Normal Stack

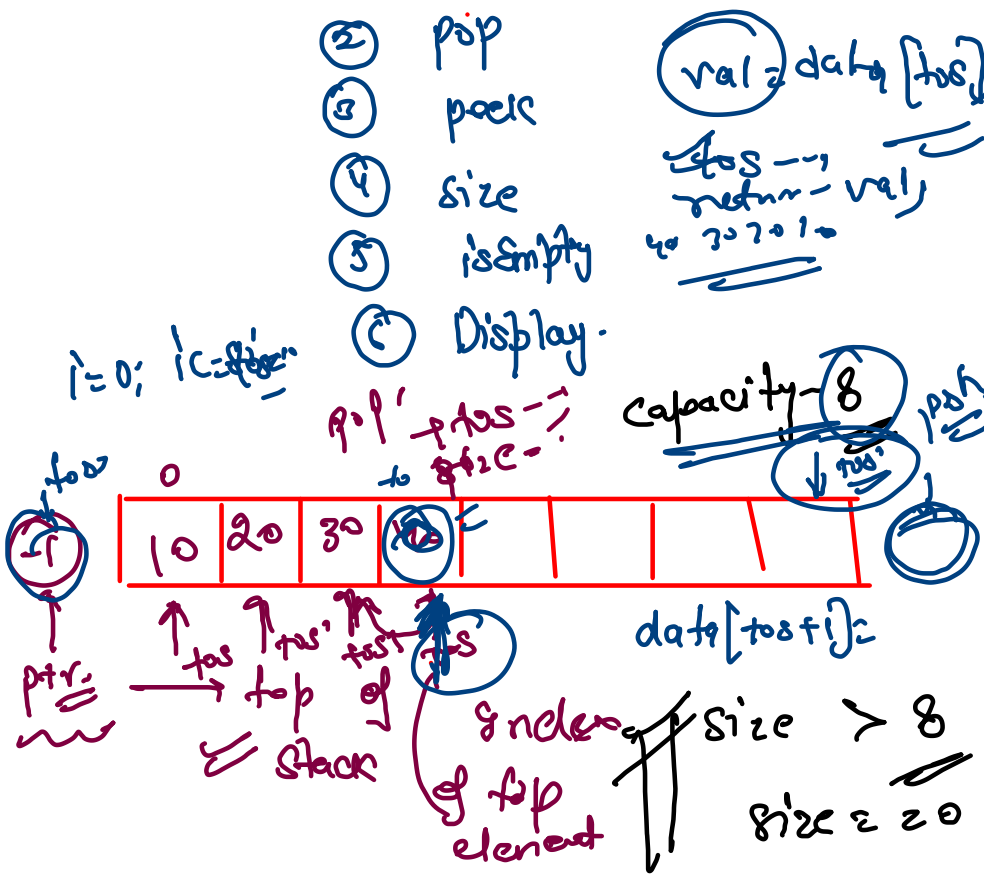
* Implement D.S. which can perform these kind of action.

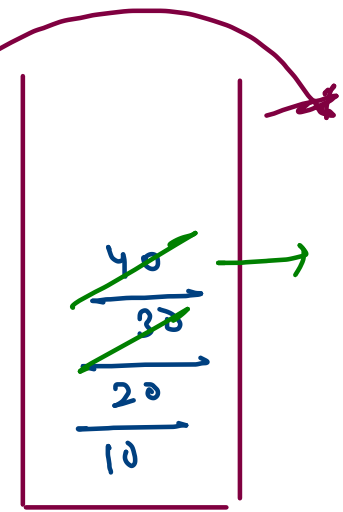
LIFO → semantic

Last in first out

functions on stack →

1. push
2. pop
3. peek
4. size
5. isEmpty
6. Display.

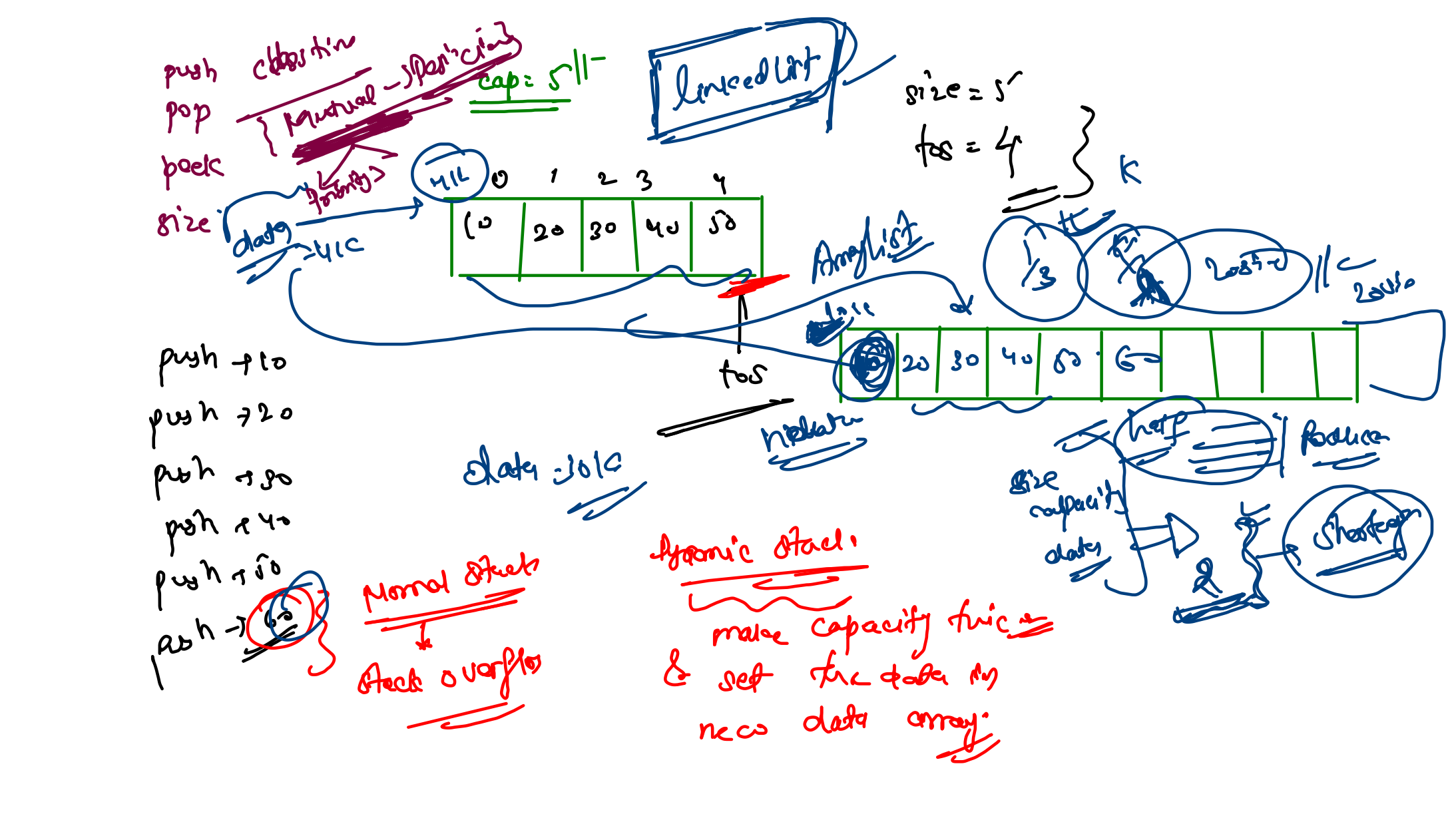




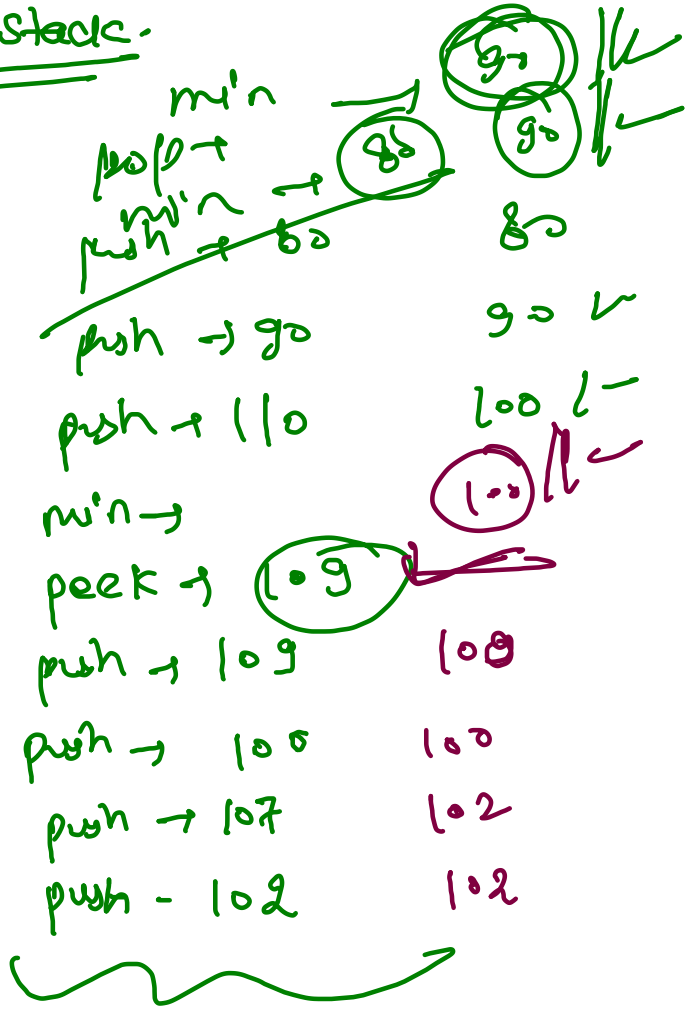
push - 10
" - 20
" - 30
" - 40
peek → 40
display - [10, 20, 30, 40]
pop() - 40
pop() - 30

size == 8 → stack overflow

size == 0 → pop / top → stack underflow


push
pop
peek
size
Abstraction
Implementation
cap: 5
Linked List
size = 5
tos = 4
data
0 1 2 3 4
10 20 30 40 50
tos
Array List
20 30 40 50
push → 10
push → 20
push → 30
push → 40
push → 50
push → 60
Normal Stack
Stack overflow
Dynamic Stack
make capacity twice
& set the data to new data array
Heap
Replica
size
capacity
data
Shortcut

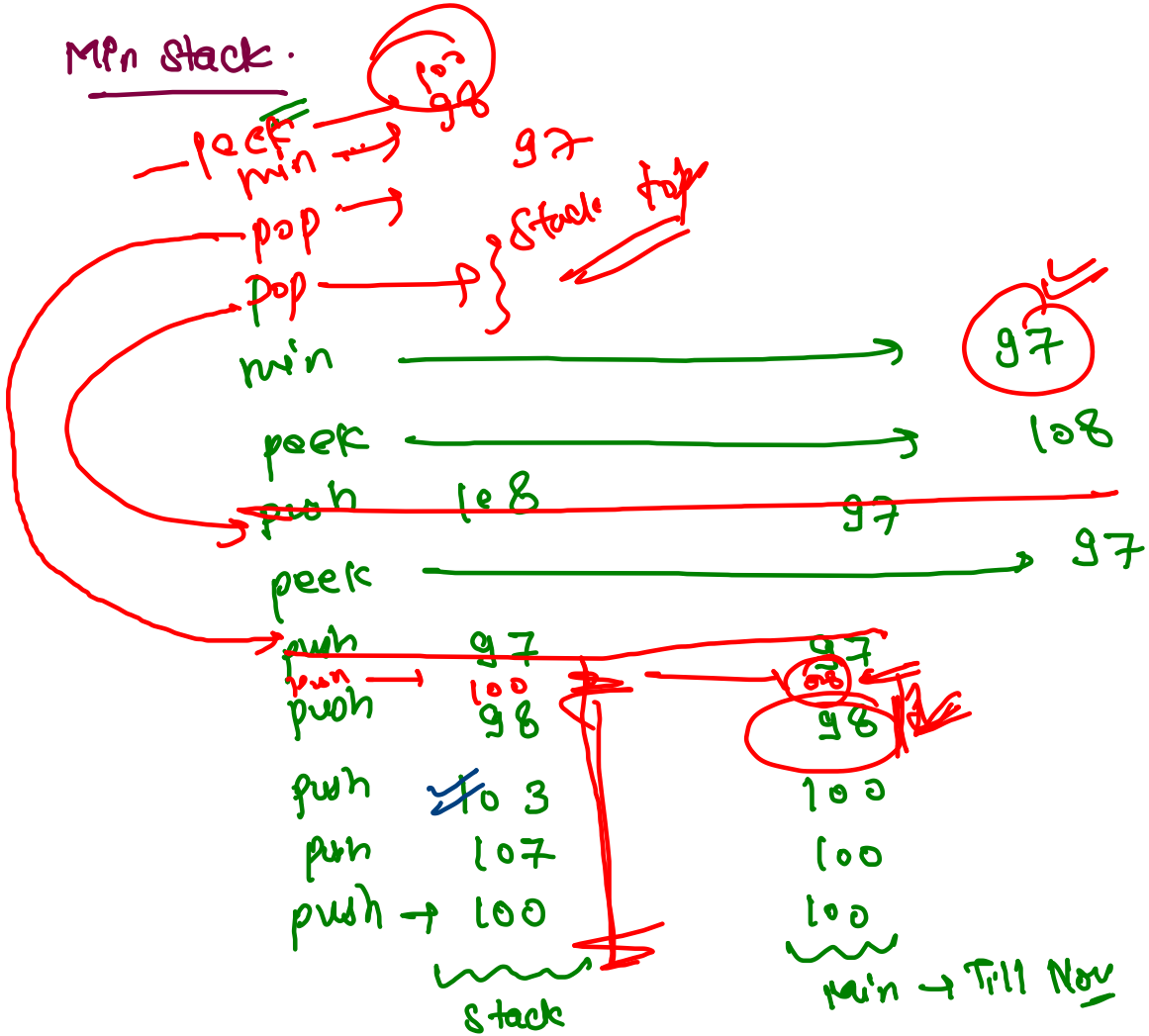
Min Stack.

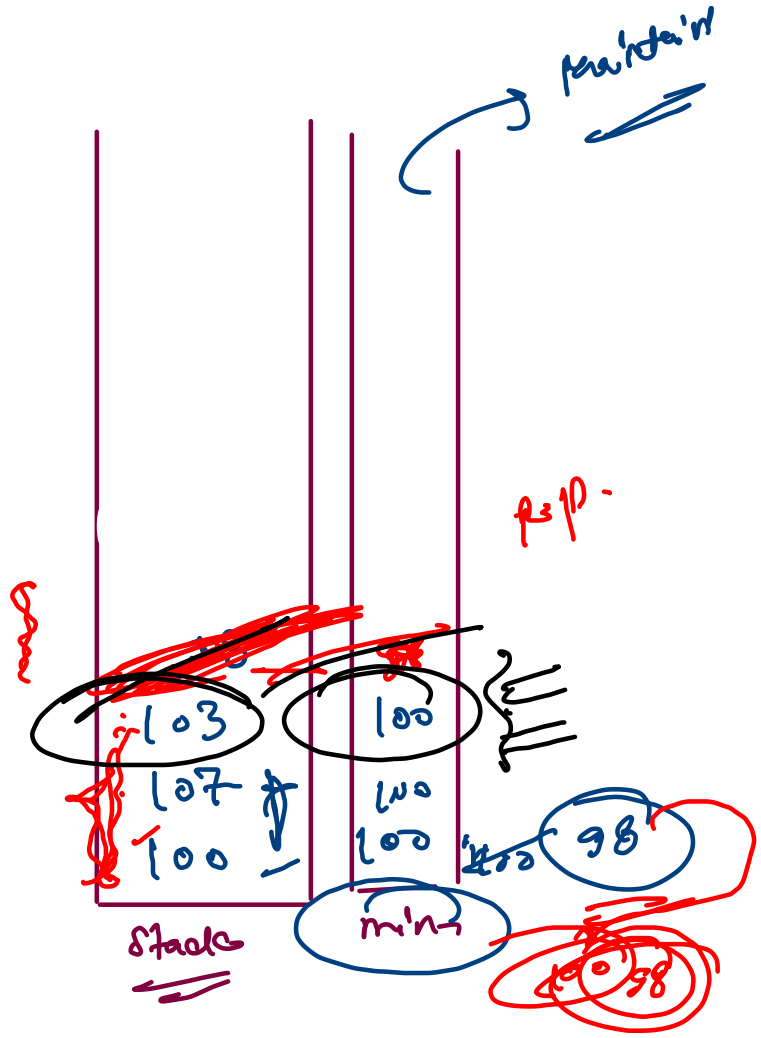


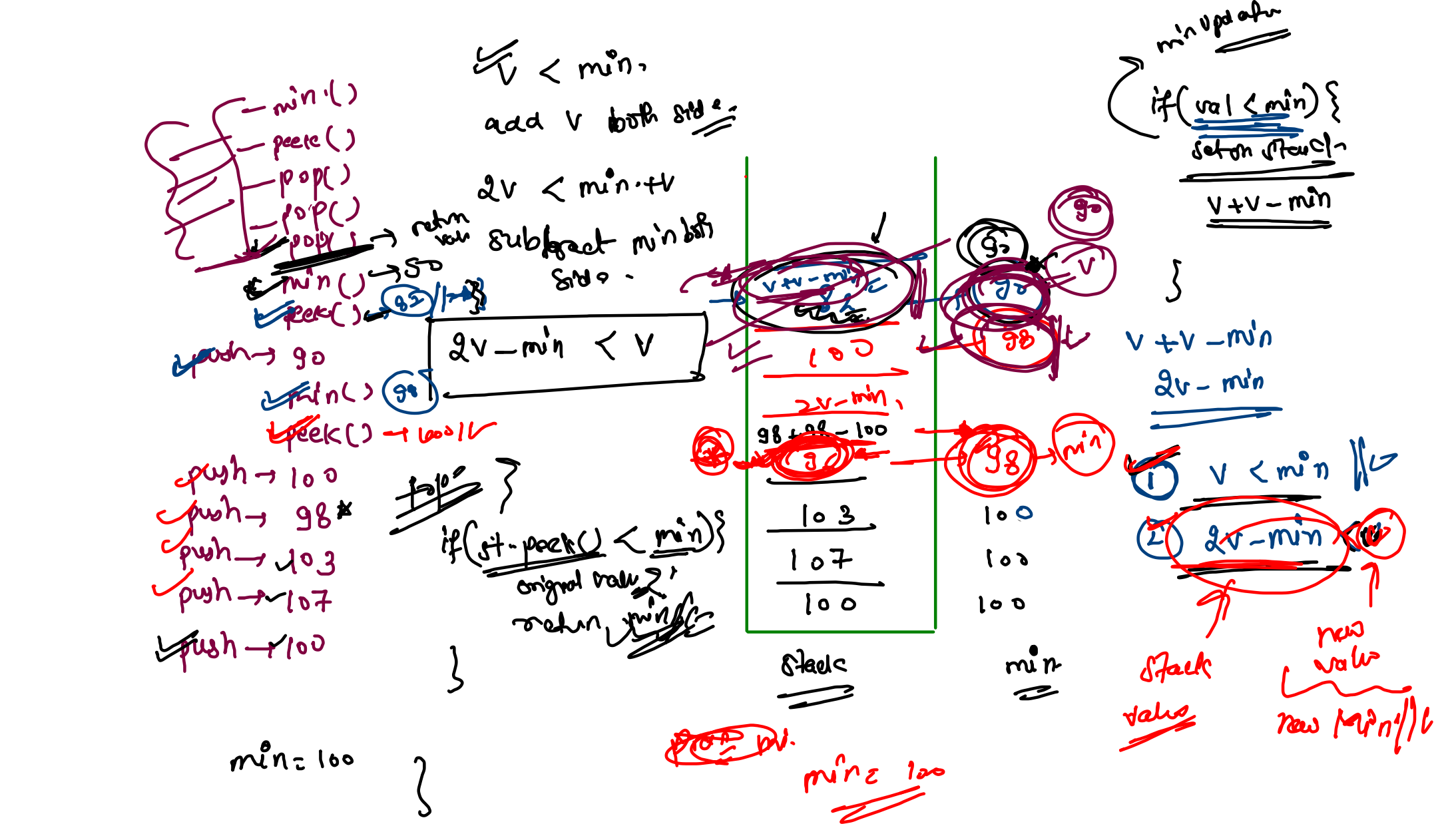
push
pop
peek
Size.

main } → think full fun.

Min Stack.


peek
min
pop
pop
Stack top
min
peek
push 108
peek
push 97
min
push 98
push 103
push 107
push 100
Stack
min → Till Now


pop.
Stack
min


min()
peek()
pop()
pop()
pop()
return val
min() → 50
peek() → 98
push → 90
min()
peek() → 100
push → 100
push → 98
push → 103
push → 107
push → 100
min = 100
if(v < min)
add v both stack
2v < min + v
subtract min both side
2v - min < v
top()
if(st.peek() < min)
original value
return min
v + v - min
100
2v - min,
98 + 98 - 100
98
103
107
100
Stack
min
min update
if(val < min){
store stack
v + v - min
2v - min
v < min
2v - min
Stack value
new value
new min
min = 100

```
push(val) {
    if (val >= min) {
        st.push(val);
    } else {
        st.push(val + val - min);
        min = val;
    }
}
```

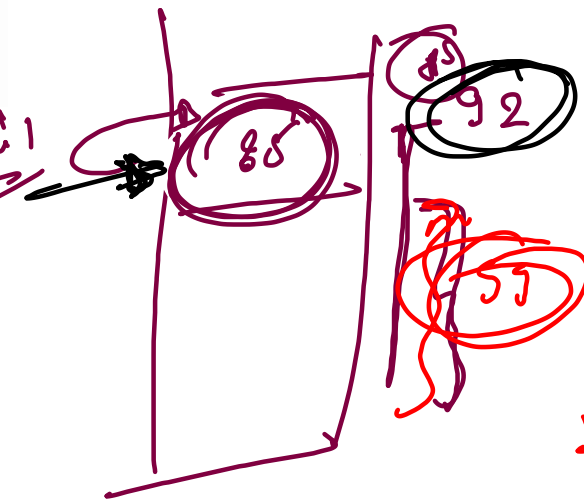
```
V = min
2v-min = st.peek
  ↑
 omin
2v_omin = st.peek();
V = min
2*min - omin = st.peek
2*min - st.peek = omin
min = 2min - st.peek();
```

```java
int pop() {
  // write your code here
  if(data.size() == 0) {
    System.out.println("Stack underflow");
    return -1;
  }
  if(data.peek() < min) {
    // invalid peek / fake peek
    int val = min;
    // I wil return val after set old min
    min = 2 * min - data.pop();
    return val;
  }else {
    int val = data.pop();
    return val;
  }
}
```

data.peek > min

103

val = 103



peek() < min

- St.peek() = 85
- val = 92.

omin ?

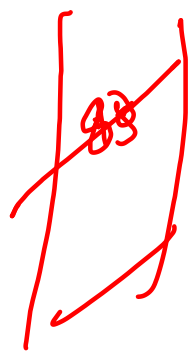
2 * v - omin = St.peek() ?

omin = St.peek();

min = 2 * min - St.pop() ?

2 * min - 85

Hashing



pmin = 2 * 92 - 85

= 184 - 85

= 99

v+v-min

v = go

pop( ){

// return value → min

if(st.peek() < min){

after removal min ?

or

}

peek( ) 82 //- st.peek() ≠ 2v - prev_min =

v = go //

}

v = min -

prev_min = 2xgo - 82

= 180 - 82

= 98 //=

prev_min = 2v - st.peek();

min

min = 2v - st.peek();

= 2*min - st.peek()

Queue →

FIFO → first In first Out

push / Add

pop / remove.

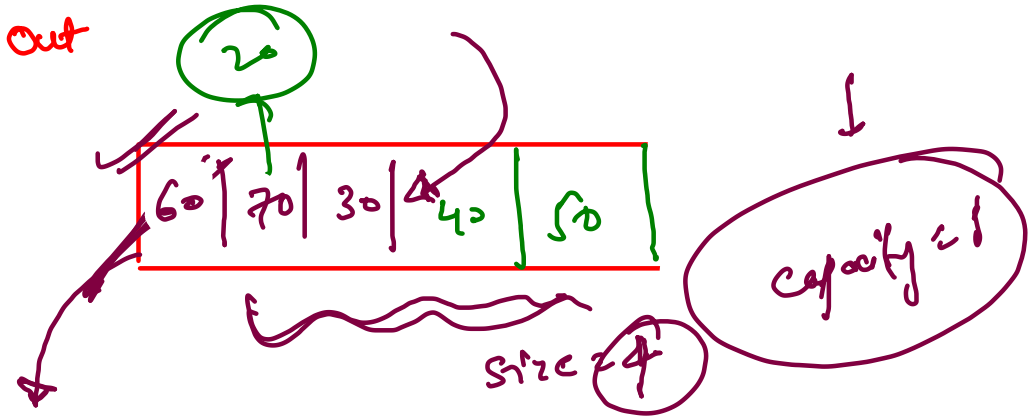
front / peek

size.

Display

Way

Queue qu = new ArrayDeque();



pop()

push (70)

display.

30 40 50 60 70

push 10

20

30

pop. → 10

40

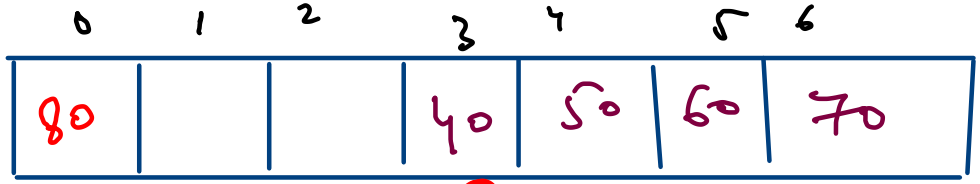
peek() → 20

push 50

push 60 push 70

Queue overflow

1. Size.
2. front
3. Rear



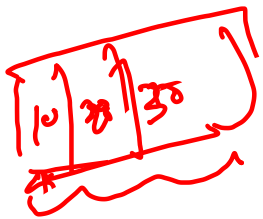
size = 8 3 4 5

rear = front + size - 1

rear = (size + front) - 1

= 4 + 1

40



3 + 4 = 7 Index out of bound

(3+4) % arr.length;

(3+5) % arr.length

push rear = front size;

arr[rear] = val;

rear = (front + size) % arr.length

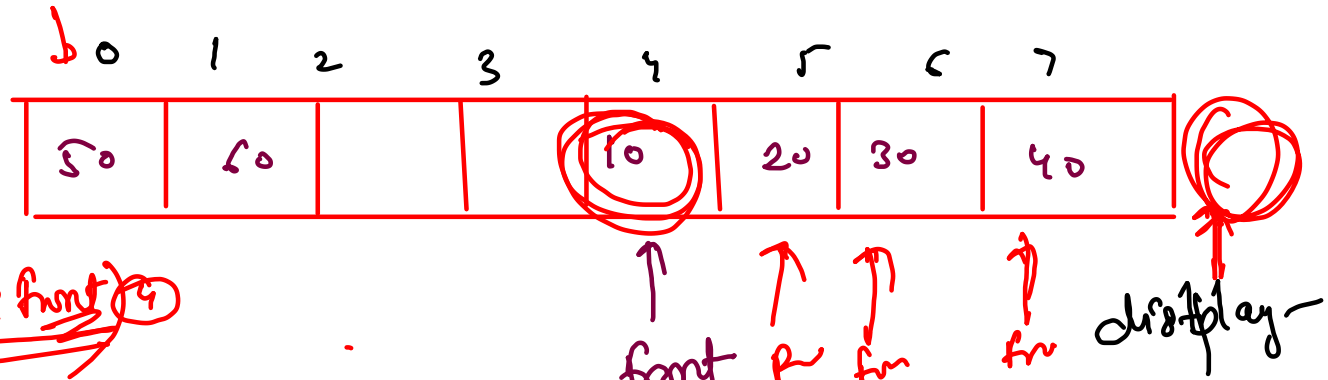
10 →
20 →
30 →
40 →
50 →
pop →
60 →
70 →
pop →
pop →
80 →

10 arr[front]
front ++
size--;

20 arr[front]
front ++
size--

front

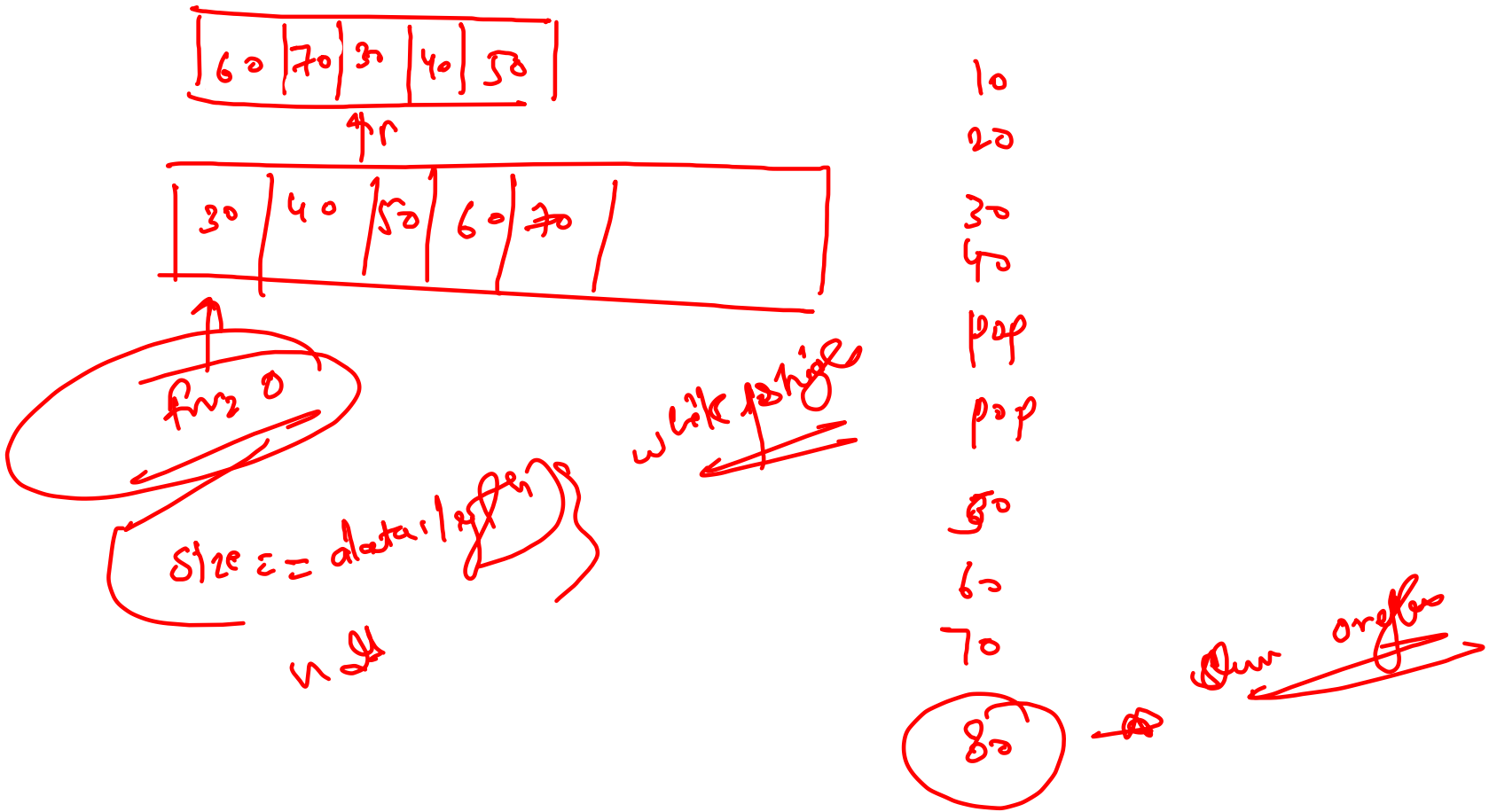
peek() → 90



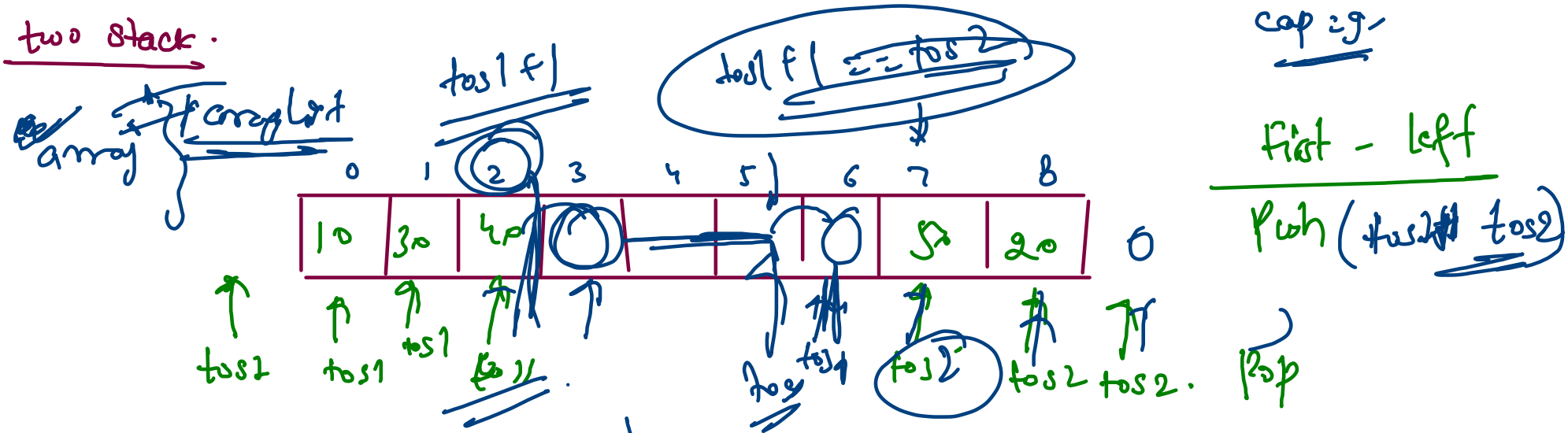
0	4
1	5
2	6
3	7
4	8 % 8 = 0
5	9 % 8 = 1

```
for(int i=0; i<size; i++){
    indx=(i+front)%length;
    System.out.print(data[indx]+" ");
}
System.out.println(); // next line
```

front = (front+1)%data.length


60 70 30 40 50
tr
30 40 50 60 70
fun 0
size == data.length
null
which range
10
20
30
40
pop
pop
50
60
70
80
new one

two stack.



cap :9-

first - left

Push (tos1++ tos2)

Pop

push 10 - 1

push 20 - 2

30 - 1

40 - 1

50 - 2

60 - 2

70 - 2

80 - 1

Size : tos1+1

availSpace = 9 - tos2

to p

Size

if (tos1 + 1 == tos2) {

Stack + ArrayList LinkedList

Queue →

}

Second - Right end

Push

Pop

top.

size