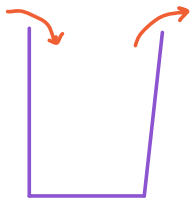
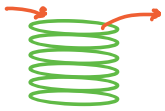
Stacks



LIFO
last in
first out.

1) Stack of plates



2) Glass of water/milk
3) Stack of books
4) Stack of chairs
5) Stack of CD's

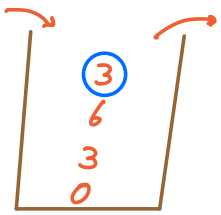
6) UNDO

$0 \xrightarrow{+3} 3 \xrightarrow{*2} 6 \xrightarrow{/2} \boxed{3} \xrightarrow{-1} 2$

operations

states

undo



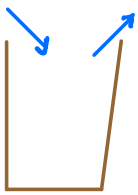
store state in stack

$x \xrightarrow{\%5} 3$

↳ {3, 8, 13, 18 . . . }

Operations

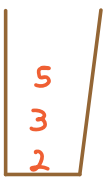
1) push(x) → insert x at top of stack.
2) pop() → remove top element from stack.
3) top()/peek() → get top element from stack.
4) isEmpty() → check if stack is empty.



TC = O(1) ∀ operations

Q → Implement stack using array.

✓ push(2)
✓ push(3)
✓ push(8)
✓ pop()
✓ peek() → 3 ✓
✓ push(5)
✓ isEmpty() → false ✓



A =

0	1	2	3	4	. . .
2	3	5			

stack is from index 0 to top.

initially top = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ 1 2

```
void push(int x) {
    top += 1;
    A[top] = x;
}
```

overflow → dynamic array

```
void pop() {
    if(!isEmpty())
        top -= 1;
}
```

underflow

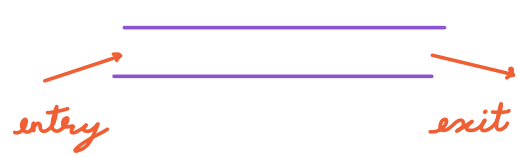
```
int peek() {
    if(!isEmpty())
        return A[top];
    else
        return -1;
}
```

Index out of bounds

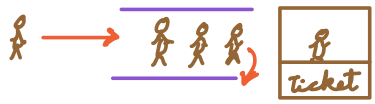
```
boolean isEmpty() {
    return (top == -1);
}
```

TC = O(1) ∀ operations

Queues



FIFO → first in first out

1) 

2) Toll tickets
3) Call center waiting
4) Straw / Pipe
5) Cab booking / Food order

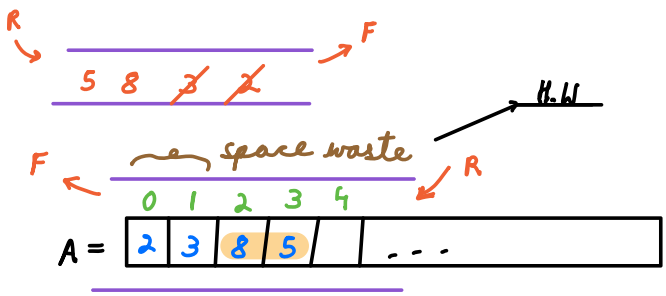
Operations

1) enqueue(x) → insert x from rear end.
2) dequeue() → remove front element from queue.
3) isEmpty() → checks if queue is empty.
4) front() → get front element.
5) rear() → get rear element.

TC = O(1) ∀ operations

Q → Implement queue using array.

✓ enqueue(2)
✓ enqueue(3)
✓ enqueue(8)
✓ dequeue()
✓ front() → 3 ✓
✓ rear() → 8 ✓
✓ dequeue()



Queue is from index f to r

✓ isEmpty() → false ✓

✓ enqueue(5)

$f = \cancel{0} \cancel{1} 2$ $r = \cancel{-1} \cancel{0} \cancel{1} \cancel{2}\ 3$

```
void enqueue(int x){
    r += 1;
    A[r] = x;
}
```

```
void dequeue(){
    if(!isEmpty())
        f += 1;
}
```

```
boolean isEmpty(){
    return f > r;
}
```

overflow → dynamic array

underflow

TC = O(1) ∀ operations

```
int front(){
    if(!isEmpty())
        return A[f];
    else
        return -1;
}
```

```
int rear(){
    if(!isEmpty())
        return A[r];
    else
        return -1;
}
```

Q → Check whether the given sequence of parenthesis is valid.

AWS
Sutherland
Walmart

{, }, (,), [,]

s = "( ) [{ } ( )]" → valid

s = "( ) [{] } (" → invalid

↳ x

s = "( ) [{ }] [ ]" → Ans = true

s = "({) }" → Ans = false

s = "({ })) (" → Ans = false

s = "({ } [" → Ans = false

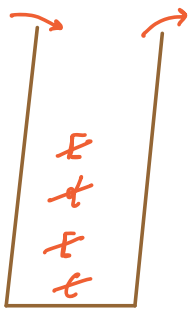
stack

∀ closing brackets check if the last opening bracket matches.

s = "( ) [{ }] [ ]"

( ) ✓ [ ] ✓

{ } ✓ [ ] ✓



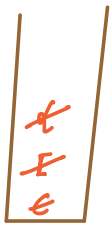
Ans = true

s = "( ) [{ }) [ ]"

( ) ✓

{ } ✓

[ ) ✗ Ans = false



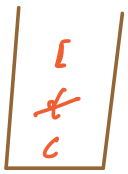
TC = O(N)

SC = O(N)

s = " ({ } ["

{ } ✓

Ans = false



stack should be empty in the end.

s = " { (}) "

(} ✗

Ans = false