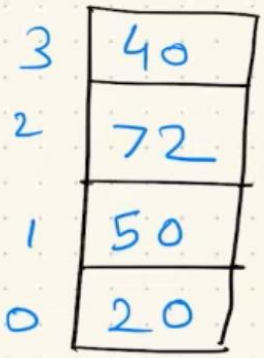
```
add ( val) {   Θ(1)
    if ( isFull() ){
        return ERROR
    }
    A[tail] = val
    tail ++
}
```

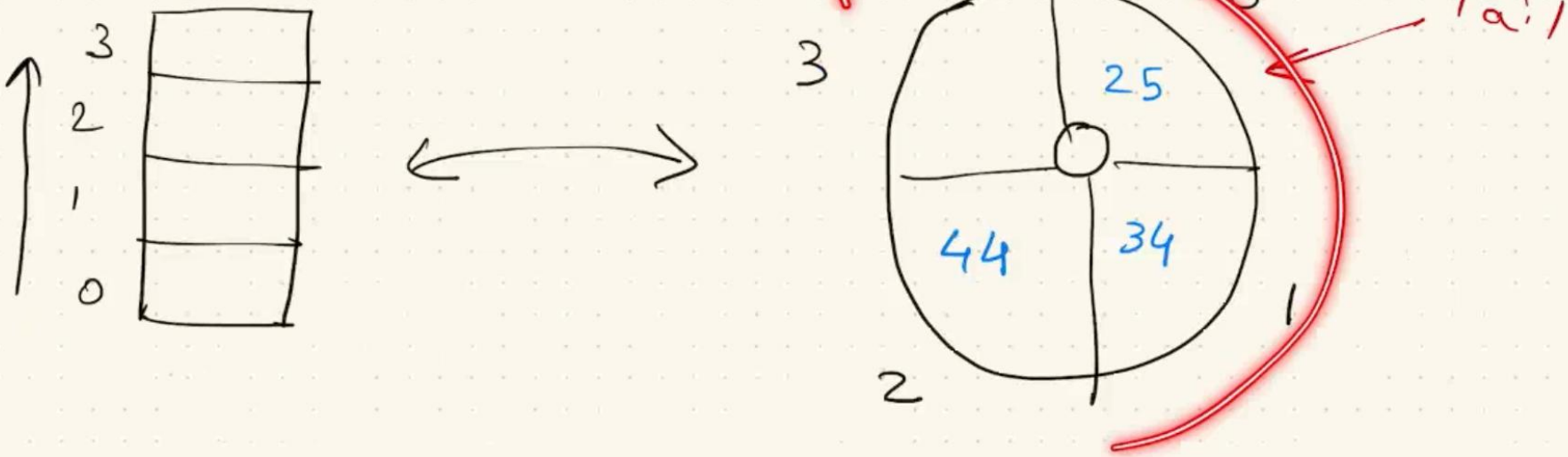
```
delete() {   Θ(1)
    if(isEmpty()) {
        return ERROR
    }
    answer = A[head]
    head++
}
```



createNew (4)
add (20)
add (50)
add (72)
add (40)
delete() → 20
add (49)

head = 1 tail = ~~0 1 2 3~~ 4

Circular Queue



Data : Array A, integer head, integer tail

```
createNew(capacity){
    A.createNew(capacity+1)
    head=0
    tail=0
}
```

$\Theta(1)$

```
isEmpty(){
    if(head==tail){
        return TRUE
    }
    return FALSE
}
```

$\Theta(1)$

```
Add1(val1){
    if(val1==capacity){
        return 0
    }
    return(val1+1)
}
```

$\Theta(1)$

```
isFull(){
    if(head==cAdd1(tail)){
        return TRUE
    }
    return FALSE
}
```

$\Theta(1)$

```
add ( val ) {
    if ( isFull() ){
        return ERROR
    }
    A[tail] = val
    tail = cAdd1 (tail)
}
```

$\Theta(1)$

```
delete () {
    if ( isEmpty() ) {
        return ERROR
    }
    answer = A[head]
    head = cAdd1 ( head )
    return answer
}
```

$\Theta(1)$

Implement ADT1 using ADT2

P

Stack

Queue

R

Queue

Stack

↳ implemented using Queue

P R

operation P1 → operations over R

operation Pn → Operations over R

Queue using Stack

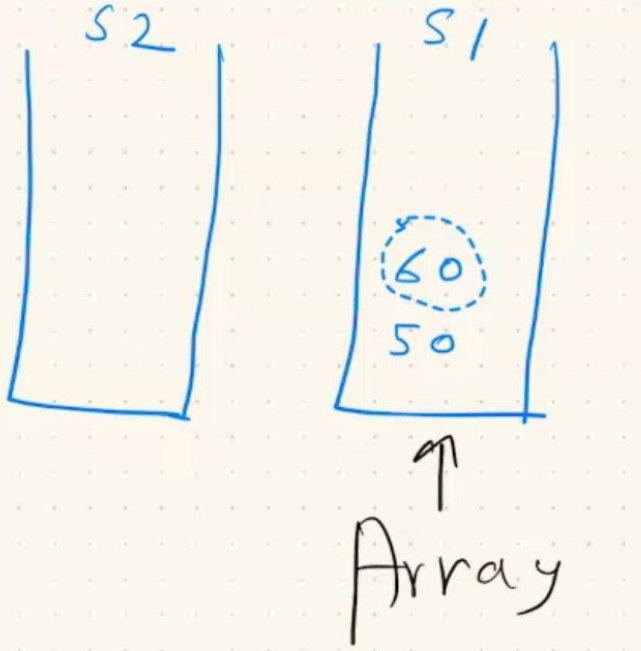
createNewQueue ⟶

isEmpty

isFull

add ⟶ Push on to S1

delete ⟶ empty S1 and push it on S2
answer = pop from S2
empty S2 and push it on S1



Queue using Stack

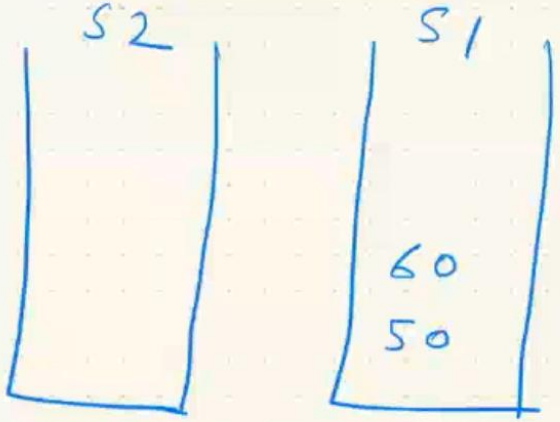
createNewQueue ⟶ Create S1 and S2

isEmpty ⟶ S1.isEmpty()

isFull ⟶ S1.isFull() 20

add ⟶ Push on to S1

delete ⟶ empty S1 and push it on S2
answer = pop from S2
empty S2 and push it on S1



Array

Counter

```
createNew ( capacity) {
        S1.createNew (capacity)
        S2. createNew (capacity)
}
```

$\Theta(1)$

```
isEmpty () {
    return ( S1. isEmpty () )
}
```

$\Theta(1)$

```
isFull () {
   return ( S1.isFull() )
}
```

$\Theta(1)$

```
add ( val ) {
    result = S1.push ( val )
    return (result)
}
```

True / False

Success / Failure

```
delete () {
    S1.emptyStack (S2)        Θ(n)
    answer = S2.pop()
    S2.emptyStack(S1)
    return(answer)
}

Stack :: emptyStack ( Stack Target ) {        Θ(n)
    while (!(isEmpty()) ) {
        Target.push(pop())
    }
}
```

Stack using Queue

Data: Queue Q1, Queue Q2

createNew ⟶ Q1.createNew, Q2.createNew

isEmpty ⟶ Q1.isEmpty

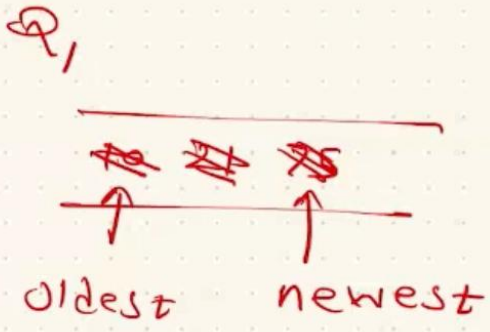
isFull ⟶ Q1.isFull

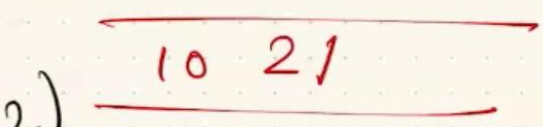
push ⟶ Q1.add

pop ⟶ Q1.transferAllButLast(Q2)

answer = Q1.delete

Q2.transferAll(Q1)





answer = 75