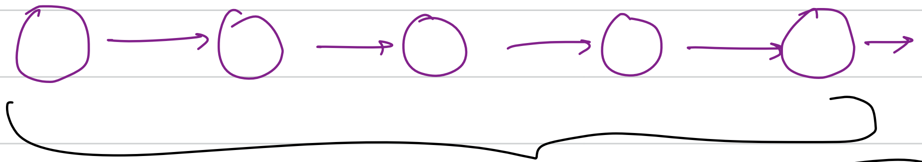
Microsoft

↳ offcampus / oncampus → online coding → [Flyer round

↳ inbuilt DS ↳ DTU

How you can make it ??

Stack Using LL

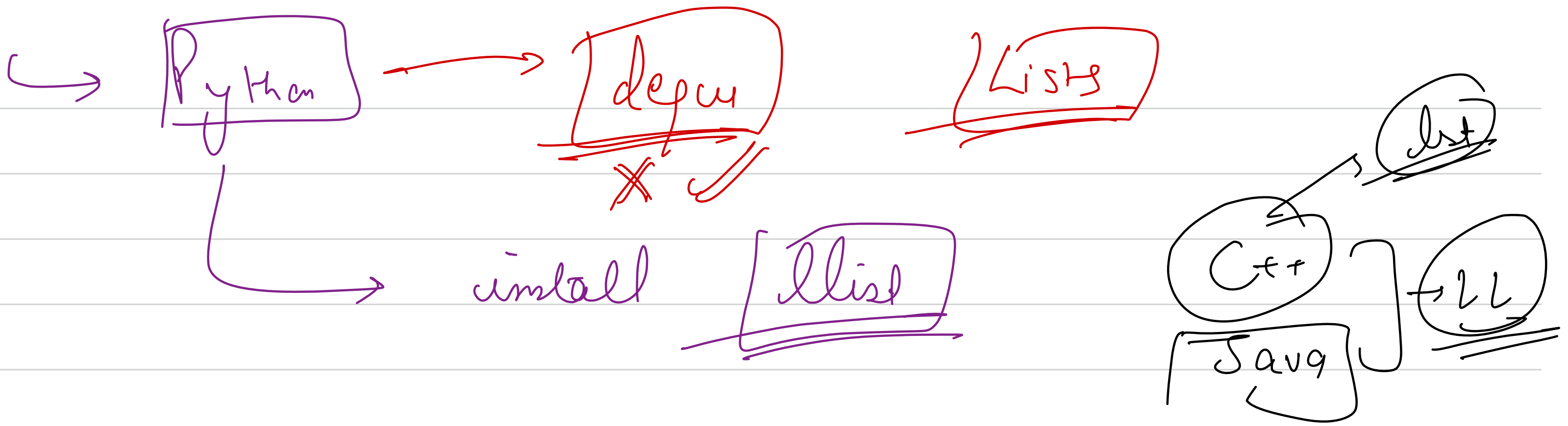


Stack

push → add At head → O(1)

pop → remove from head → O(1)

1. Size can grow
2. – non-contiguous memory loc



Python
deque
Lists
install
llist
C++
Java
list
LL

$$\begin{tikzcd}
\text{list} \arrow[r] & \text{append} \arrow[r] & \text{push} \\
 & \text{pop} \arrow[r] & \text{pop} \\
 & \text{peek} \arrow[r] & \text{li[-1]}
\end{tikzcd}$$

Q:- Given a String S, containing characters → '(', '[', '{', '}', ']', ')'. Determine whether the string is balanced or not??

$() \rightarrow$ True

$(] \rightarrow$ False

$()[]\{\} \rightarrow$ True

$(([\{\}])) \rightarrow$ True

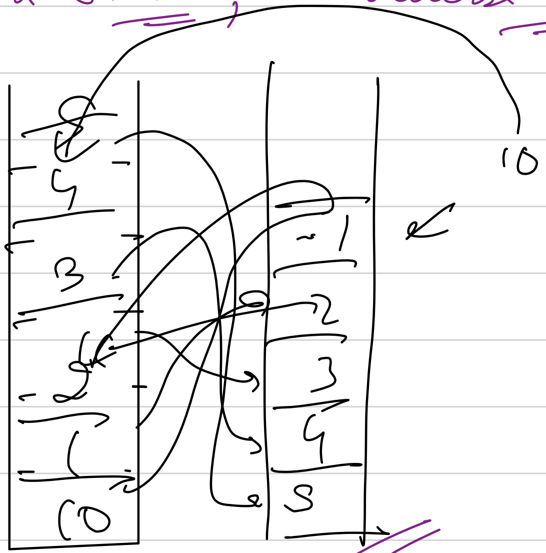
balanced ↓

① open brackets must be closed by same type of bracket

② order of closing should be correct

↪ generally in problems when we have to trace the last element relevant or, when you have to keep track of parent

Q. Given a Stack, reverse it.



→ insert at Bottom

iteratively
Recursively

→ 1 elmt → O(n)

→ O(

```
insertatBottom(stack s, n)
    if (s.isEmpty())
        s.push(n)
        return
    }
    temp = s.pop()
    insertatbottom(stack, n)
    s.push(temp)
```

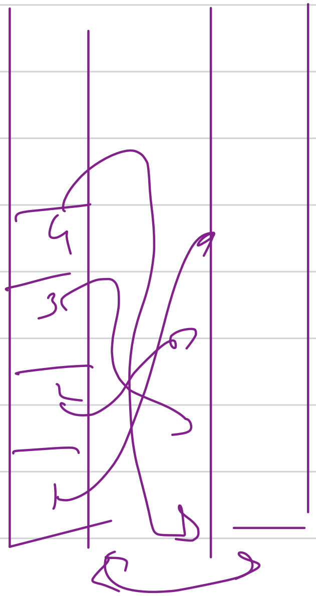
```
insert at bottom ( Stack s , int n )
    if ( s. isEmpty() )
        s. push( n )
        return
    temp = s.pop()
    insert at bottom ( s , n )
    s. push( temp )
```

TC
$O(n^2)$
$\underline{\underline{\quad}}$
SC $\rightarrow O(n)$

recursive soln

Stack

auxillary stack



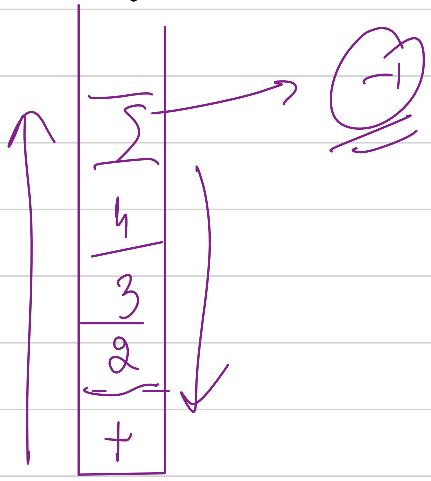
→ O(n) TC

O(n) SC

Qn Given a stack → Sort the stack

Try doing by max 2 stack (1 given + 1 auxillary)

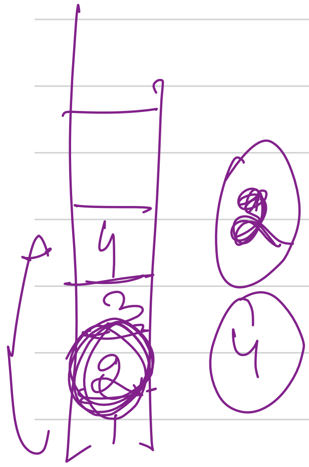
Try Recursively



<-stack

```
insertElementInSortedForm(stack s, int el)
        if (s.isEmpty() OR st.top() < element)
                s.push(el)
        else
                temp = s.pop()
                insertElementInSortedForm(s, el)
          ->    s.push(temp)
```

Base Case



```
SortStack(Stack s)
        if (not s.isEmpty())
                temp = s.pop()
                SortStack(s)
                insertElementInSortedForm(s, temp)
```

SC $O(n)$

TC $O(n^2)$