Stack And Queue

- Data Structure
  - Linear Data Structure

Stack

- LIFO (Last In First Out)

Real time application

- Recursion → Stack DS to Store the function call
- Undo Operation → Ctrl + Z
- Parenthesis Matching
  - ( )

ADT → Abstract Data Type

1) Push → Add element
2) Pop → Delete element

LIFO

Stack → Implementation

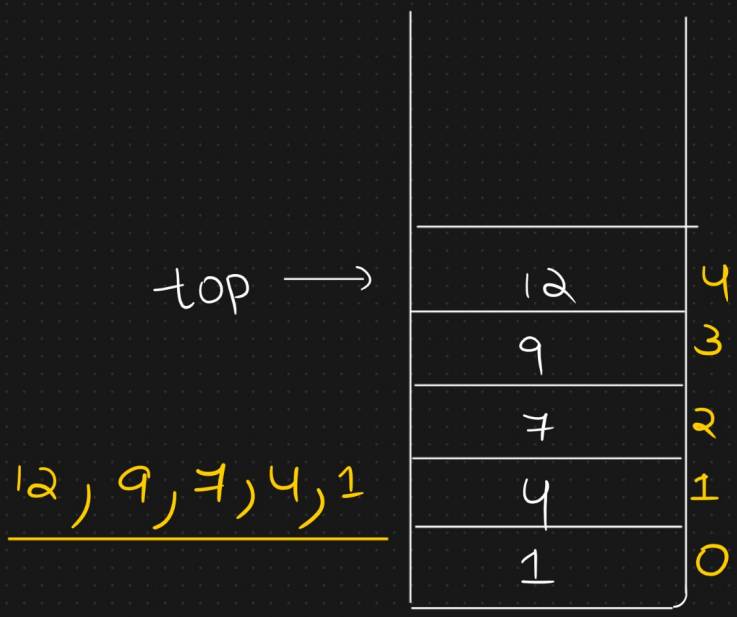
Array

Linked List

Push

time complexity ⇒ O(1)

Space complexity = O(1)



Stack

top = -1

```
push(arr, x):
  if(top == n) {
    print(Stack
          Overflow);
  }
  else:
    top = top+1
    arr(top) = x
```

Pop (Deletion)

time complexity → O(1)

```
pop():
  if(top == -1) {
    print("Stack underflow");
  }
  else {
    x = arr(top)
```

(Stack is empty)

x = 12

space complexity → O(1)

top = top - 1;

}

return x;

Queue — Array, Linked list

↓

FIFO

(First In First Out)

↳ Web browser history

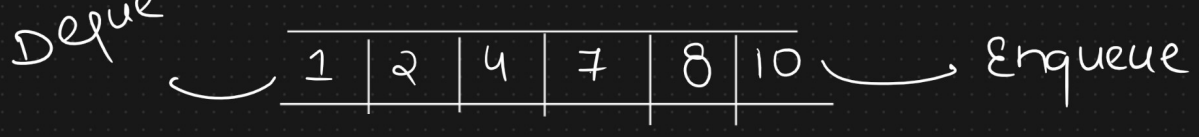
Abstract Data type

↳ Enqueue() ⇒ insertion in the queue

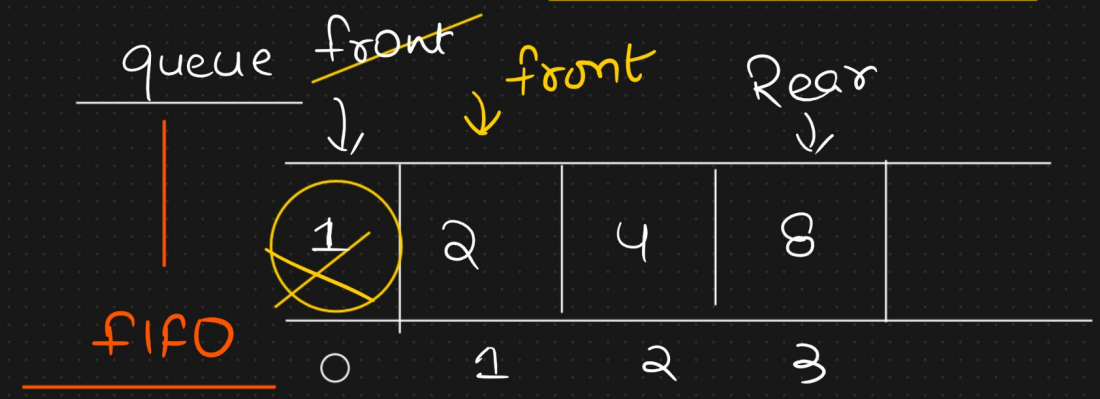
Deque()

⇓

front, Rear — deletion in the queue

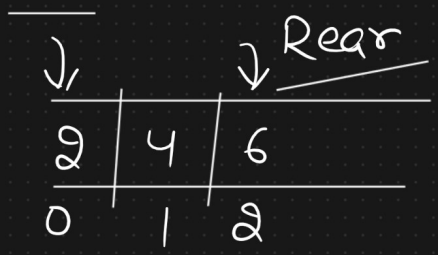


front = Rear = -1



History

Queue Implementation



enqueue(x):

if (rear == n):

Print(queue overflow)

time complexity ⇒ O(1)

else

if (front = rear = -1):

space complexity ⇒ O(1)

front += 1;

rear += 1;

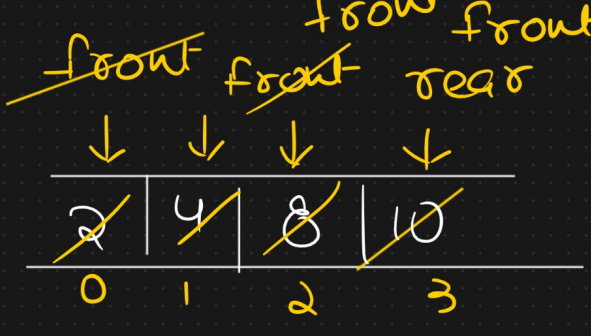
2, 4, 8, 10

x = 2, 4, 8, 10

else

rear += 1;

queue[rear] = x;



front = 0, rear = 3

time complexity = O(1)

space complexity = O(1)

FIFO

```
dequeue():
    if (front == -1) {
        Print(Queue underflow)
    }
    else {
        x = arr[front];
        if (front == rear)
            front = rear = -1;
        else:
            front = front + 1;
        return x;
```

Queue Empty

Last element to be deleted

peek()

↓

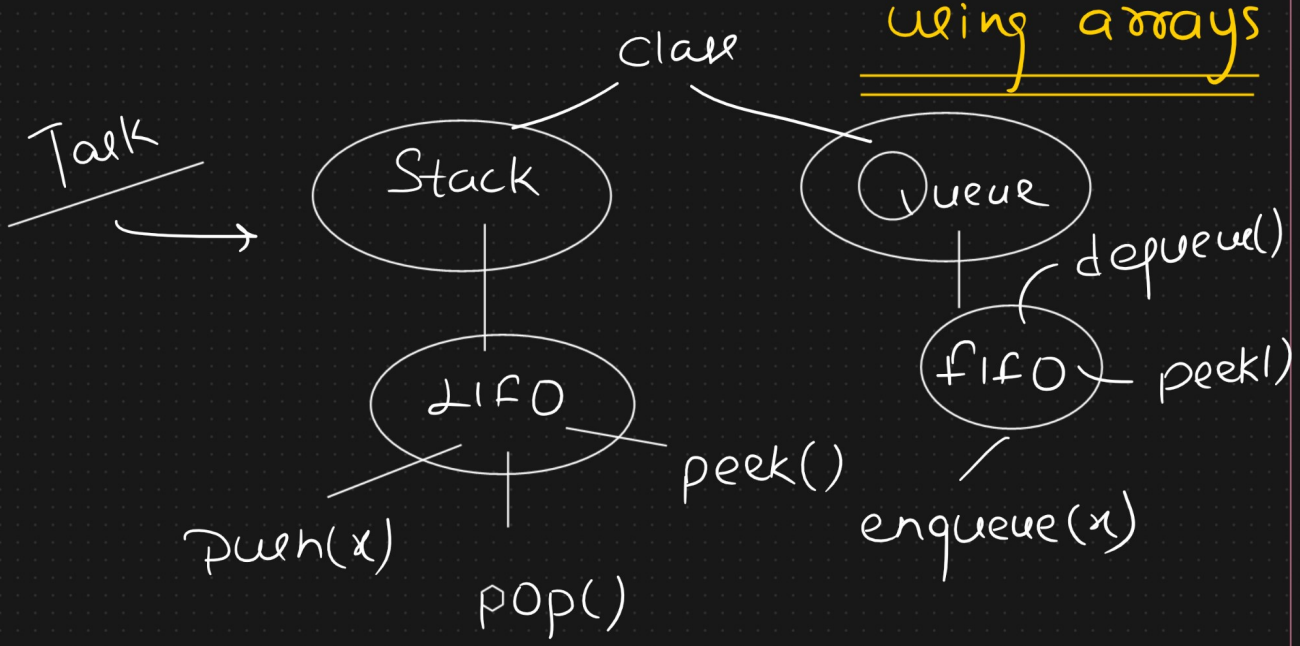
display of the topmost element of the stack

pop()

↓

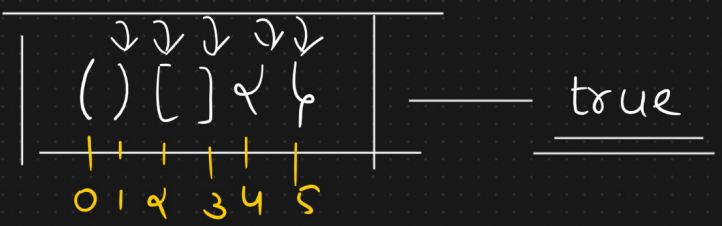
deletion of the topmost element of the stack

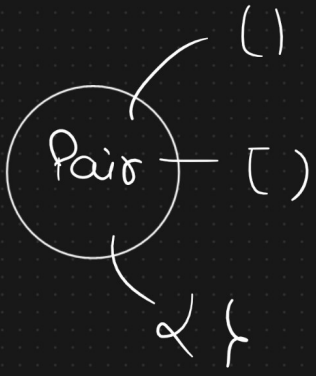
3	2	1	0
2	4	6	7

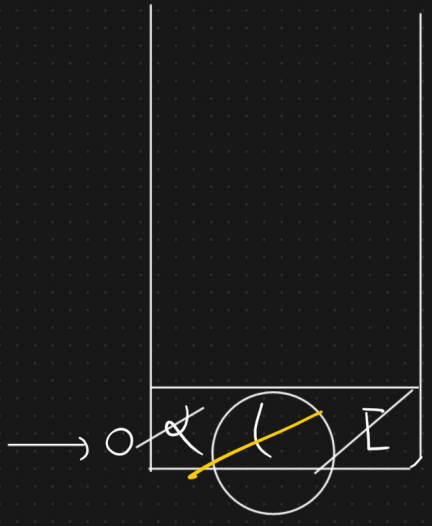


Interview Questions

1. Valid Parenthesis
2. Implement Stack using queue
3. Implement Queue using Stack



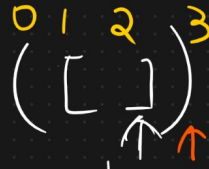




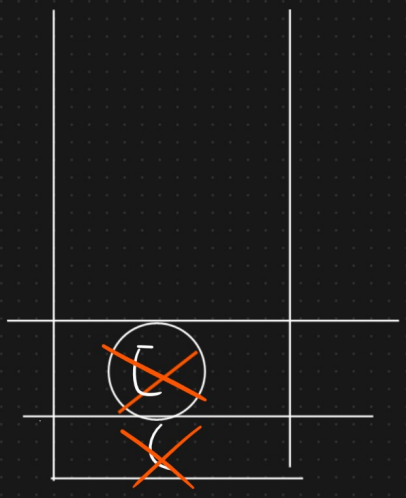
open bracket

Stack

Stack is empty

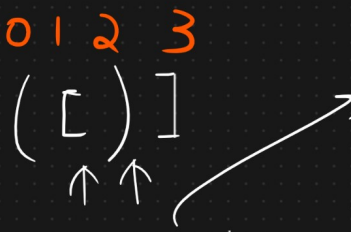


→ True

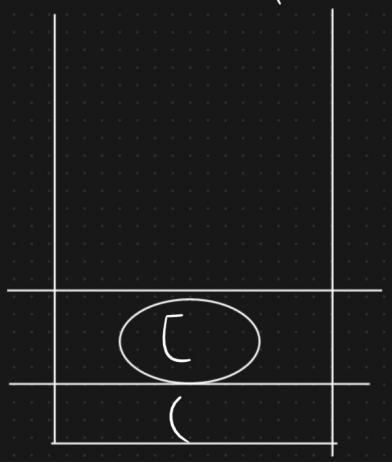


Stack → LIFO

↳ True



→ false



Stack

Hashing / Dictionary

Dict

↳ Mapping

Key → value

↳ (key, value)

O(1)

{ } : ( )

↳ [ ] : [ ]

{ } : { }

Dict[1] = [

Key

Stay = [1] [ ]

No

Dummy value

topEle = H

mappy[1]

↳ [

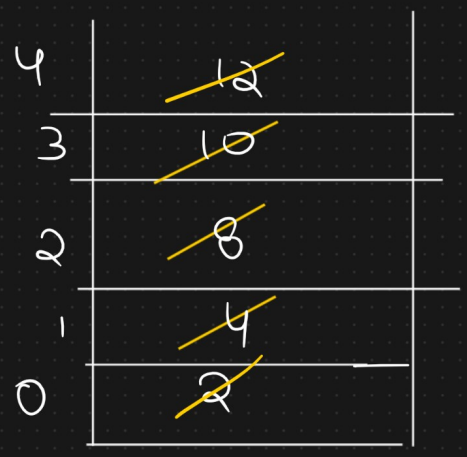
Stack

Given

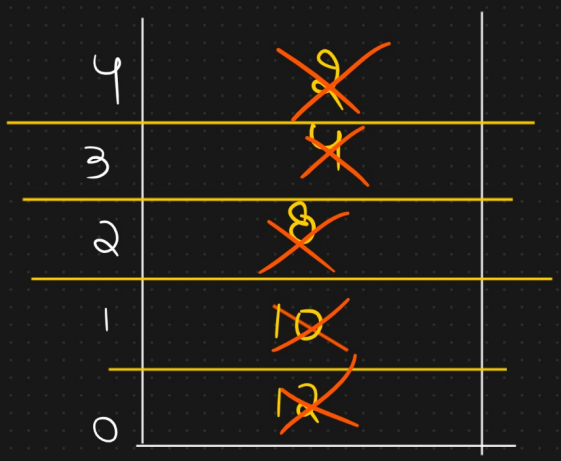
LIFO

Implement Queue

FIFO



Stack 1



Stack 2

2, 4, 8, 10, 12

queue

Push

↳ Stack 1.

append(x)

pop()

```
while(!stack1.isEmpty()){
    stack2.push(stack1.pop());
}
return stack2.pop();
```

```
peek()
return stack2.peek();
```

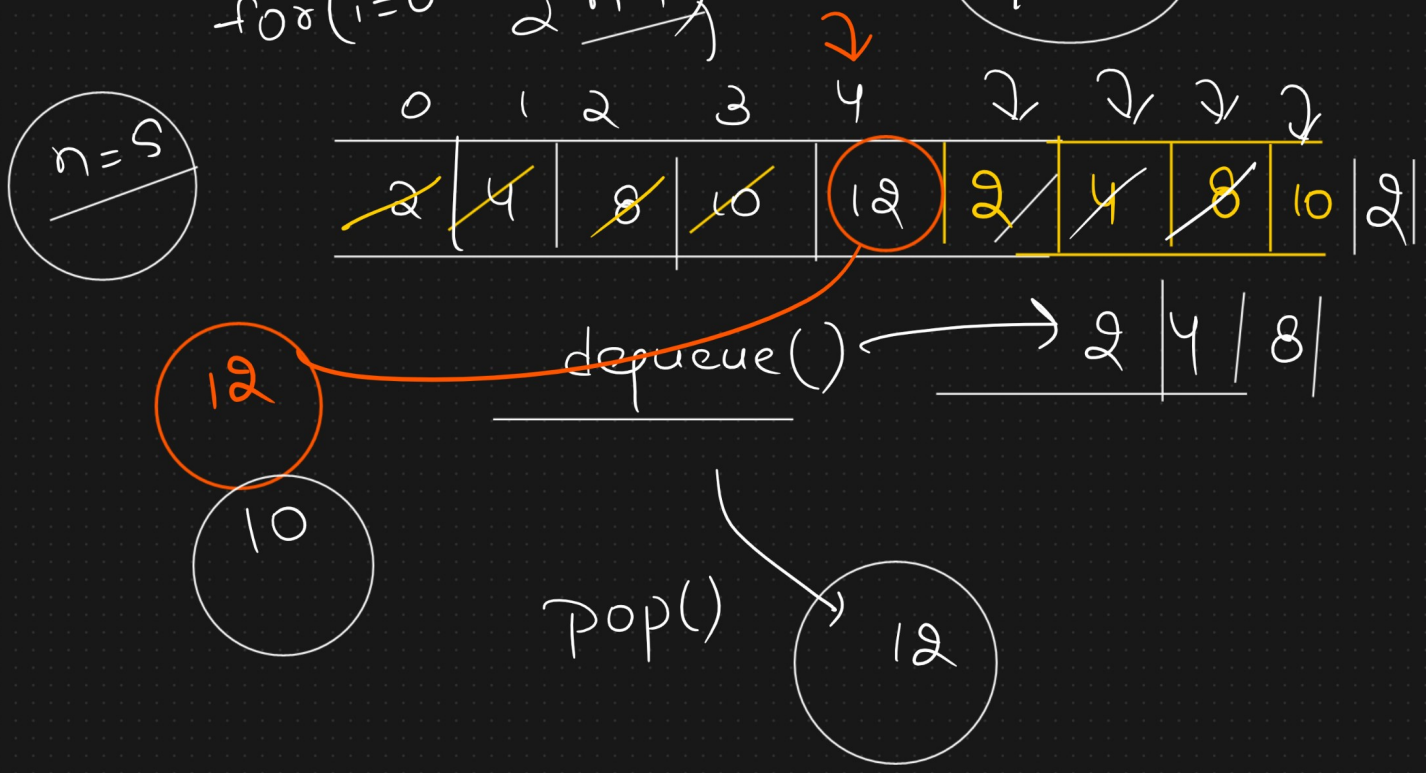
```
return (stack1.isEmpty() && stack2.isEmpty())
```

1 1 2

Implement Stack

→ via queue

for(i=0 to n-1)




```
for( i=0 to i<n-1)<
{
    q
        push( q.popleft())
    }
return q.popleft()
```

Task 2

https://leetcode.com/problems/implement-stack-using-queues/description/