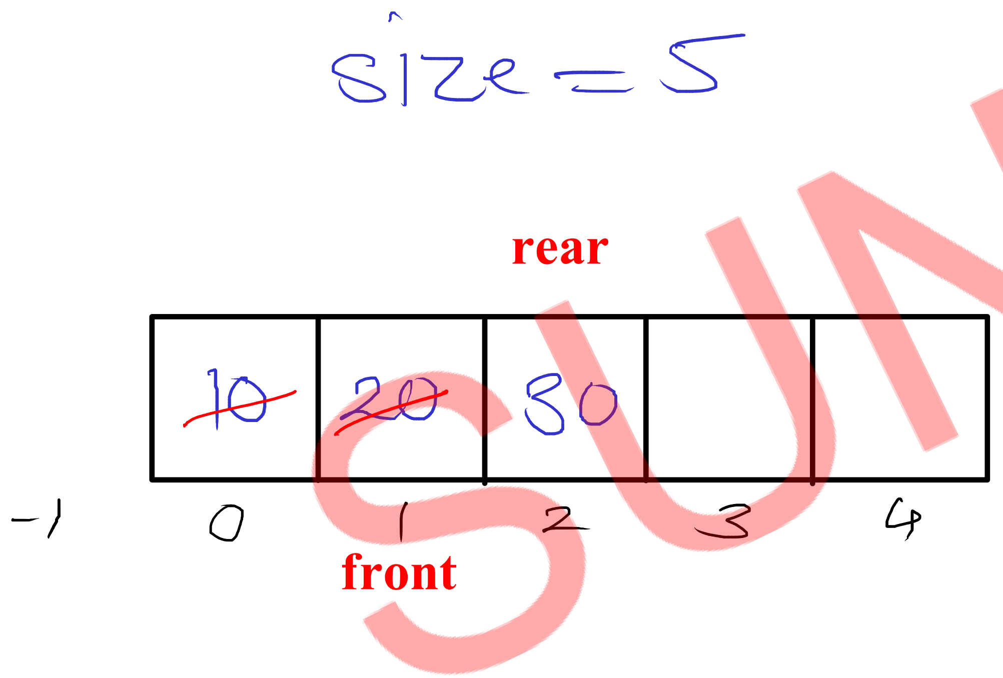
	Best case	Avg case	Worst case	Auxillary space
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

Linear Queue

- queue is a linear data structure in which data is stored sequentially.
- every queue has two ends
- data is inserted from one end of queue (rear)
- data is removed from another end of the queue (front)
- queues can be implemented using arrays
- queue works on the principle of "First In First Out" / "FIFO"



Operations:

1. Insert/Add/Enqueue/Push:

a. reposition rear (inc)
b. add data/value at rear index

2. Remove/Delete/Dequeue/Pop:

a. reposition front (inc)

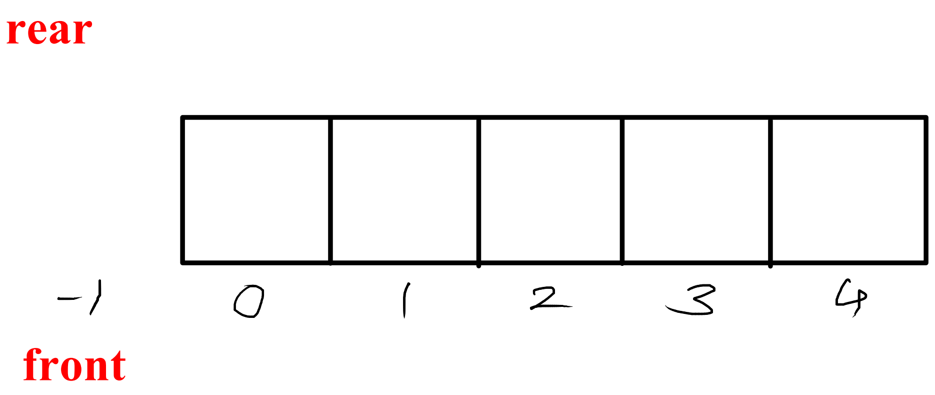
3. Peek

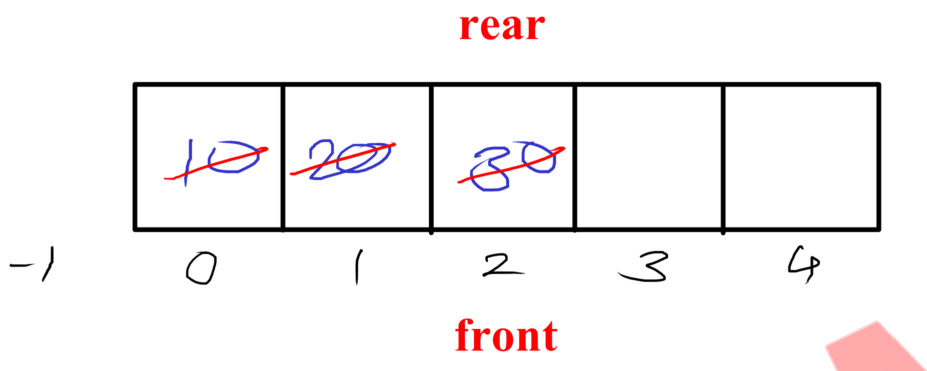
a. read/return data of front + 1 index

- All operations of queue are performed into O(1) time complexity.

Conditions

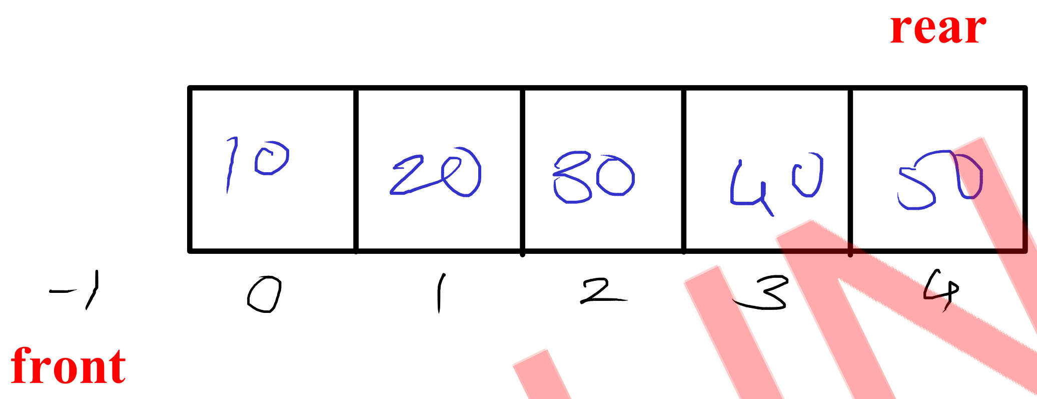
1. Empty



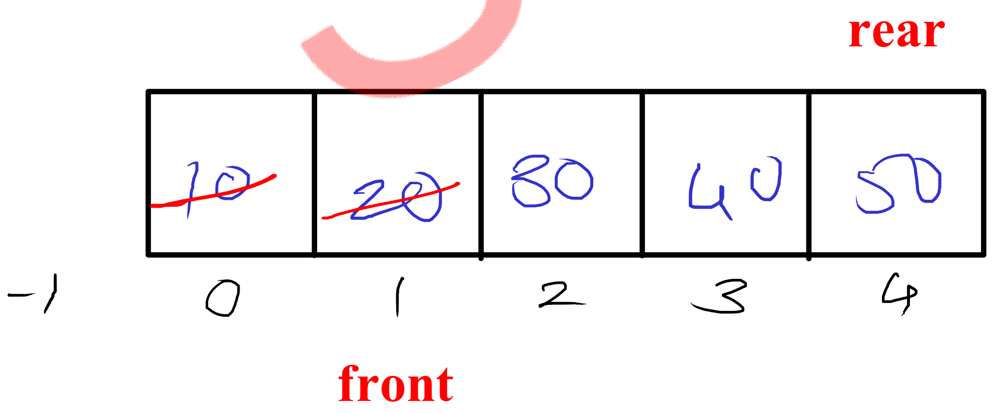


front == rear

2. Full



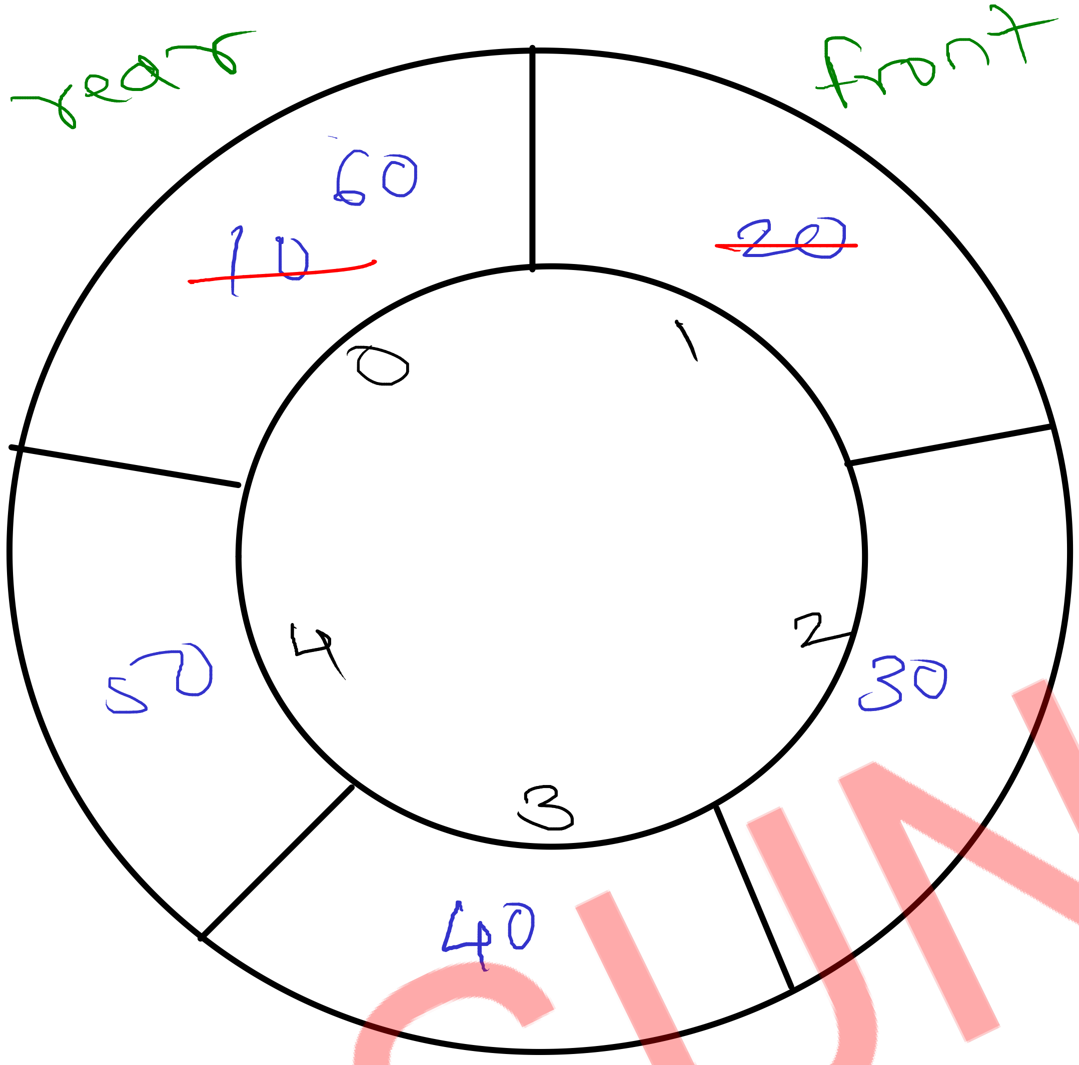
rear == size-1



- once rear is reached to last index of array and initial few locations are empty, still we are not able to use those location.
- This lead to poor memory utilization.

Circular Queue

size = 5



rear = (rear + 1) % size

front = (front + 1) % size

front = rear = -1

= (-1+1) % 5 = 0

= (0+1) % 5 = 1

= (1+1) % 5 = 2

= (2+1) % 5 = 3

= (3+1) % 5 = 4

= (4+1) % 5 = 0

- along with array, front & rear, also maintain count of elements

init -> count = 0

push -> count++

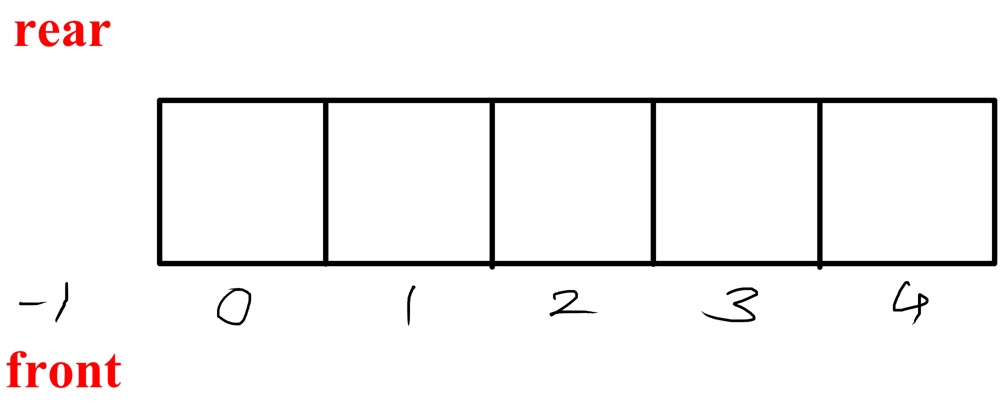
pop -> count--

empty -> count == 0

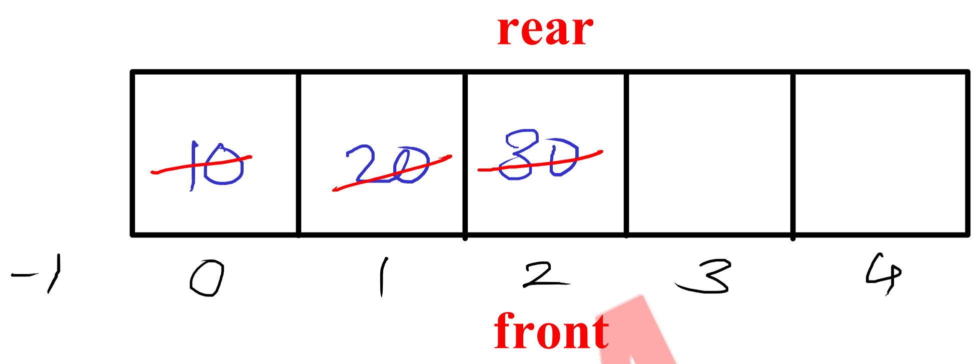
full -> count == size

Circular Queue - Empty and Full conditions

Empty

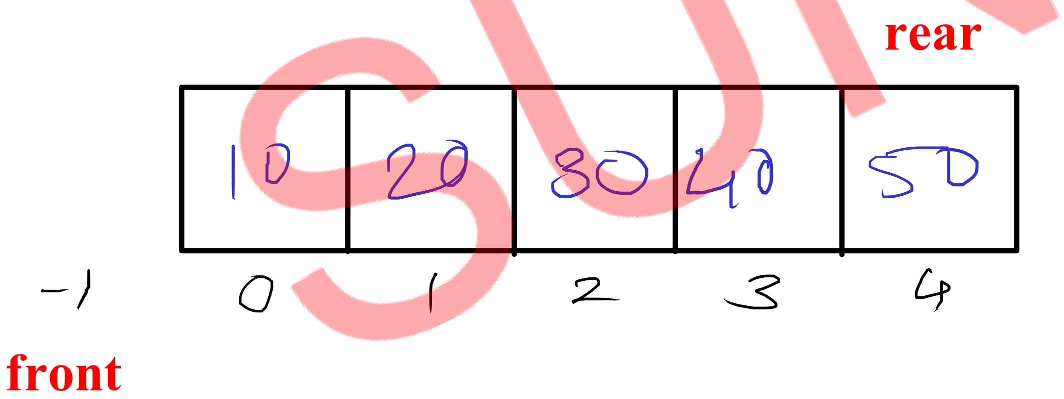


front == rear && rear == -1

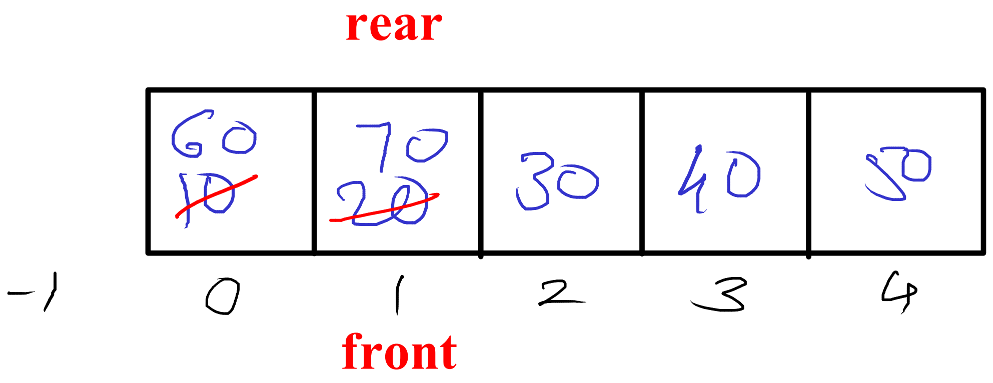


```
pop(){
        front = (front+1)%size;
        if(front == rear)
                front = rear = -1;
}
```

Full



front == -1 && rear == size - 1

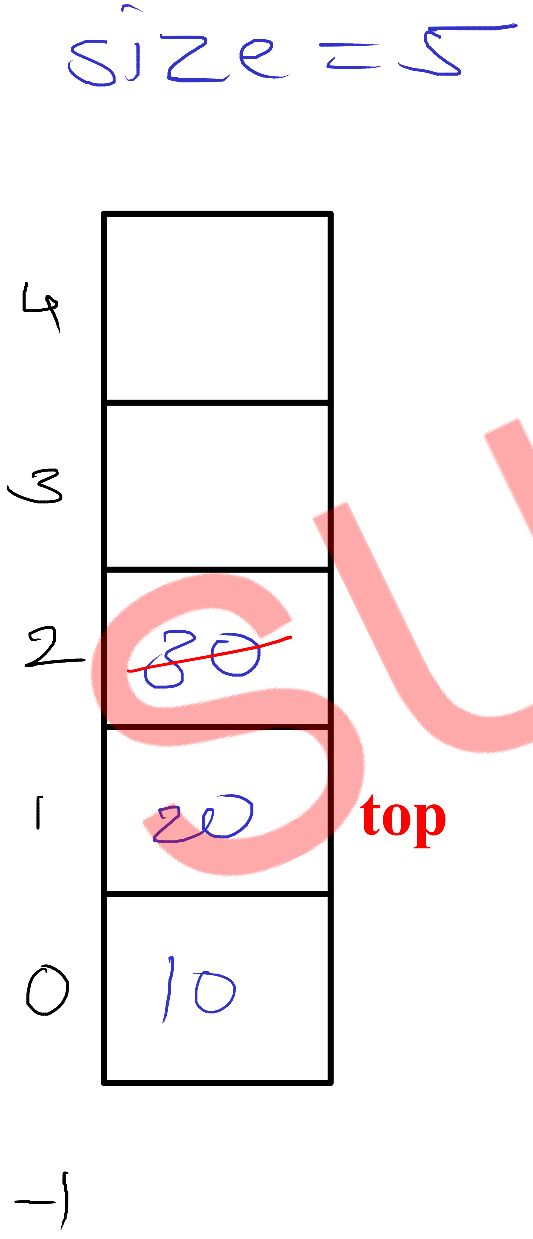


front == rear && rear != -1

(front == -1 && rear == size - 1) || (front == rear && rear != -1)

Stack

- stack is a linear data structure in which data is stored sequentially
- stack has only one end and it is known as "top"
- data is inserted or removed from top end only
- stack works on the principle of "Last In First Out" / "LIFO"
- top always points to last inserted data



Operations:

1. Insert/add/push:

a. reposition top (inc)
b. add value/data at top index

2. Remove/delete/pop:

a. reposition top (dec)

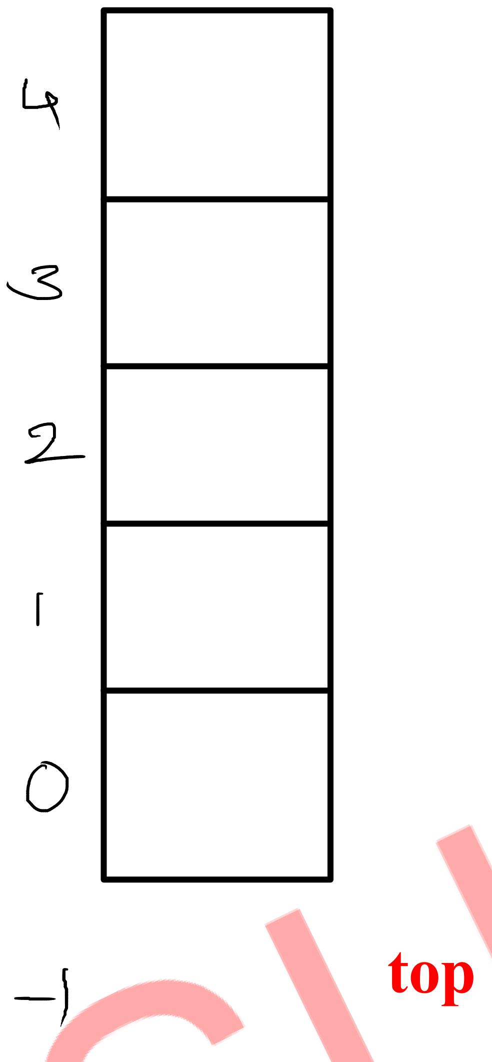
3. Peek:

a. read/return value of top index

- all operations of stack are performed in O(1) time complexity.

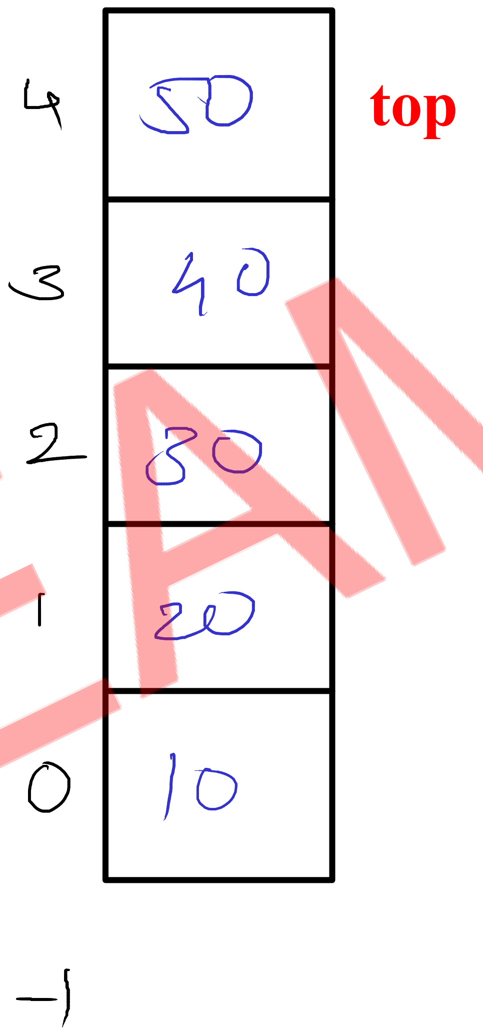
Conditions

Empty



top == -1

Full



top == size-1

Ascending stack - fill stack from lower address to higher address
(0 → size-1)

Descending stack - fill stack from higher address to lower address
(size-1 → 0)

Stack & Queue Time complexity
(Array implementation)

	stack	linear queue	circular queue
push	O(1)	O(1)	O(1)
pop	O(1)	O(1)	O(1)
peek	O(1)	O(1)	O(1)