Stacks ⇒ LIFO

⇓

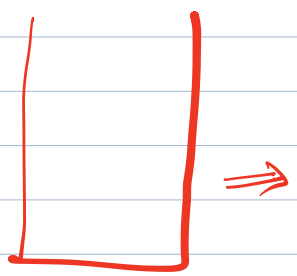
last In first Out

→ Insertion / Extraction happens from one end only

Use Cases

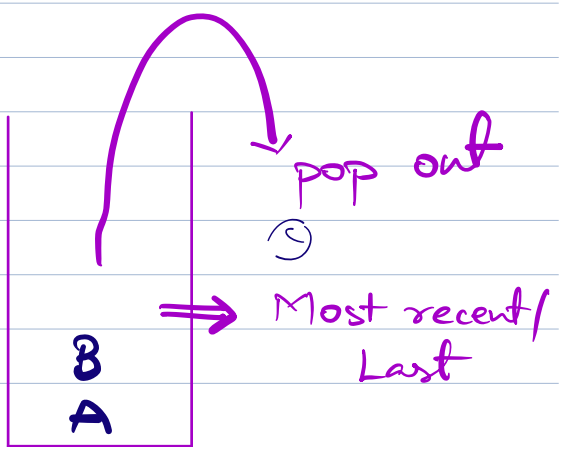
1) Function Stack / Call Stack

```
main () {
    print ( fibonacci (10))
}
```



2) Undo

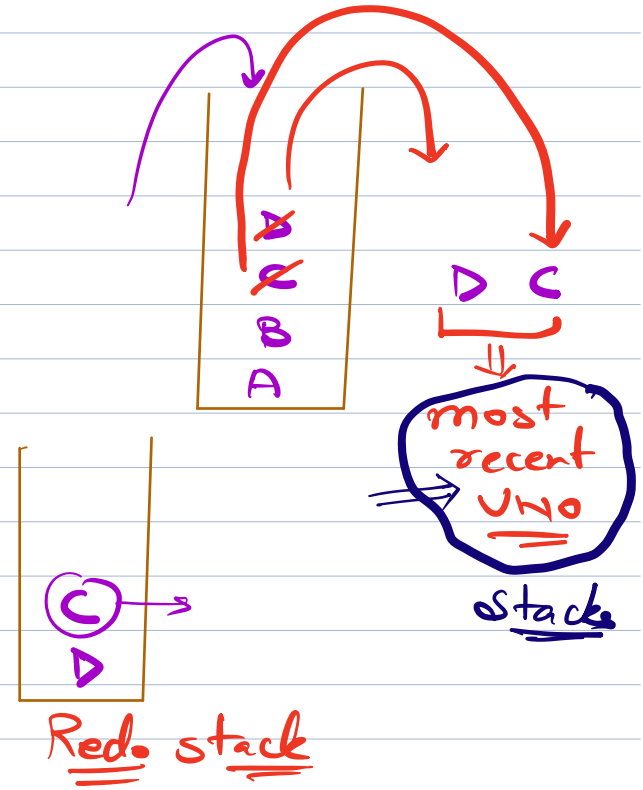
1 A, 2 B, 3 C



3) Redo

1 A, 2 B 3 C Undo 4 D Undo

Redo



Operations

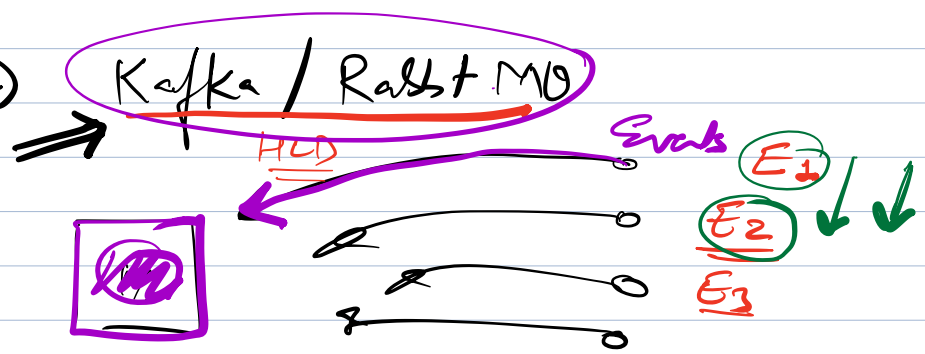
- push(x) → insertion
- pop() ⇒ Removes the top ele of stack
- size()
- isEmpty() ⇒ Returns true when the stack is Empty.
- top() / peek() ⇒ Returns the top of stack w/o deleting

2) Queues ⇒ FIFO

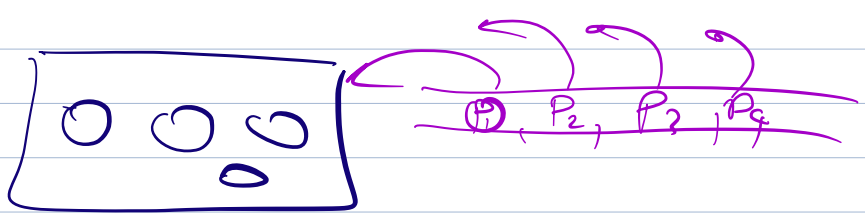
⇓ First In First Out

Application

1)



2) Scheduling Algo

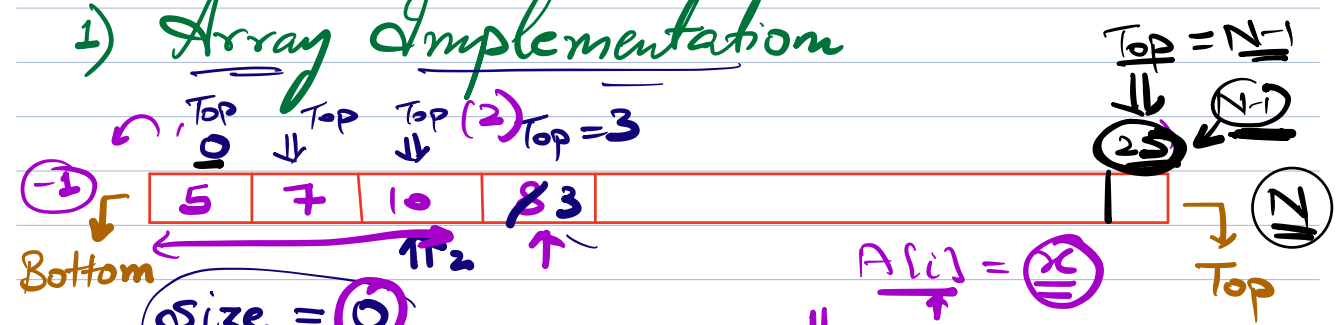


Implementation

1) Stack

- 1) Array
- 2) Dynamic Array
- 3) Linked list

1) Array Implementation



size = 0

top = -1

0

index of most recent inserted element

A[i] = x

top++

A[top] = value

size++

top--

size--

size = top+1

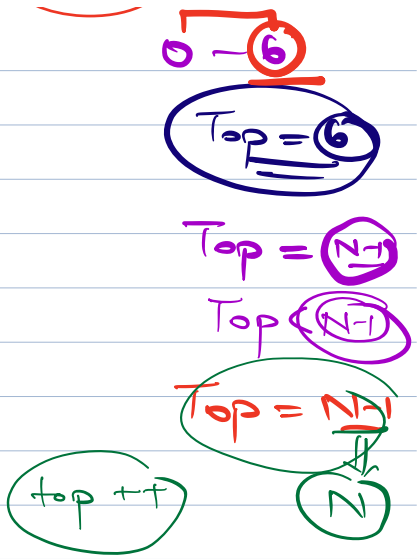
size	top
0	-1
1	0
2	1
3	2
4	3
3	2

N=7

Code

```
int arr [ maxPossibleSize]
int top = -1
```

```
void push (x) {
    if ( top < (N-1) ) {
        top++;
        A[top] = x;
    }
}
```



T. C. = O(1)

Integer

```
int pop() {
    if (top >= 0) {
        temp = A[top]
        top--;
        return temp;
    }
    return NULL;
}
```

⇒ O(1)

3)
```
bool isEmpty() {
    if ( top == -1) {
        return True;
    }
    return false;
```

⇒ O(1)

```
}
```

4)
```
int peek() {
    if (top >= 0) {
        return A[top];
    }
    return NULL;
}
```

⇒ Dynamic Array / ArrayList / list / Vector

↳ No restriction on size.

2) Linked list

⇓

```
class Node {
    int data;
    Node next;

    Node (int a){
        this.data = a;
        this.next = NULL;
```

Python

```
class Node:
    def __init__
        (self, a):
        self.data=a
        self.next
            =Null
```

}

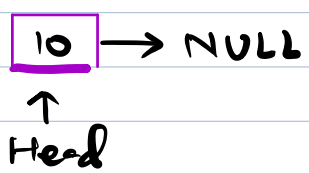
}

push(a) ⇒ Insert at head
pop() ⇒ Delete the head
] ⇒ O(1)

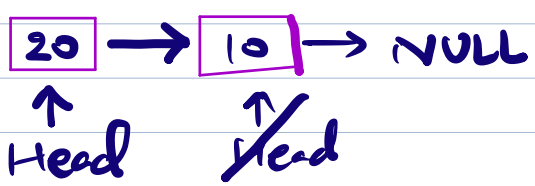
Dry Run

Head = NULL // Initialize

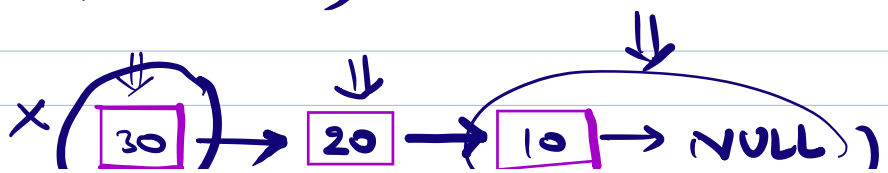
1) push(10)



2) Push(20)

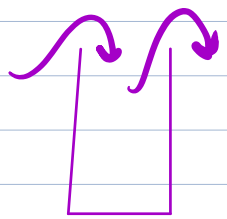


3) Push(30)



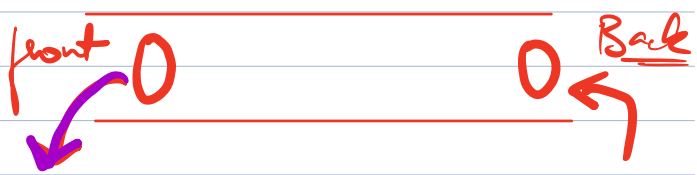
4) Pop()

```
temp = head
head = head.next
free (temp);
```



Queue Implementation

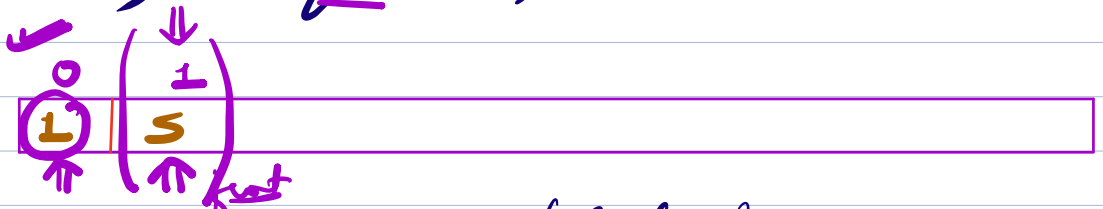
1) Array
2) linked list



Operations

1) Enqueue (x) ⇒ Insert
2) Dequeue () ⇒ Delete

] O(1)



1) front = -1 (Index from where you

2) tail = -1 (Index where you will remove) (Index where you last inserted)

Enqueue (1)

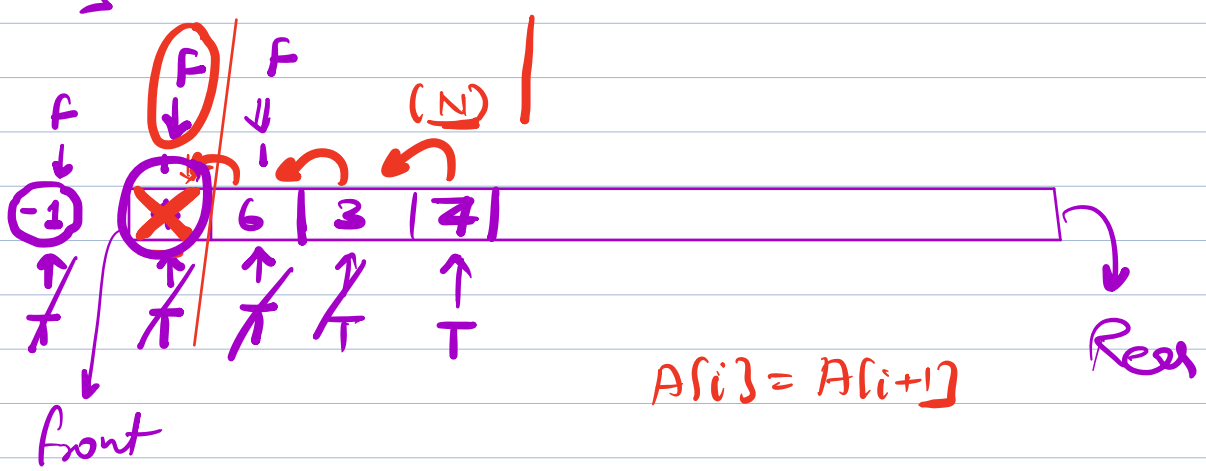
Enqueue (5)

```
tail++
A[tail] = 1;
if (front == -1){
    front++;
}
```

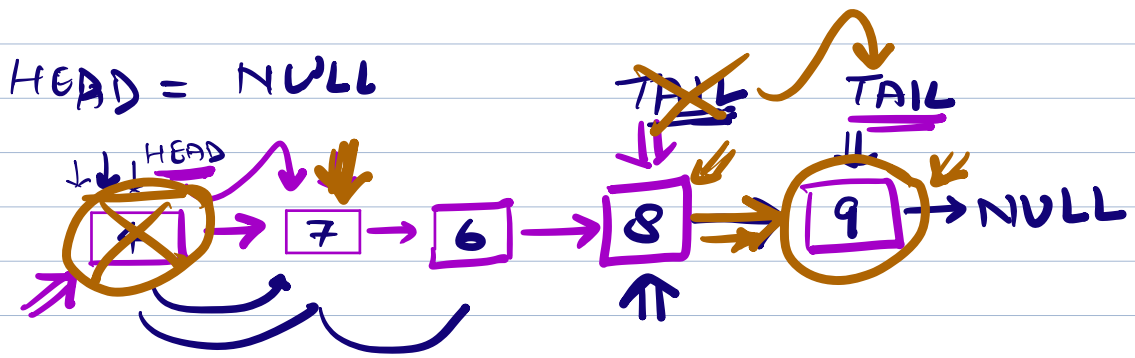
Dequeue()

```
if (front >= 0){
    temp = A[front];
    front++;
    return temp;
}
```

H.W. Define edge cases for enqueue & dequeue



E4, E7, E6, E8, E9, D



T.C. of insertion = ~~O(N)~~ O(1)

```
temp = head
head = head.next
free (temp);
```

SI

Stack

```
Push(10)
push(15)
pop(20)
pull
push(5)
push(40)
pop
pull
pull
```

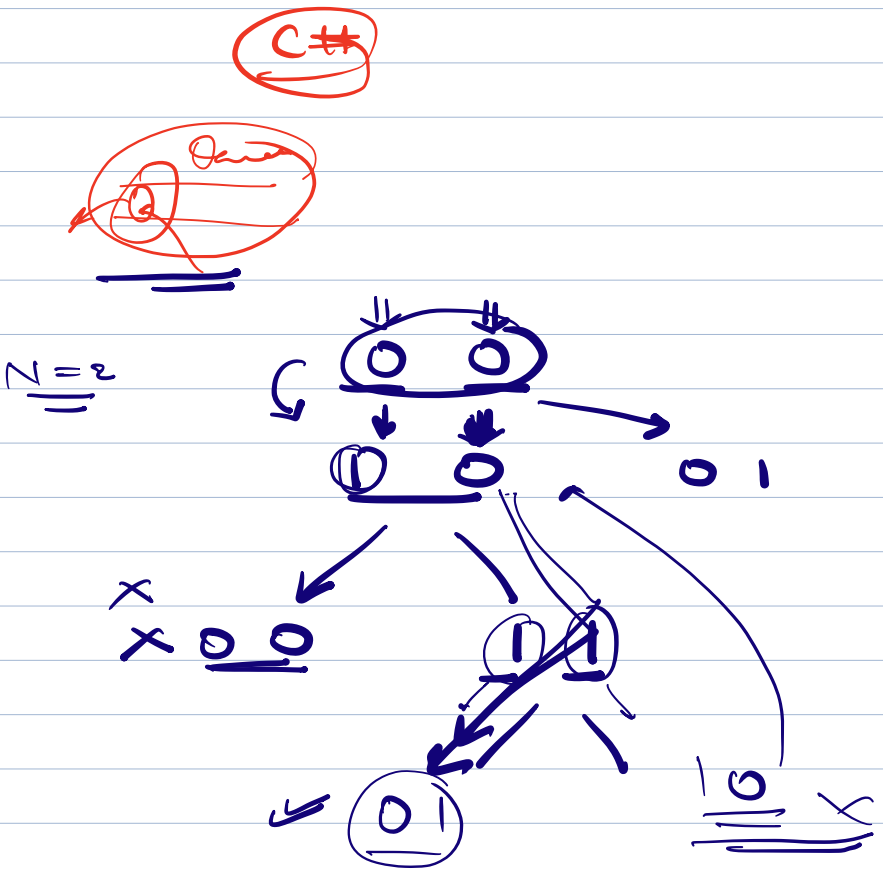
SO

20
40
15
15

Doubt (LL documentation for JS)

⇓

Add JS reference in Classroom Page



generate (N , 0) (

A ⇒ 3, 5, 14, 6, 12, 9, 1

B ⇒ 2, 3, 5, 11, 32, 16, 120

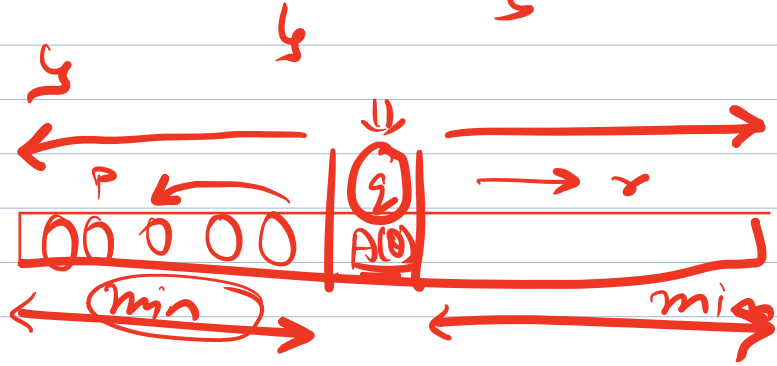
p, q, r

p < q < r

$A_p < A_q < A_r$

min

```
for (p = 0; p<N; p++) {
    for (q = p+1; q<N; q++) {
        for (r = q+1; r<N; r++) {
            if (A[p] < A[q] && A[q] < A[r])
                Total count++;
        }
    }
}
```



⇒ O(n²)

⇓

O(n)

fix q

Stack → O(1) Queue

(Head)