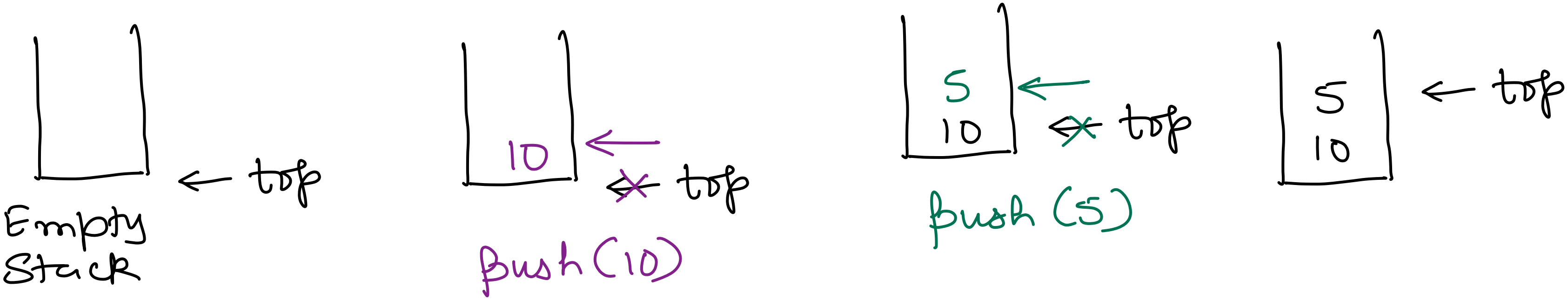
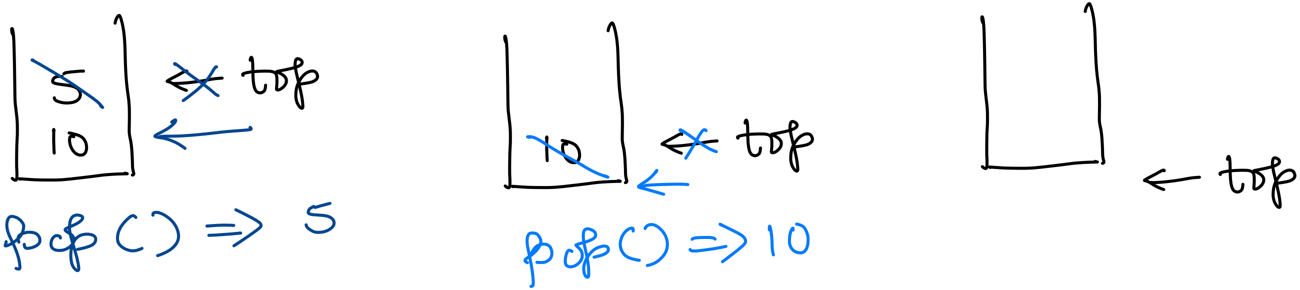
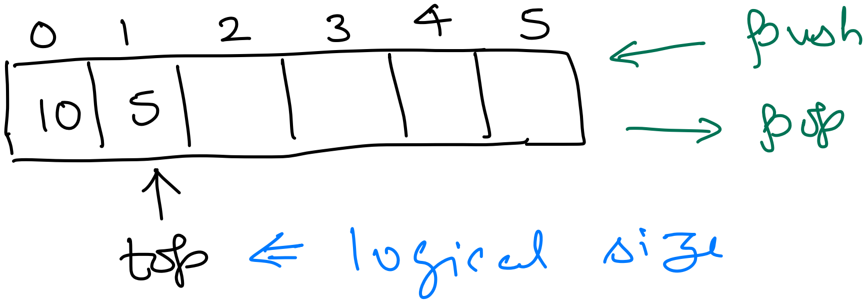
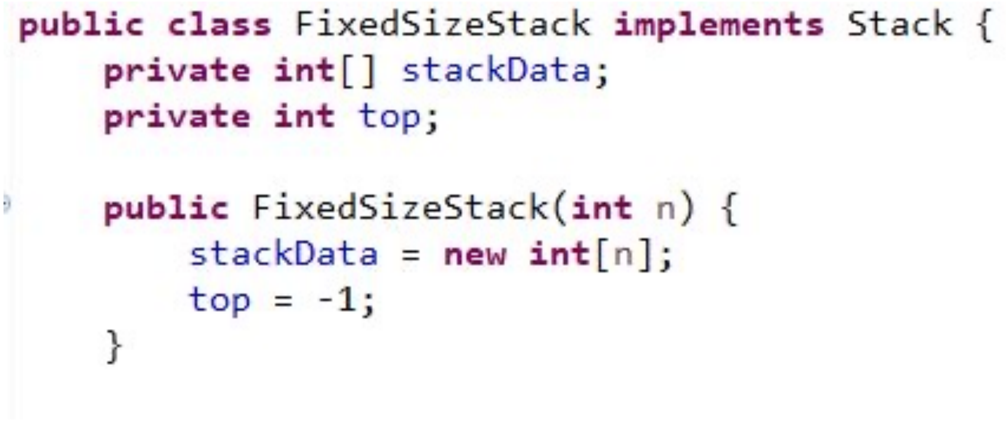
Stack operation





Stack using Array





```
public class FixedSizeStack implements Stack {
    private int[] stackData;
    private int top;

    public FixedSizeStack(int n) {
        stackData = new int[n];
        top = -1;
    }
```

In assignment, we will resize array here.

Push(element)

- If stack is full then stop. → if (isFull()) ~~return;~~ throw exception
- Make space at top for new element. → ++top;
- Store new element and make it topmost element. → stackData[top] = element;

Pop()

- If stack is empty then stop. → if (isEmpty()) ~~return;~~ throw exception
- Set topmost element as result. → result = stackData[top];
- Remove topmost element and make element below top, the topmost element. --top;
- Return the result. → return result;

IsEmpty()

- If no element stored at top then return true. → if (top == -1) return true;

Else return false → return false;

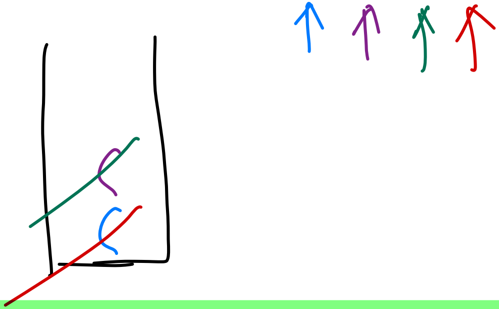
IsFull()

- If no space left for new element to be stored then return true. → if (top == stackData.length - 1) return true;

Else return false. → return false;

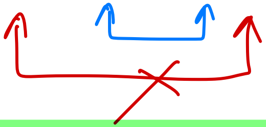
Check if string of paranthesis is balanced.

(( ))



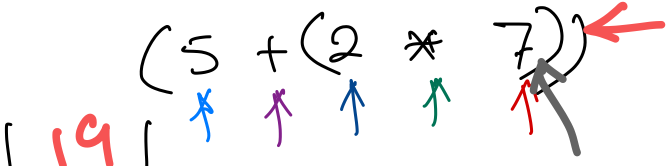
Check if string of paranthesis is balanced using ({ [

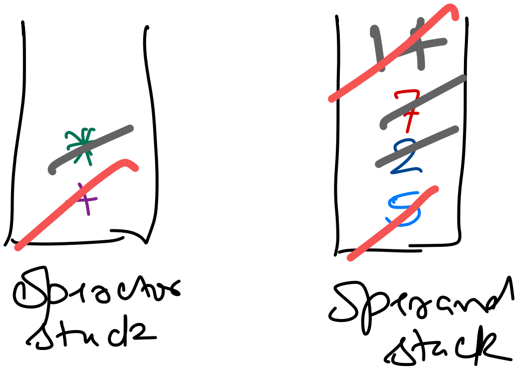
({ [ ]) }



Evaluate an expression that is fully parenthesised.

(5 + (2 * 7))





$2 * 7 = 14$

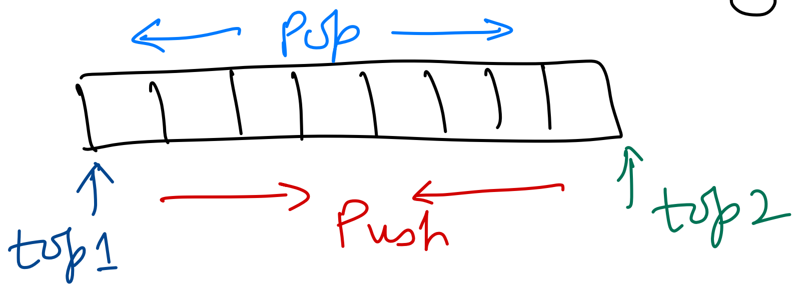
$5 + 14 = 19$

Min Stack

Max Stack

Implement m stacks in a single array.

Implement two stacks in a single array.



Applications of Stack

- → O.S. ⇒ function calls.
- → Recursive to iterative conversion.
- → Other algorithms.

Analysis of Algorithms

- Time complexity
- Space complexity.

Big O notation

Find sum of 3 numbers

1. Read 3 numbers (no1, no2, no3). → ①
2. Sum = no1 + no2 + no3 → ①
3. Print Sum. → ①
4. Stop.

Total = 3

Constant time complexity ⇒ algorithm takes time indipendent of I/P size.

If its a constant then time complexity = $\boxed{O(1)}$

Find sum of N numbers

```
1. Read N.                                  → 1
2. Create memory to store N numbers (nums). → 1
3. for i = 0 to (n-1) ⇒ loop, how many times? n times
        3.1 Read a number (nums[i]).        → 1

4. Sum = 0                                  → 1
5. for i = 0 to (n-1) ⇒ loop, how many times? n times
        5.1 Sum = Sum + nums[i]             → 1

6. Print Sum                                → 1
7. Stop.
```

Total = $4 + 2n \Rightarrow O(n)$

1 Ignore constants.

linear time complexity

for i = 0 to (n-1)

values i will take will be in range $[0, n-1]$ ⇒ Number of elements in a closed range $[L, R] = (R - L + 1)$

Find time complexity of following

```
for i = 0 to (n-2)
    for j = (i+1) to (n-1)
        ... do something ...
```

Sum of first N natural numbers

$$= \frac{N(N+1)}{2}$$

i ↓		j		How many times inner loop runs
0	→	1 2 ... $(n-1)$	→	$(n-1)$
1	→	2 3 ... $(n-1)$	→	$(n-2)$
2	→	3 4 ... $(n-1)$	→	$(n-3)$
⋮		⋮		⋮
$(n-3)$	→	$(n-2)$ $(n-1)$	→	2
$(n-2)$	→	$(n-1)$	→	1

Sum

Total times inner loop runs $= \frac{(n-1)(n-1+1)}{2}$

$$\frac{(n-1)(n)}{2} = \frac{1}{2}(n^2 - n)$$

① Ignore constants.

$$n^2 - n$$

② Pick term with highest power of n

$\Rightarrow O(\underline{n^2)}$ → quadratic time complexity.

$n^2 - n \approx n^2$ for very large values of n.

Find time complexity of following

Exercise

```
for i = 0 to (n/2)
    for j = (i+1) to (n-1)
        for k = 0 to (m-1)
            ... do something ...
```

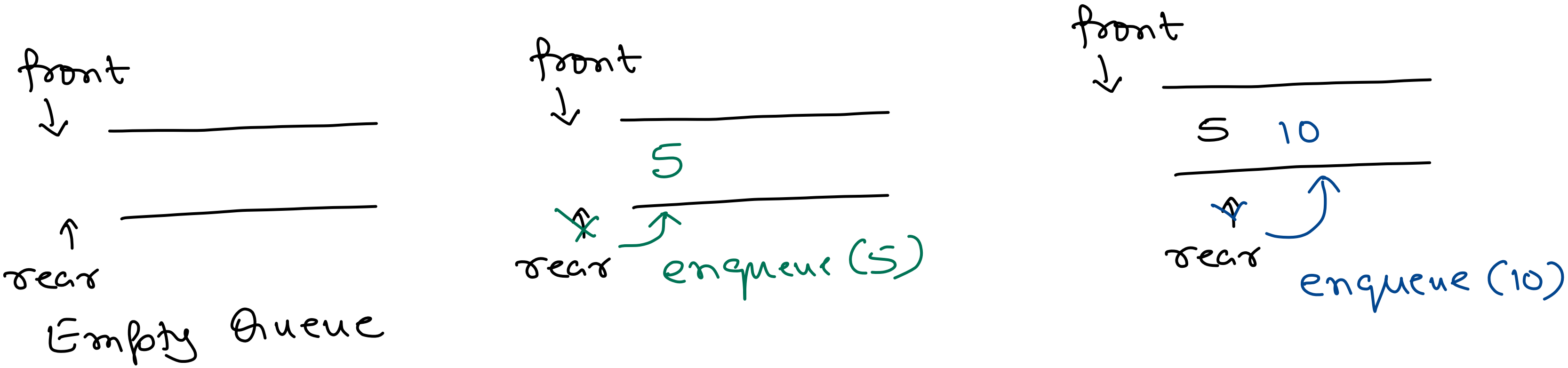
Queue

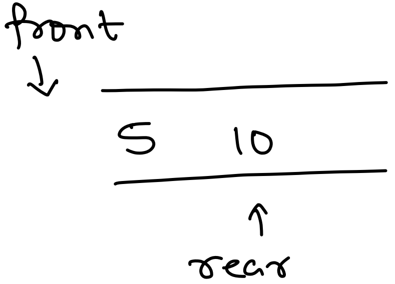
- Queue is a linear data structure.

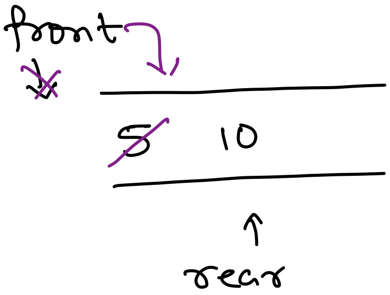
- Queue is a container of objects.

Queue operations

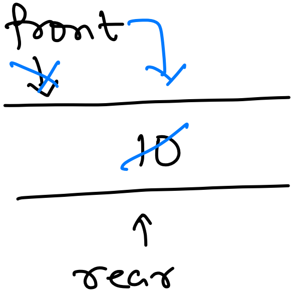
- FIFO – First In First Out
- Elements are added and removed according to FIFO principle.
- Addition of elements are performed at “**rear**” of queue.
- Elements are removed from “**front**” of queue.



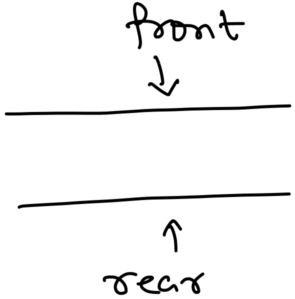




dequeue() => 5



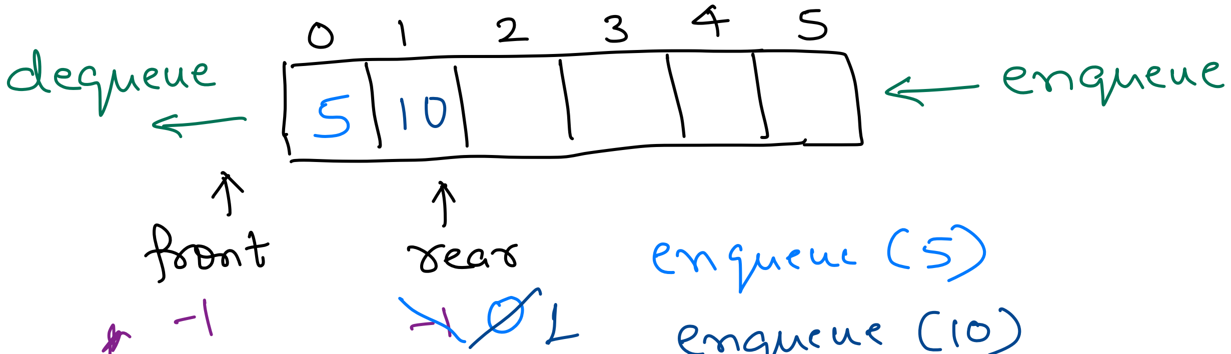
dequeue => 10



Define queue as Abstract Data Type

```
public interface Queue {

        void enqueue (type element);

        type dequeue ();

        boolean isEmpty();

        boolean isFull();
}
```

Queue using Array



enqueue (5)

enqueue (10)

```
class Queue implements Queue {
    type [] queueData;
    int front;   int rear;

    public Queue( int n) {
        queueData = new type [n];
        front = -1;  rear = -1;
```

Enqueue(element)

- If queue is full then stop. → if (isFull()) throw Exception.
- Make space at rear for new element. → ++rear;
- Store new element and make it the rear element. → queueData[rear] = element;

Dequeue()

- If queue is empty then stop. → if (isEmpty()) throw exception
- Move the front towards rear. → ++front;
- Remove and return the front element as result.
  - → Remove element as result, → result = queueData[front]
  - → Return result → return result;

IsEmpty()

- If no elements stored in queue then return true. → if (front == rear) return true;

Else return false. → return false;

IsFull()

- If no space left for new element to be stored then return true. → if (rear == queueData.length -1) return true;

Else return false. → return false;

Assignment: Implement generic queue using array.