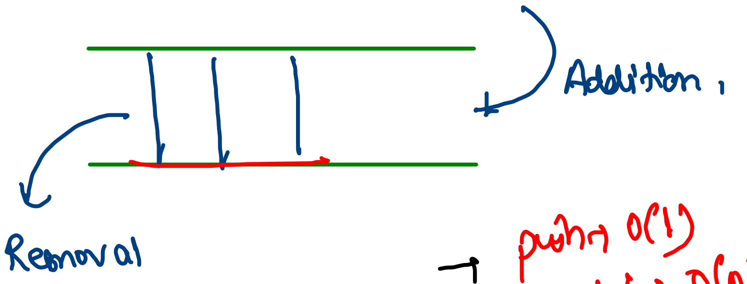
Queue to Stack → (Push Efficient)

push → O(1)

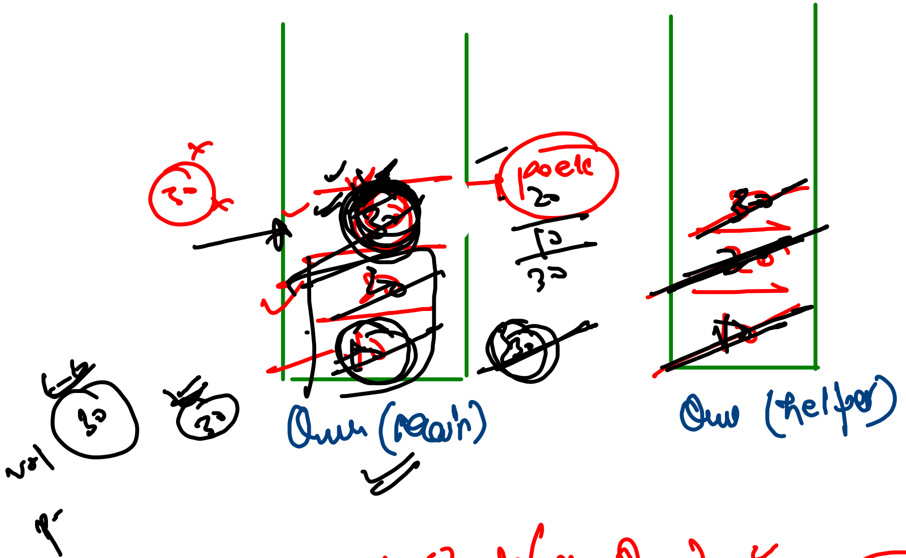
push → Add Last

pop → Remove first



push → O(1)
peek → O(n)
pop → O(n)

qu.add(data) } O(1)
size → mainsize()



push

For peek of stack (in Queue)

pop from main Queue & push it in helper Queue until size != 1;
again pop & push it in helper stack.

pop from helper Queue & push it in main Queue

stack → LIFO

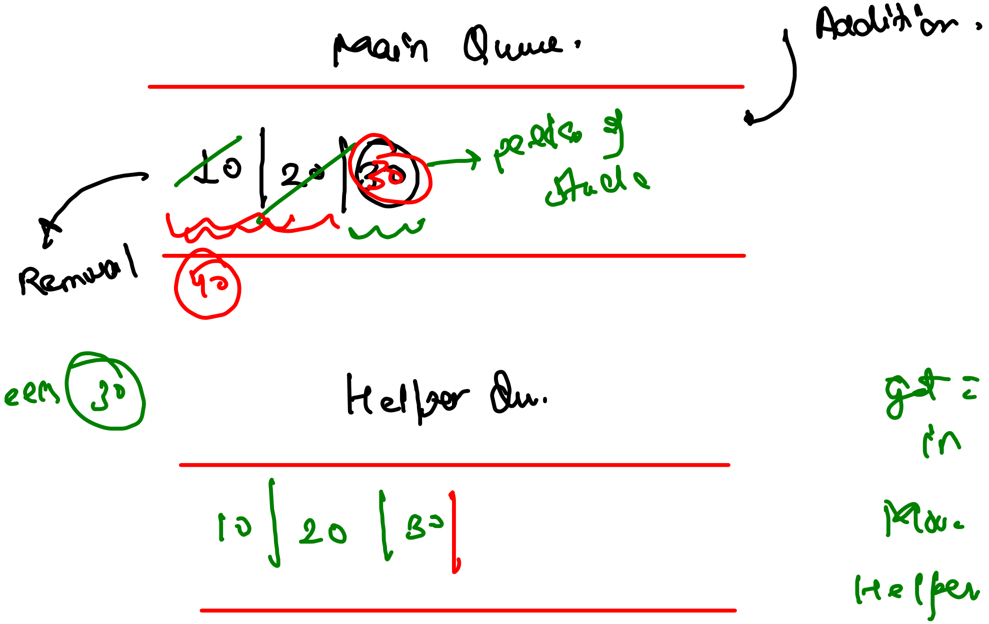
push . 10
push . 20
push . 30

peek → 30 → O(n)
pop → 30
peek → 20
pop → 20

push 40
peek → 40
push 50
peek → 50

push Efficient →

- push → 10
- push → 20
- push → 30
- peek → 30
  - push → 40
- pop → 30 → Remove
- peek → 20
- push → 40
- push → 50
- pop → 50 → Remove.
- peek → 40
- pop → 40 Remove



peek 30

Ist = < add in helper Q.
Move element from Helper to Main Q.

pop →

1. If size == 0 { message.
2. pop from main Q and push in helper Q. until size is greater than 1.

functions of Queue.

1. Add → last
2. Remove → first
3. front → first
4. size → initially even

Stack Using Queue →

pop Efficient →

(pop)

remove, peek → O(1)

O(1) } Qu → front

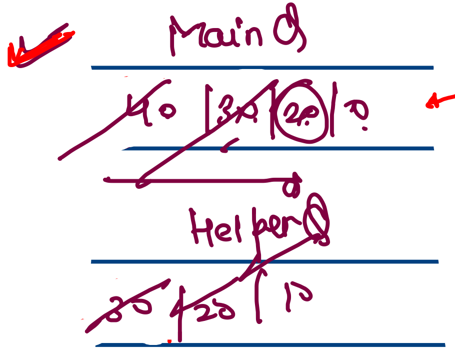
O(1) → last input element

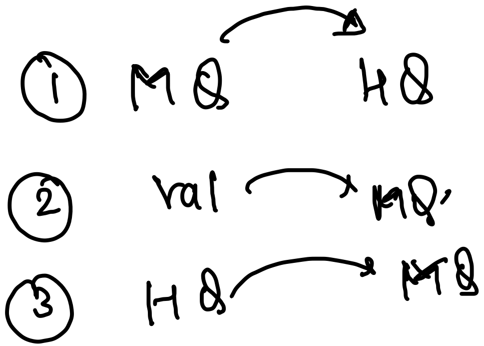
add → O(n)

size → O(1)

make a stack using a Queue
- → pop efficient
- → push efficient

- push → 10
- push → 20
- push → 30
- peek(): → 30
- push(40)
- peek → 40
- pop() → 40
- pop() → 30
- peek → 20





peek → Return mainQ.peek();

size → mainQ.size

pop();

push →
1. Remove Elements from mainQ & fill it in HelperQ,
2. Add val in mainQ,
3. Remove Elements from HelperQ & add it in mainQ.

Implement Queue using two Stack → (Add efficient) → push → O(1)

peek() → O(n)

pop → O(n)

push →

mainSt.push(val);

push → 10

push → 20

push → 30

front → 10

push → 40

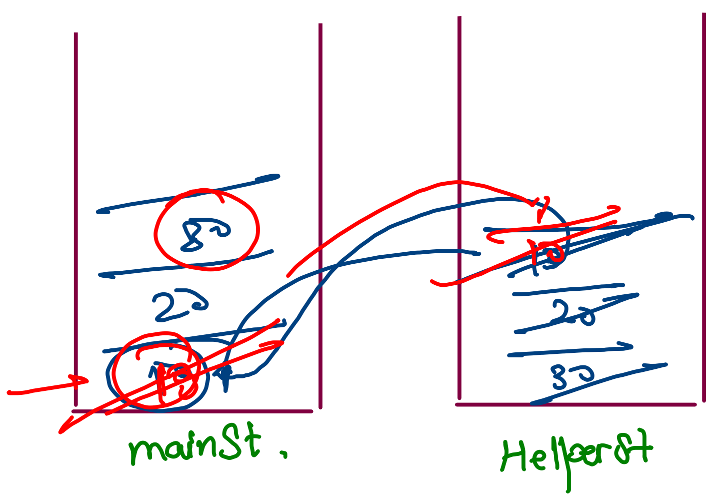
pop → 10

pop → 20

front → 30

val = 10

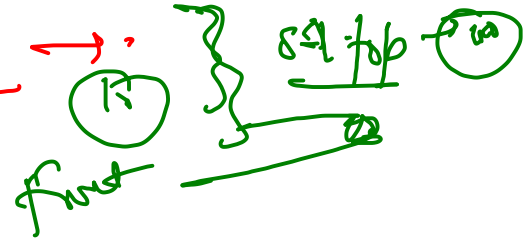
pop

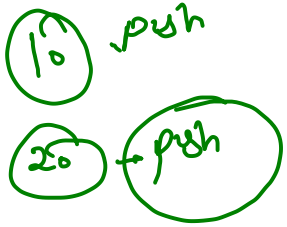


peek →

1. pop mainSt until size is greater than one, & add it in HelperSt
2. get the last & fill it in HelperSt.
3. Refill main from H.St.

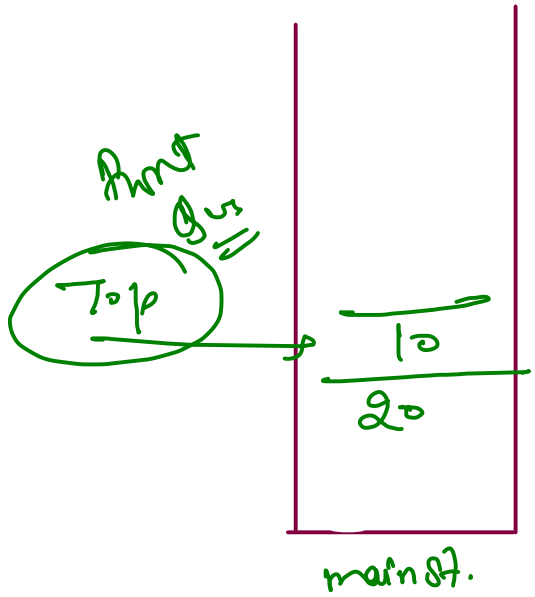
Implement Queue using Stack (pop Efficient)





- front
- pop
- Top

push

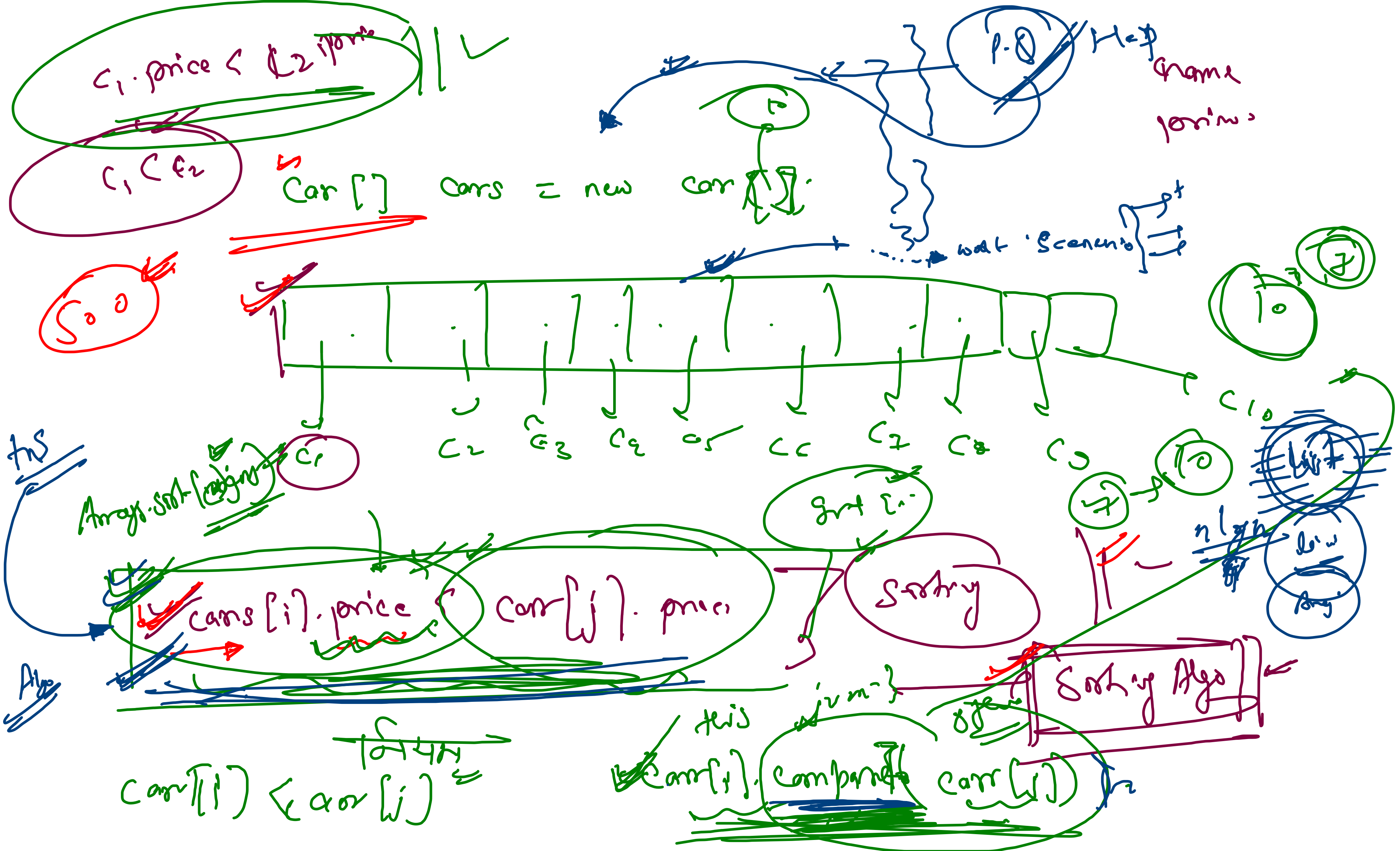


push →

→ remove fm mainSt. & push it helperSt.

→ main.push(val);

→ Refill mainSt.



c1.price < c2.price
c1 < c2
Car[] cars = new Car[10];
P.Q
Heap
name
price
'Scenario'
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
Arrays.sort(cars)
cars[i].price < car[j].price
sort
Sorting
Sorting Algo
nlogn
List
Algo
this
car[i].compareTo(car[j])
car[i] < car[j]