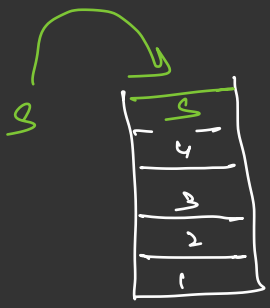
STACK

↳ Data Structure

→ Linear DS

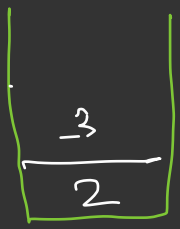
→ Last In First Out (LIFO)



eg → Call / Function Stack in memory

→ Undo / redo

→ Forward / Back in browser



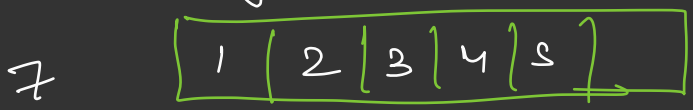
Functions of stack

O(1)

1. Push → add at top
2. Pop → remove from top
3. Peek → return top
4. Size → returns size of stack
5. IsEmpty → returns boolean depending on size

→ push(4)
→ push(3)
→ push(5)
→ pop()

ArrayList

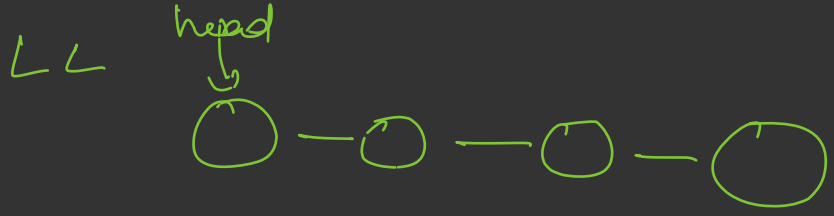


Add First	O(n)	remove First	O(n)
Add Last	O(1)	remove Last	O(1)
get	O(1)		

```
class Stack {

    ArrayList        list;

    push (int item) {
    list.add (item);
}
```

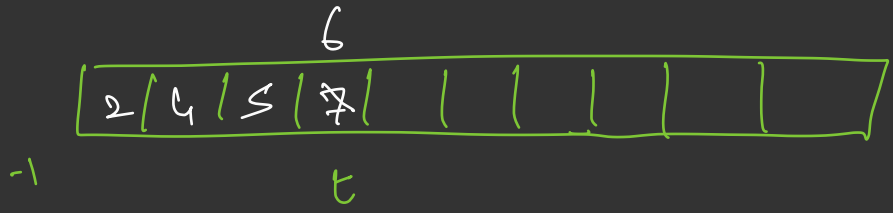


addFirst O(1)

addLast O(n)
with tail O(1)

removeFirst O(1)

removeLast O(n)



push(2)
push(4)
push(5)
push(7)

pop()
push(6)
peek() → return arr[t];

Stack {

size

}

1. Push → addFirst
2. Pop → removeFirst
3. Peek → getFirst → getNode(0)
4. size
5. IsEmpty → size == 0


```
push(val) {
    arr[++top] = val;
}

pop() {
    top--;
    return arr[top+1];
}

peek() {
    return arr[top];
}
```

Ques Duplicate bracket

((a + b) + c) // false

↑

→ (((a+b)) + c) // true

↑

(((a + b)) + c)

i

Ques Balanced brackets

Balanced {a + [b + (c+d)]} ← {[](())}

↑

Not balanced

{a +]

{[)}

{a [b+] }}

{{

{a +]}

{[]}}

{[()]}

i

(1) Not compliment

(2) Extra opening

(3) Extra closing

Ques Next greater element on right

	3	5	11	7	6	5	1	4	8
→	5	11	-1	8	8	8	4	8	-1

(1)

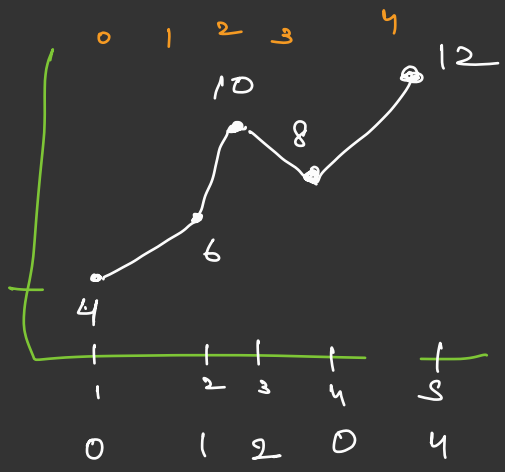
1 — n
2 → n-1
3 — n-2
⋮
n — 1

⇒ $O(n^2)$

0	1	2	3	4	5	6	7	8
3	5	11	7	6	5	1	4	8
			8	8	8		8	↑


```
while (arr[st.peek()] < arr[i])
{   int idx = st.pop();
    res[idx] = arr[i];
}
```

Ques Stack span



(1) Next greater from left

(2) currIdx - ngeIdx - 1

Ques Celebrity party

	0	1	2	3
0	✓	✓	✓	✗
1	✗	✓	✓	✗
2	✗	✗	✓	✗
3	✗	✗	✓	✓

(1) $O(n^2)$

(2)

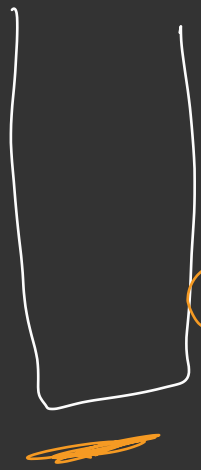
0 1 2 3

r

0

$(a + (b + c) + d) + e)$

$(a + (b + c) + d) + e$



$(a + (b + c) + d) + e$