CS & IT ENGINEERING

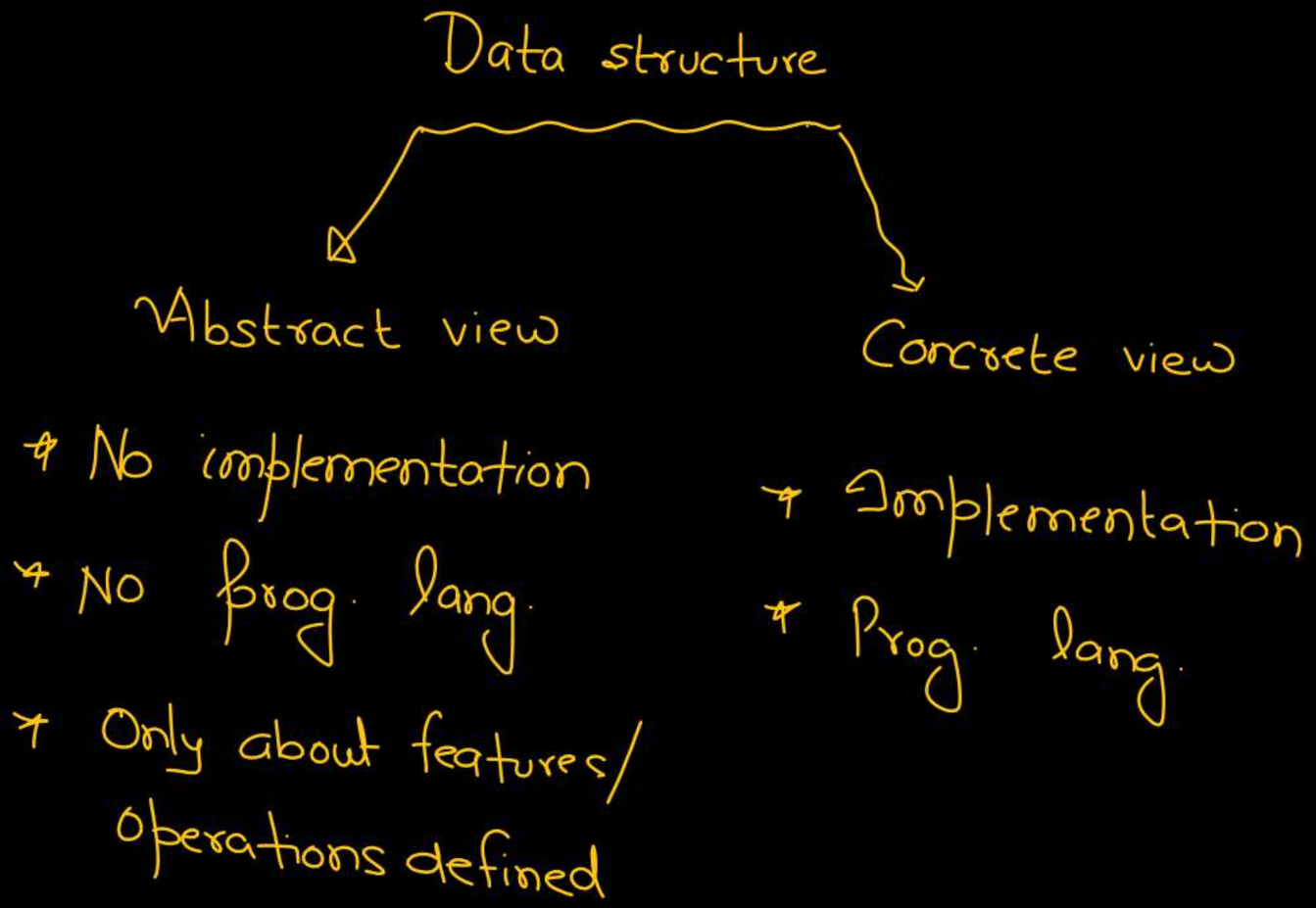
Data Structure & Programming

Stack and Queue

Lec- 01

By- Pankaj Sharma sir

Stack-I


Data structure
Abstract view
* No implementation
* No prog. lang.
* Only about features/ operations defined
Concrete view
* Implementation
* Prog. lang.

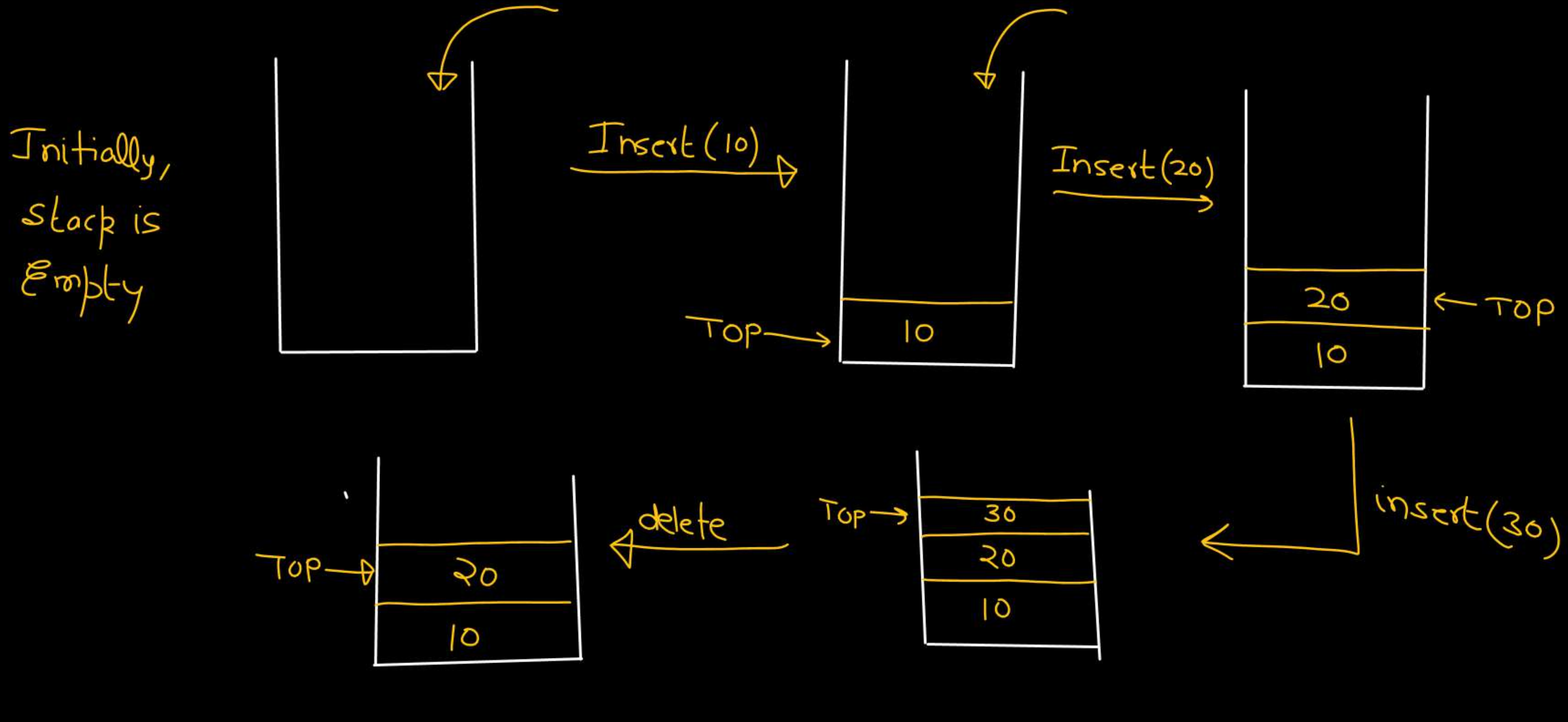
stack

* Linear data structure
* Works on Last-In First-out policy. (LIFO)
* Order of deletion : reverse order of insertion
* Both insertion & deletion are performed only at one end called as TOP of stack.

Stack as ADT

TOP : element added most recently

Stack of numbers



Insert → Push

delete → Pop

IsEmpty() :

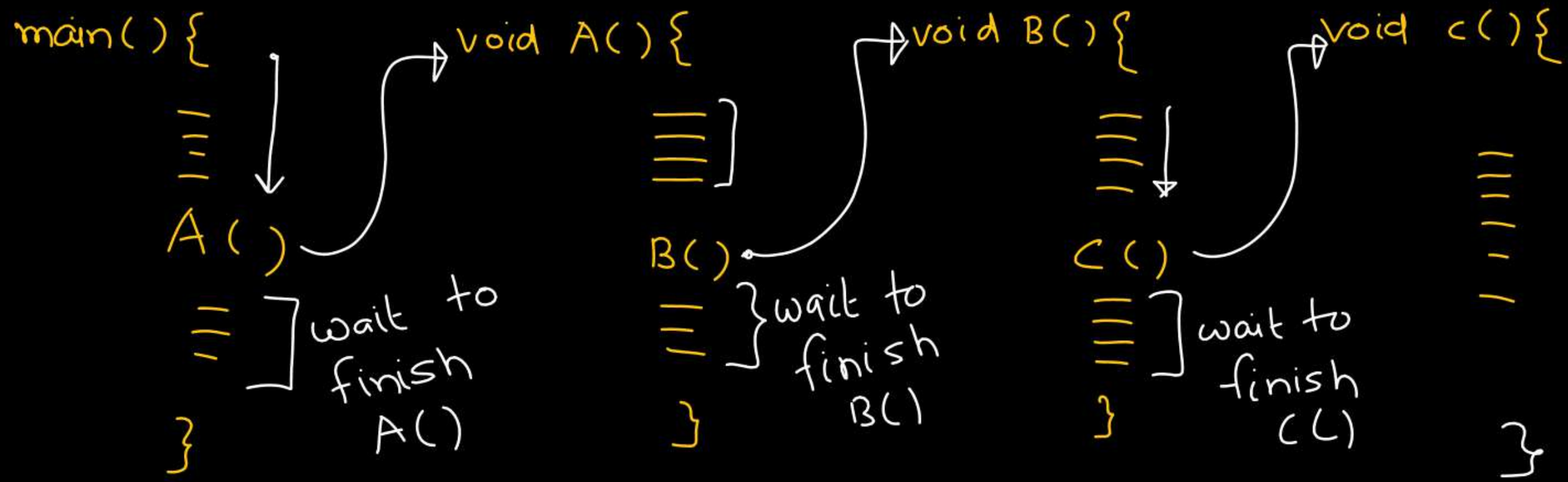
- → True if stack is empty
- → False if stack is not empty

IsFull() :

Applications

1) Recursion / Function
2) TOH
3) Infix to prefix
4) Prefix Evaluation
5) Infix to postfix
6) Postfix Evaluation
7) Balanced paranthesis check

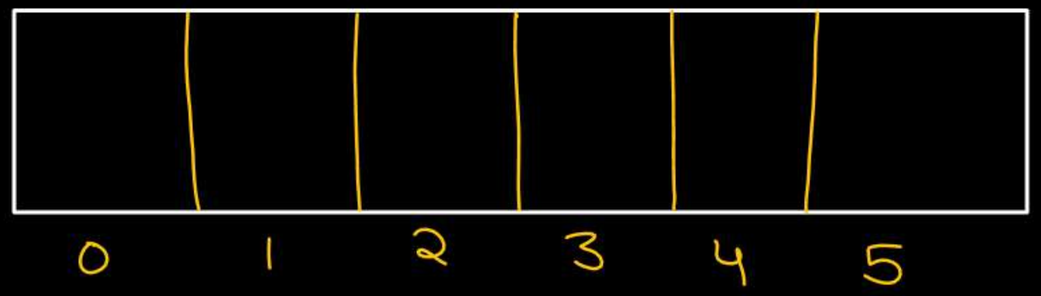
{ wait कराना
To delay
To postponed decisions }


main(){
A()
}
wait to finish A()
void A(){
B()
}
wait to finish B()
void B(){
C()
}
wait to finish C()
void C(){
}

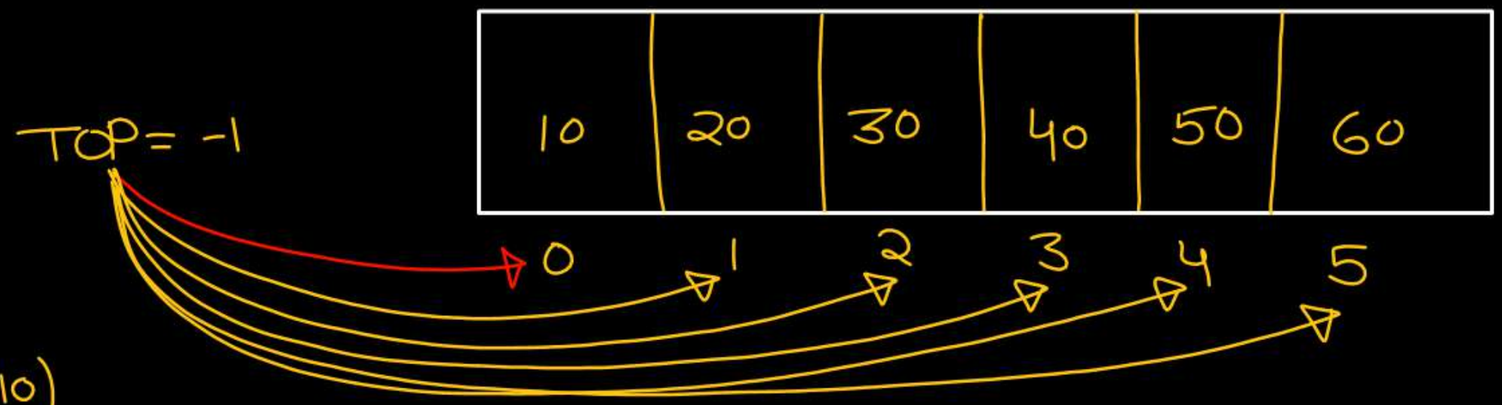
Implement stack using array

```
#define SIZE 6
int STACK[SIZE];
```

TOP: represent index of most recently added element.



Initially, TOP = -1



SIZE-1

(i) Push(10)

(ii) Push(20)

(iii) Push(30)

(iv) Push(40)

(v) Push(50)

(vi) Push(60)

(vii) Push(70)

```
void Push(int x){
    TOP = TOP + 1 ;
    STACK[TOP] = x ;
}
```

DIGIT

```
void Push(int x){
        if(TOP == SIZE-1)
                return;
        TOP = TOP+1;
        STACK[TOP] = x;
        }
```

loop ✗
recursion ✗

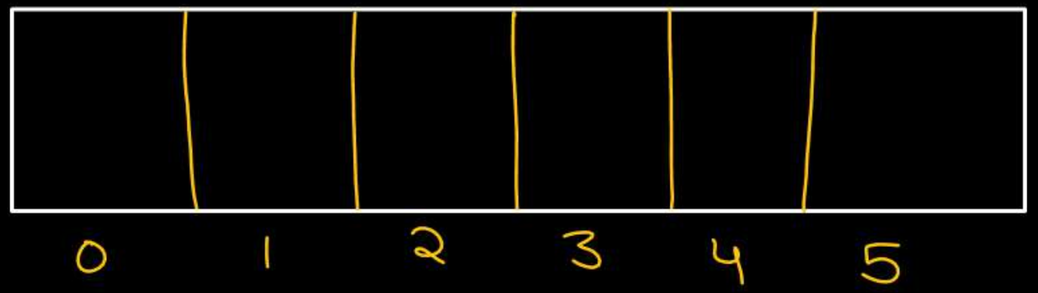
O(1)

(i) Pop()

return the element

```
int Pop() {
    int temp;
    temp = STACK[TOP];
    TOP = TOP-1
    return temp;
}
```

TOP = -1



(i) Pop()

constant time

int Pop(){ int temp;
if(TOP == -1)
return INT_MIN;
temp = STACK[TOP];
TOP--;
return temp;
}

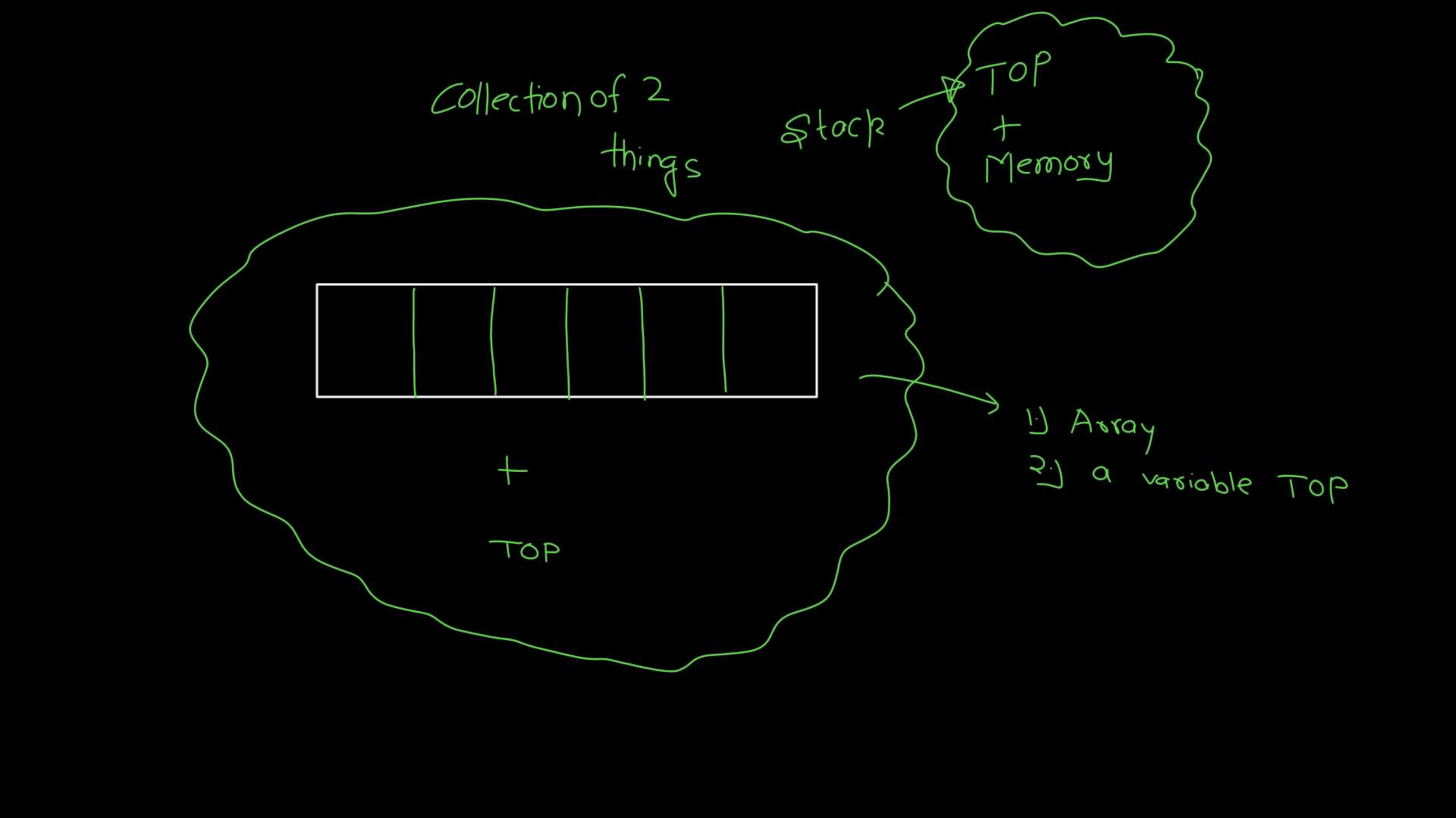
int Pop() { return the element

int temp;
temp = STACK[TOP];
TOP = TOP-1
return temp;
}

```
#define SIZE  10
int STACK[SIZE];
int TOP = -1;
void Push(int x){
        if(TOP == SIZE-1)
            return;
        TOP++;
        STACK[TOP] = x;
    }
int Pop() {
    ==
}
```

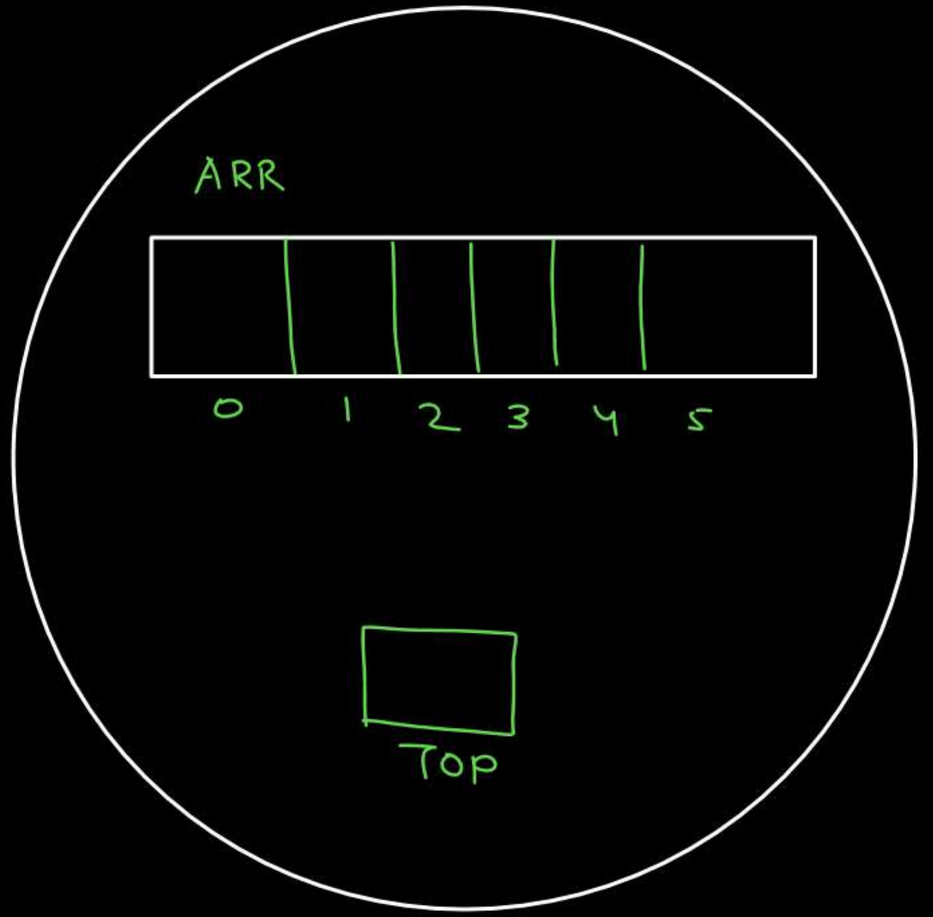
Problem ?

```
void main() {
    ==
}
```

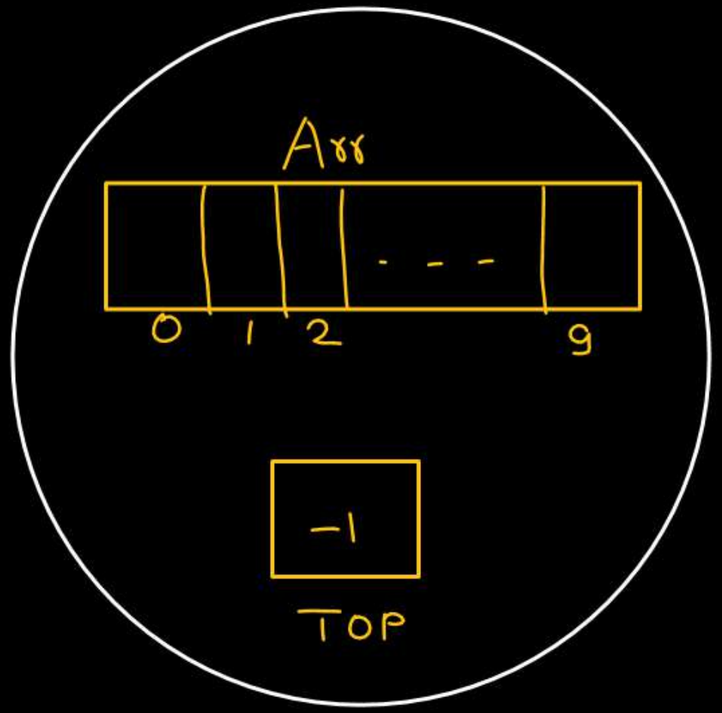

Collection of 2 things
Stack
TOP
+
Memory
+
TOP
1.) Array
2.) a variable TOP

Collection of diff. types of elements.

```
#define SIZE 10 ;

struct STACK {

    int Arr[SIZE] ;
    int TOP ;

};
```



```
struct STACK{
        int Arr[10];
        int TOP;
        };

void main(){
        struct STACK S1,S2;
        S1.TOP = -1;
        S2.TOP = -1;
```



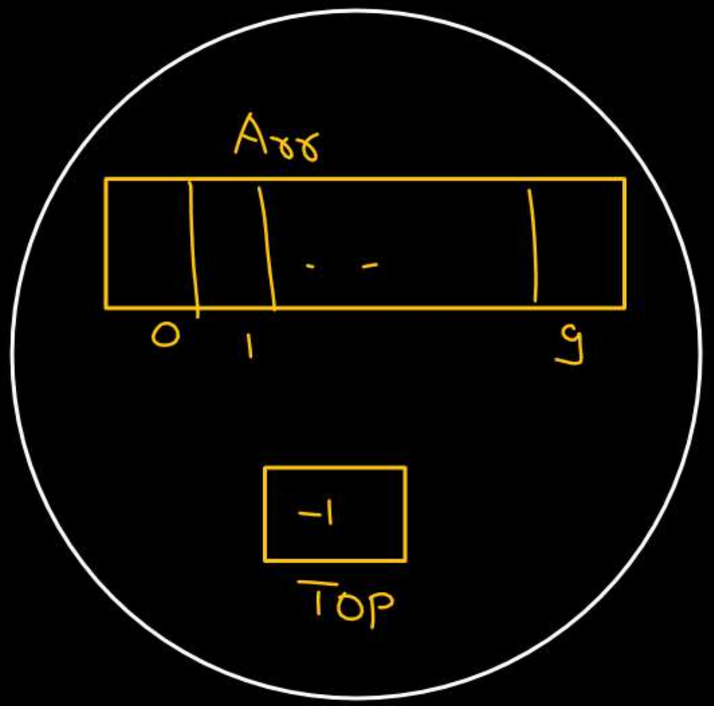
Call by value

Push( )

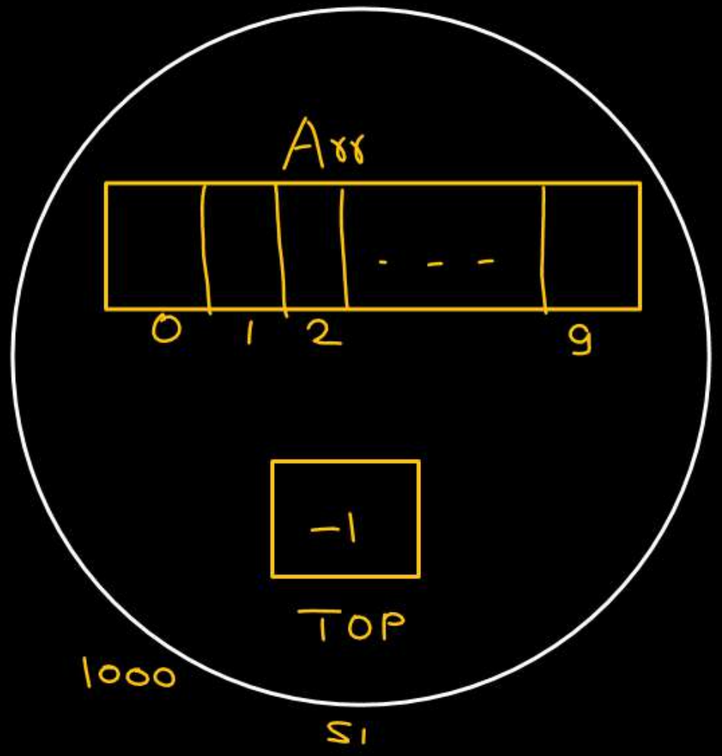
In which stack we want to push

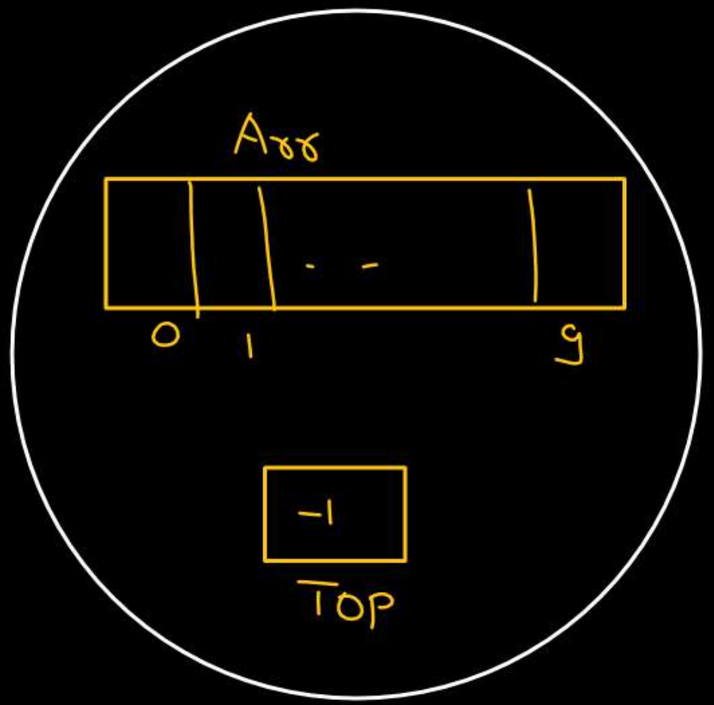
Push(S1, 10);

Push(S2, 20);

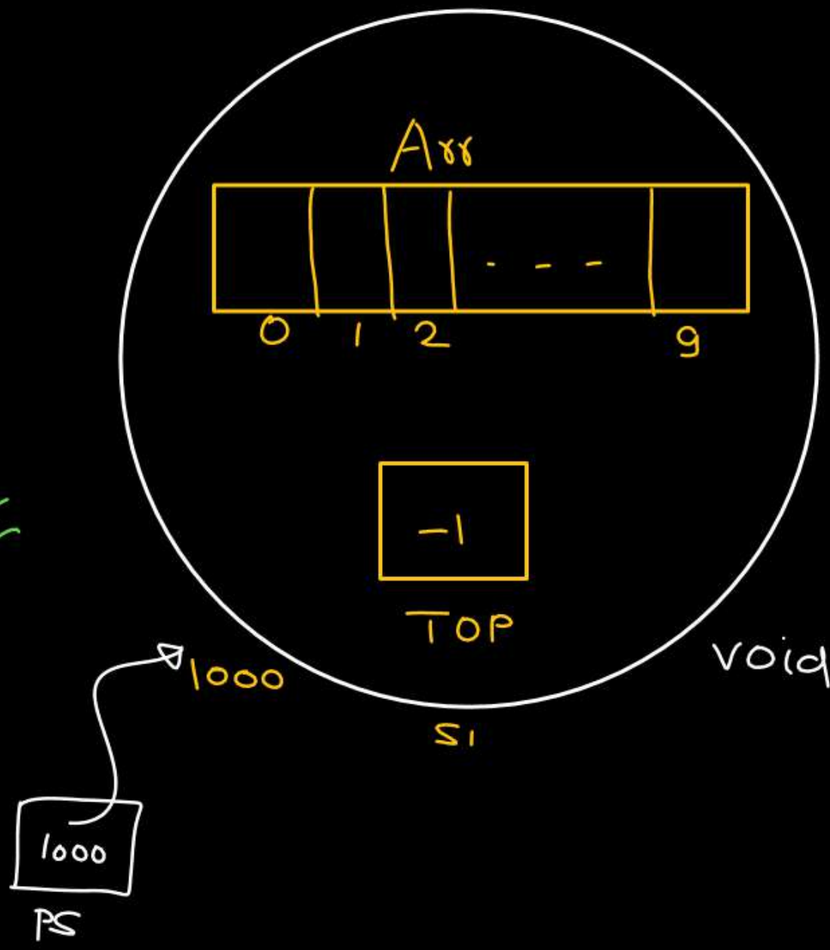


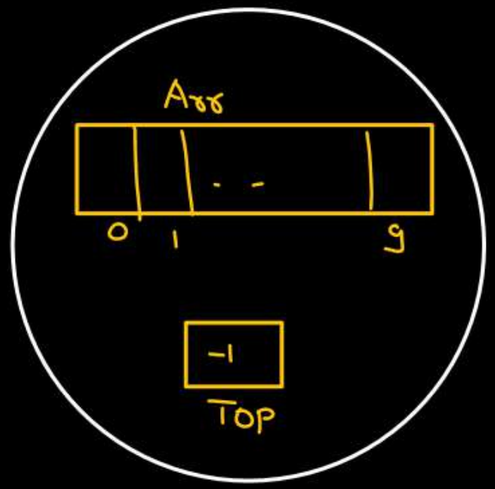
```
struct STACK{
        int Arr[10];
        int TOP;
        };
void main(){
Struct STACK S1,S2;
   S1.TOP=S2.TOP=-1
Push(&S1,10);
```





```
#define SIZE 10
struct STACK{
        int Arr[10];
        int TOP;
        };
void main(){
struct STACK S1,S2;
S1.TOP=S2.TOP=-1;
Push(&S1,10);
```






```
void Push(struct STACK* PS, int x)
{
    if(PS->TOP==SIZE-1)
        return;
    PS->TOP=PS->TOP+1;
    PS->Arr[PS->TOP]=x;
}
```

stack permutation

Order of insertion of given elements is fixed.

What could be possible order of pop (stack permutation)

1, 2, 3 →

$n = 3$ 1, 2, 3

order of insertion 1, 2, 3 →

you can pop at any time

Possible permutation

(i) 1, 2, 3

(ii) 1, 3, 2

(iii) 2, 1, 3

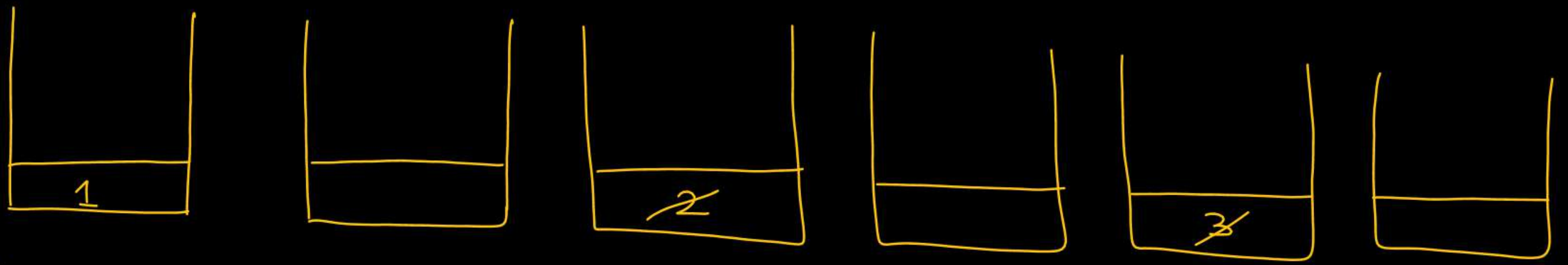
(iv) 2, 3, 1

(v) 3, 1, 2

(vi) 3, 2, 1

} 3!

(i) 1, 2, 3 ✓ ✓



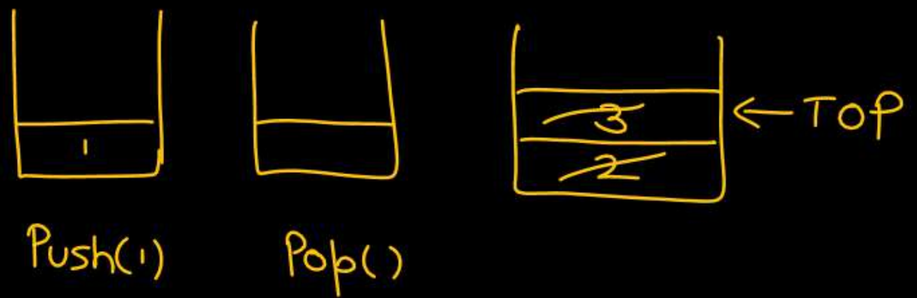
1, 2, 3

a) Push(1)
b) Pop( )
c) Push(2)
d) Pop( )
e) Push(3)
f) Pop( )

1, 2, 3 is a valid stack Permutation.

(ii) 1,3,2 ✓ 1,3,2

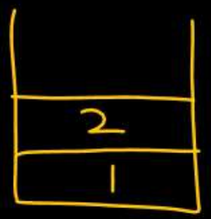
a) Push(1)
b) Pop()
c) Push(2)
d) Push(3)
e) Pop()
f) Pop()



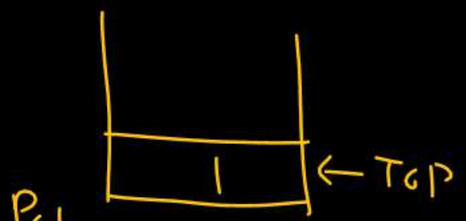
1,3,2 is also a valid stack permutation.

(iii) 2,1,3

Push(1)
Push(2)
Push(3)
Pop()



Pop

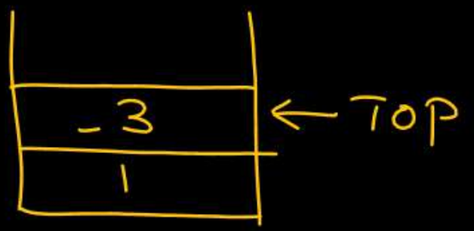


Pop

2,1,3 is $\Rightarrow$ valid stack permutation

(N) 2, 3, 1 ✓

Push(1)
Push(2)
Pop()
Push(3)
Pop()
Pop()



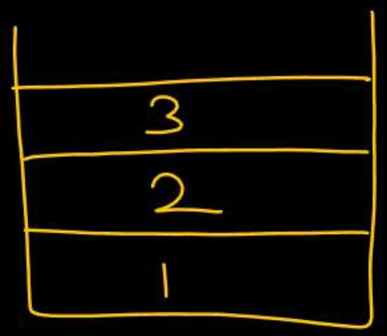
Valid stack permutation

(V) 3,1,2 ✓

Push(1)

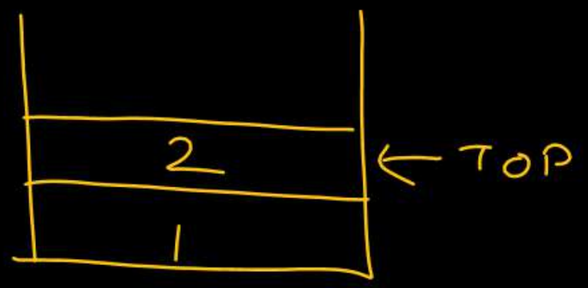
Push(2)

Push(3)



Pop()

Can we pop 1 before 2?



3,1,2 is not a valid stack formation.

vij 3, 2, 1

Push(1)

Push(2)

Push(3)

Pop()

Pop()

Pop()

$n = 3$

(1, 2, 3)

1, 2, 3 ⟶

6 Permutation

(i) 1, 2, 3
(ii) 1, 3, 2
(iii) 2, 1, 3
(iv) 2, 3, 1
(v) 3, 1, 2
(vi) 3, 2, 1

Out of these 6 ⟹ Only 5 are valid stack permutation.

$n!$ Permutation ⟹ $\frac{2nC_n}{n+1}$ stack permutation

1, 2, 3, 4 →

a) 2, 1, 4, 3

b) 2, 1, 3, 4

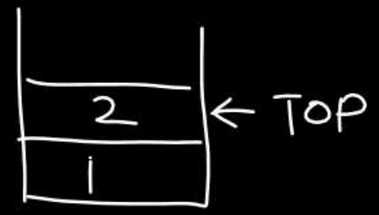
c) 4, 3, 1, 2 ✓

d) 4, 3, 2, 1

Push(1), Push(2), Pop(), Pop(), Push(3), Push(4), Pop(), Pop()

Push(1), Push(2), Pop(), Pop(), Push(3), Pop(), Push(4), Pop()

Push(1), Push(2), Push(3), Push(4), Pop(), Pop(),



Push(1), Push(2), Push(3), Push(4),
Pop(), Pop(), Pop(), Pop()

Q

Insertion : 1, 2, 3, 4 ⟶

How many valid SP are possible ?

$$\frac{{}^{2n}C_n}{n+1}$$

$$\frac{{}^{8}C_4}{5}$$

$$= \frac{1}{5} \times \frac{8!}{4! \times 4!}$$

$$= \frac{8 \times 7 \times 6 \times \cancel{5} \times \cancel{4!}}{5 \times 4! \times \cancel{4!}}$$

$$= \frac{8 \times 7 \times 6}{\cancel{3}\cancel{4}} = 14$$

Infix, prefix & postfix

Infix : 2 + 3 : operator is in-between operands.

operands

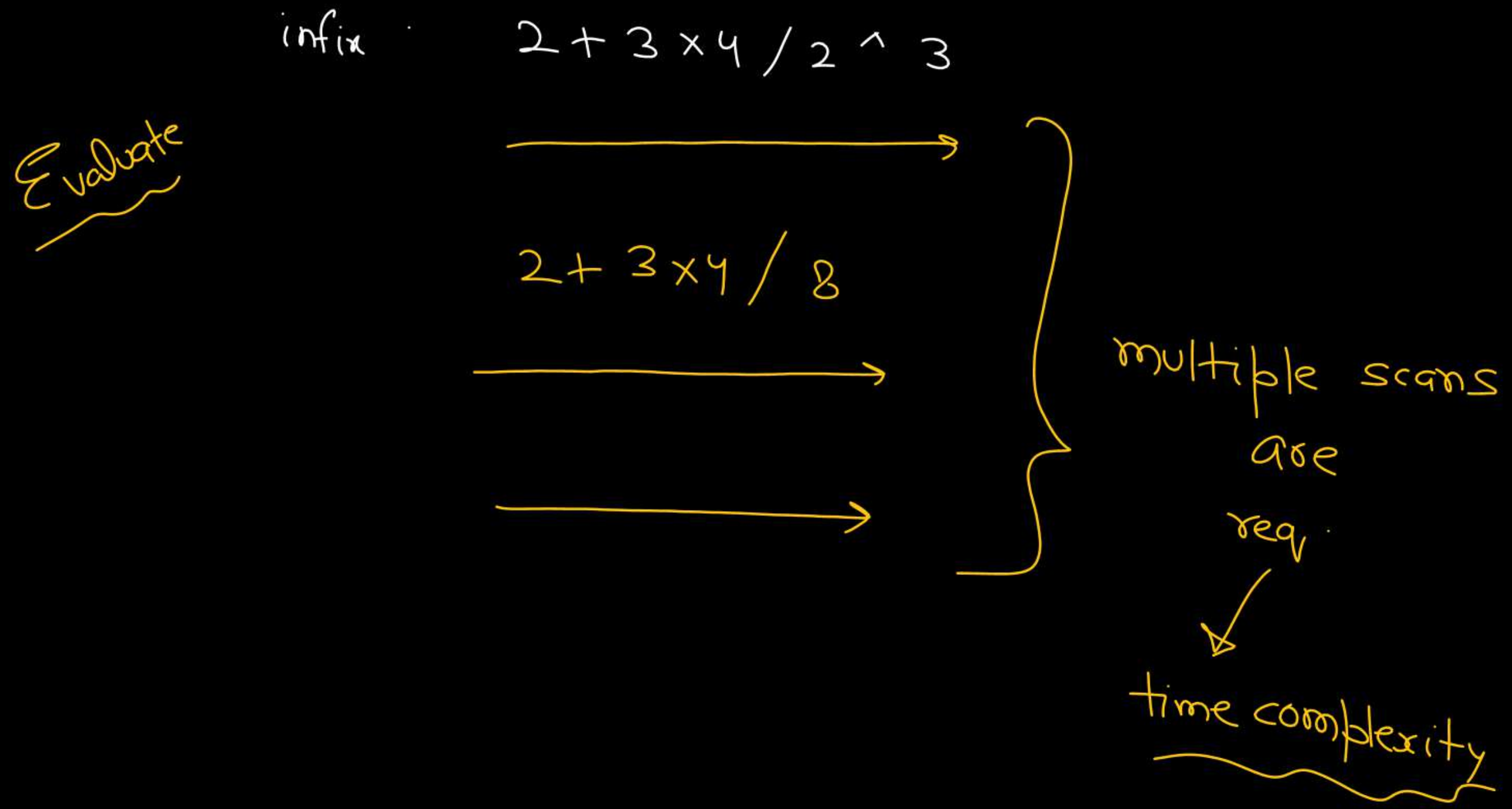
Prefix : operator is before operands

+ 2 3

Postfix : operator is after operands

2 3 +

why postfix ?


infix : 2 + 3 x 4 / 2 ^ 3
Evaluate
2 + 3 x 4 / 8
multiple scans are req.
time complexity

infix : 2 + 3 x 4 / 2 ^ 3

08 : 30

Evaluate

(i) Postfix

(ii) Postfix Evaluate


THANK YOU GW
SOLDIERS !