// Game → first person to reach 20 wins

Adapt

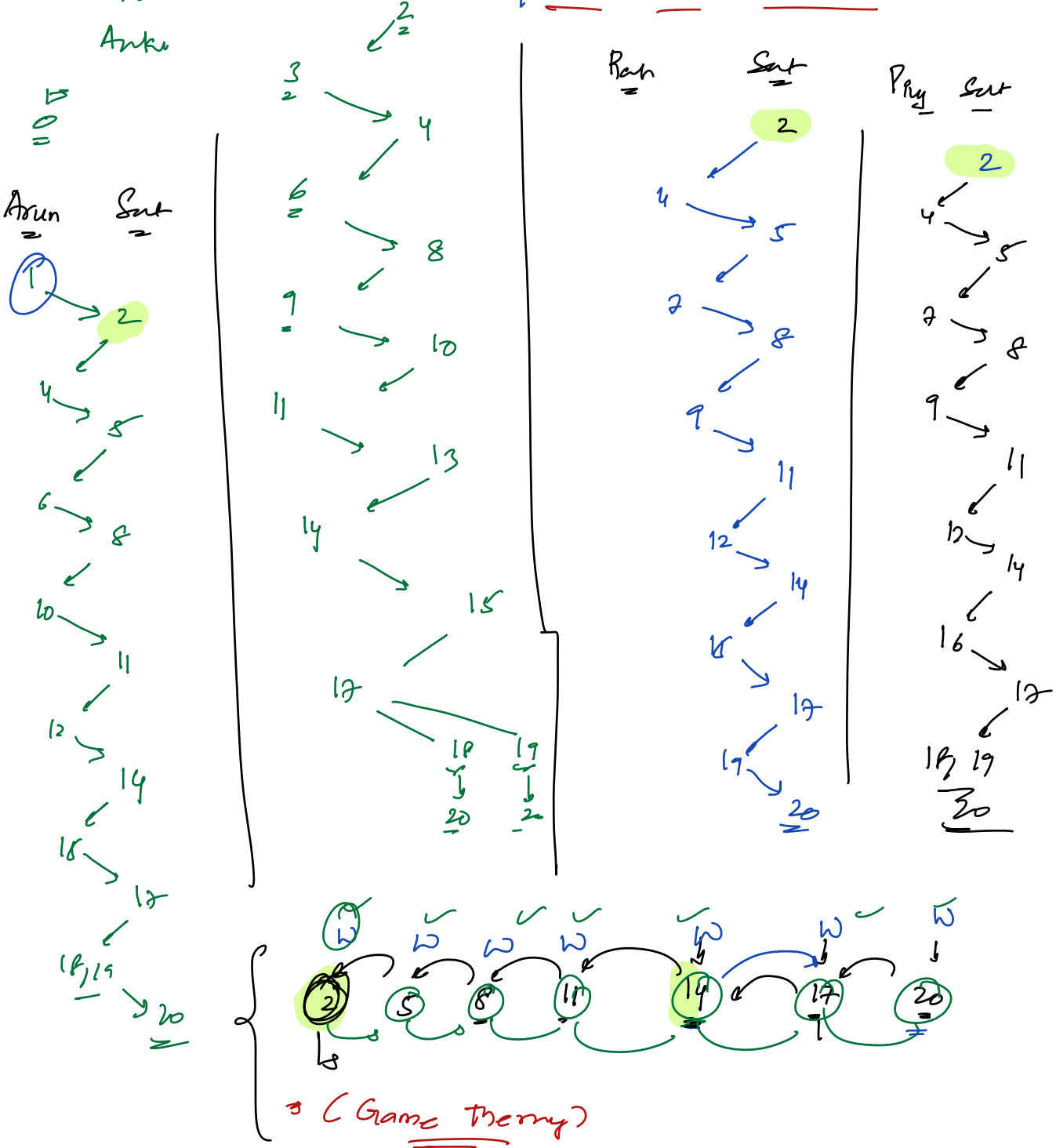
Ram

Anku

A ↔ B

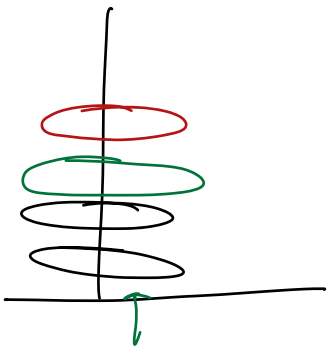
either 1 or 2

person who reaches 20 wins



28)

(Idly Cookers / Server Plates In marriage)



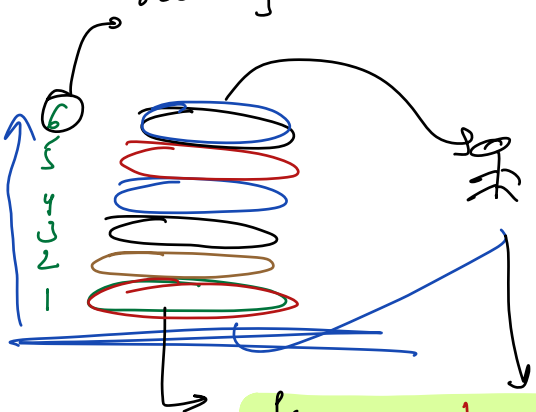
- Remove at top.
- Insert at top.

Container L Side is open

- Remove
- At Same Side Insert

Eg:

recently inserted plate = going out



first in last out or last in first out

FILO or LIFO

Data Structure = Stack

Stack:

DataStructure

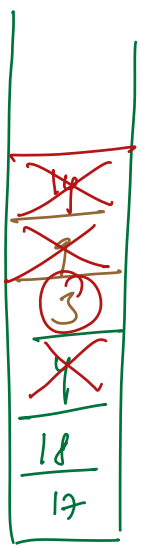
- push(n) = push at top
- pop() = top elem will pop
- top() = return top element
- size() = Total ele
- isempty() = Is stack empty or not?

Ex:

17 18 4 pop() top() 3 9 pop() 14 pop() top()

top() → 18

top() → 3



Real Time program Use

- Recursion
- Browsing history
- Undo Redo ← → Stack backward/forward/Stacks.
- Aps in Home, home button Recent
- Evaluating expression

3 + (4 + 6) → postfix → postfix expression → Evaluate → val

Invalid para (

)

3

- Stacks

Implement: Stacks → Arrays
→ Lists (Dynamic Arrays)
→ Linked List

0	1	2	3	4	5	6
10	20	40	80	90	120	160

int arr[7] =

int top = -1 (-1)

push(10)
push(20)
push(40)
push(60)
pop()
push(80)
push(90)
push(120)
push(160)
push(180)

pop(), pop(), pop(), pop(), pop(), pop(), pop()

pop() → (underflow) { There is nothing to delete }

int top = -1

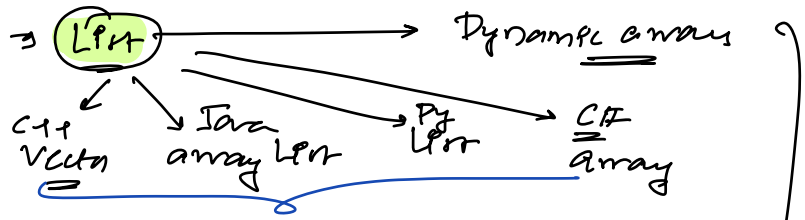
```
void push(int x) {
    // [Take care of overflow]
    top = top+1
    arr[top] = x
}
```

```
void pop() {
    if (IsEmpty()) {
        // underflow
        return 0, error
    }
    top = top-1
}
```

```
int peek() {
    if (IsEmpty()) {
        // underflow
        return Error
    }
    return arr[top]
}
```

```
bool IsEmpty() {
    if (size() == 0) {
        return True
    }
    else return False
}
```

```
int size() {
    return top+1
}
```



- insert() = insert at back
- delete() = delete at back
- size() =

→ O(1) (avg)

List<int> st;

st.insert(20) : [20] [40] [60]

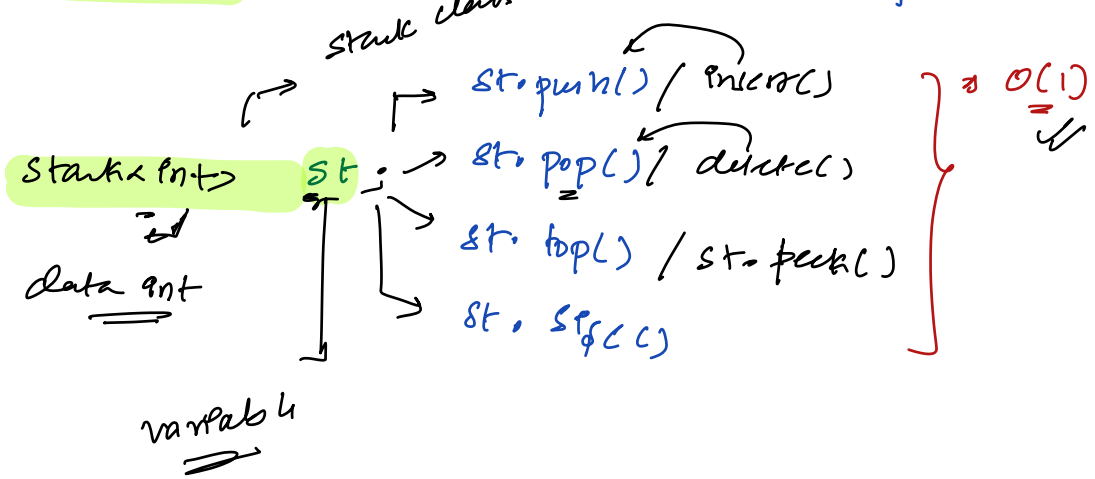
st.insert(40)

st.insert(60)

st.delete() : [20] [40]

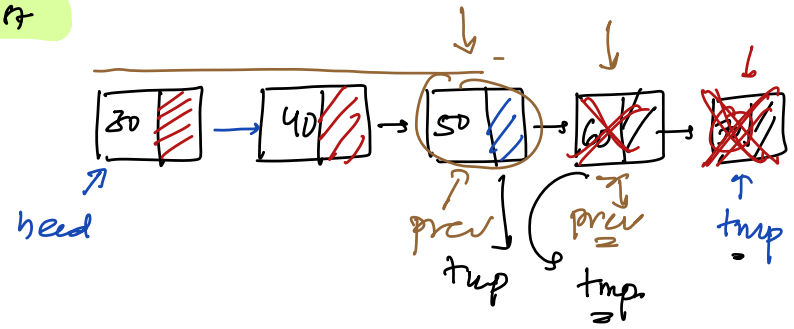
st.size() → 2

// → Stack class → check proper syntax in your language of choice.



Stack with Linked List

```
class Node{
    int data
    Node next
    Node(n){
        data = n
        next = NULL
    }
}
```



Node head = new Node(30)

push(40)
push(50)
push(60)
push(70)

push: O(1)
pop: O(N)

not effective

Inserting & during at Bank

⇒ push(): O(1) ✓

pop(): Iterate & get last but 1 & delete last node

pop():

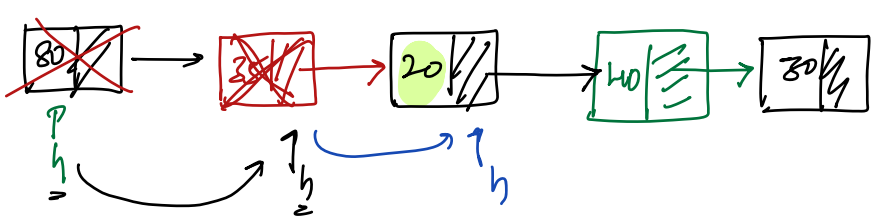
Stack: Insertion & deletion should happen from same end.

Deletion & Insertion from front()?

O(1)

O(1)

Node h = new Node(30)



if(head == NULL) you cannot pop head.data()

push(40) push(20) push(35) push(80) pop() pop() top()

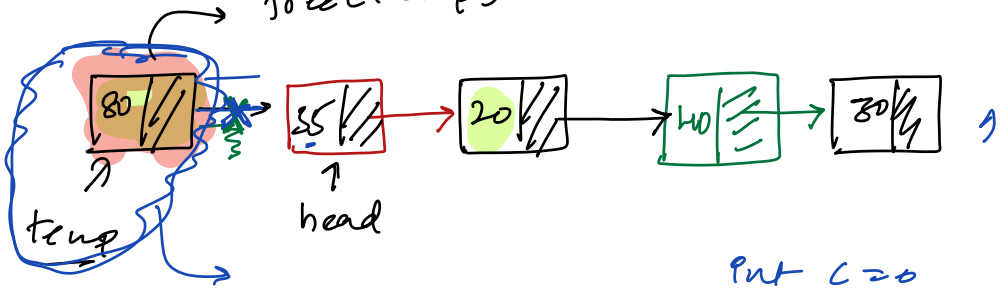
20

head.data()

Stack with LinkedList

Insert & delete at front ()

free(temp): (deallocated / removed from memory)



pop(head){

Node temp = head

head = head.next

temp.next == NULL

free(temp)

(deallocating memory for a node?)

C++ In Java:

int c=0

pop( ){

c--

// int ar[5]

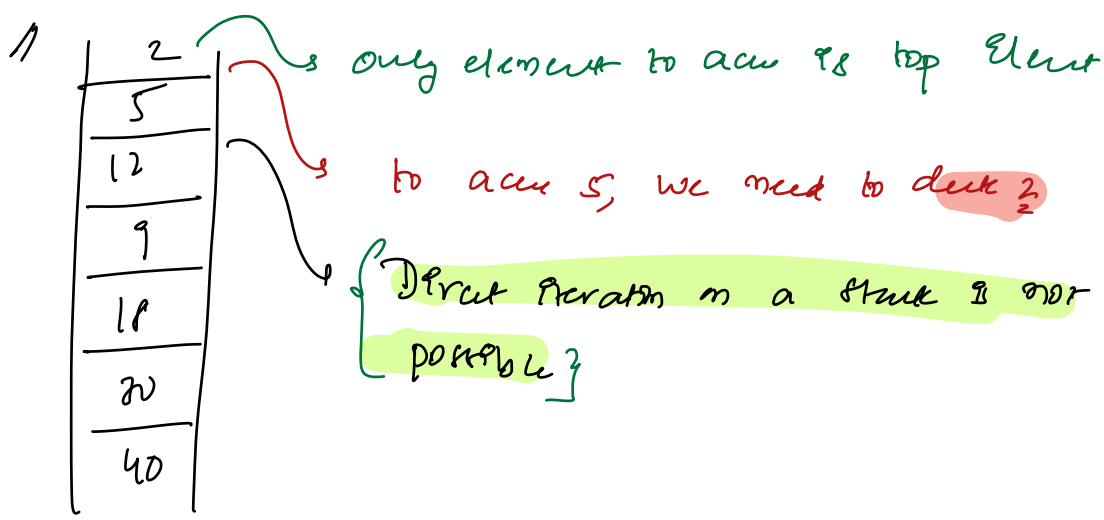
5 Elems

check

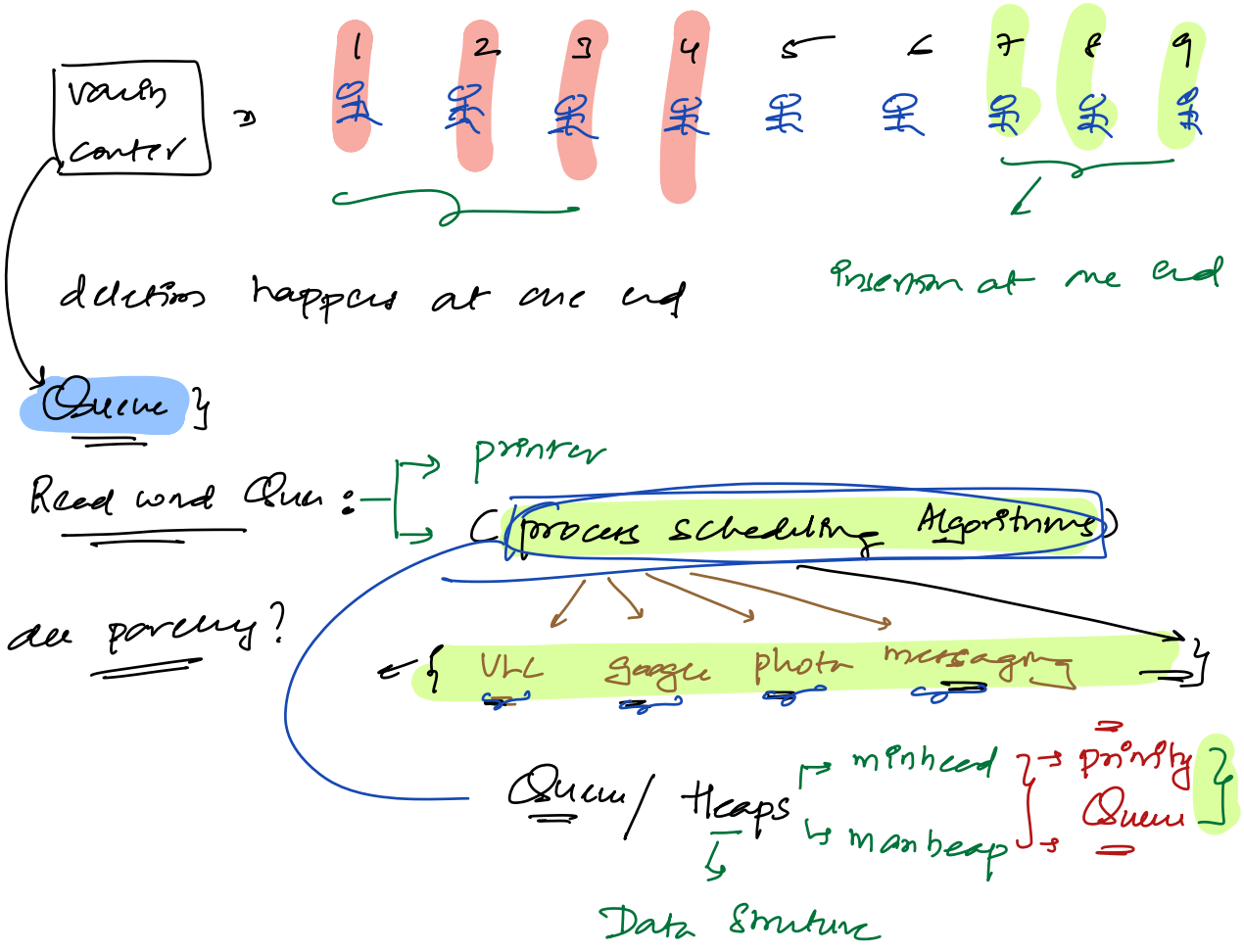
LinkedList<int> a

10: 45pm

```
Node head = NULL
int   c = 0

— push( head, x)
    Node t = new Node(x)
    t.next = head;
    head = t;
    c++
    return head
}

—— popC  ){
    if(head == NULL) {
        // underflow
        return "error"
    }
    Node temp = head
    head = head.next
    free( temp)
    c--
}

—  sizeC){
    return c;
}

⊥ topC){
    if(head == NULL) {
        // underflow
        return "error"
    }
    return head.data;
}
```



// Covid Vaccination



Eg: Queue:

- insert() / Enque() / : Insert at back
- delete() / Deque() / : Delete at front
- rear() / : [getting recently inserted element]
- front() / : [Element we are about to delete]
- size() / : [no: of Elements]

→ O(1)

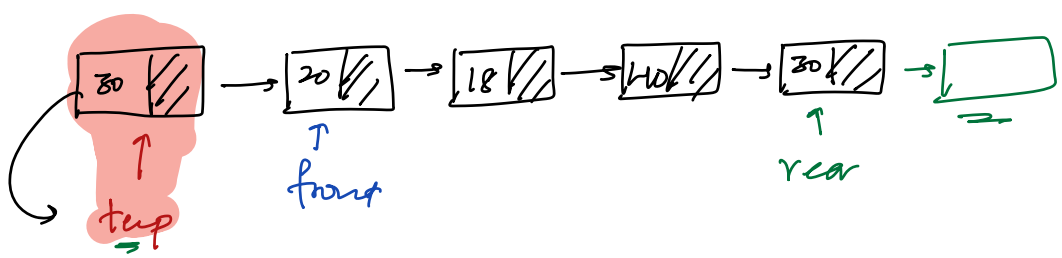
Eg:

30 (20) (18) rear() (40) front() deque() 80 front()

18 80 20

~~80~~ 20 18 40 80

// { Queues using LinkedList }



free memory

It's very good practice:

: { Enque() Deque() size() } : O(1) }

Doubts Session

→ Strings : (Longest palindromic String)

→ gray code : (Adv Batch)

→ Christmas Tree ✓

→ Subarray OR ✓

Help from Sam ✓

→ print Rubik ✓

→ k^{th} Symbol ✓

Any Ds
- → Linked
- → Stack
- → Tree

Sunday : ✓

order of Teaching : =

1) 7:30 PM → 10:30 PM

2) 11 am → 2 PM

Flatten Linked List → Adv

// Strictly optional →

11am – 2PM