NO. _____

Title:

Stacks & Queues.

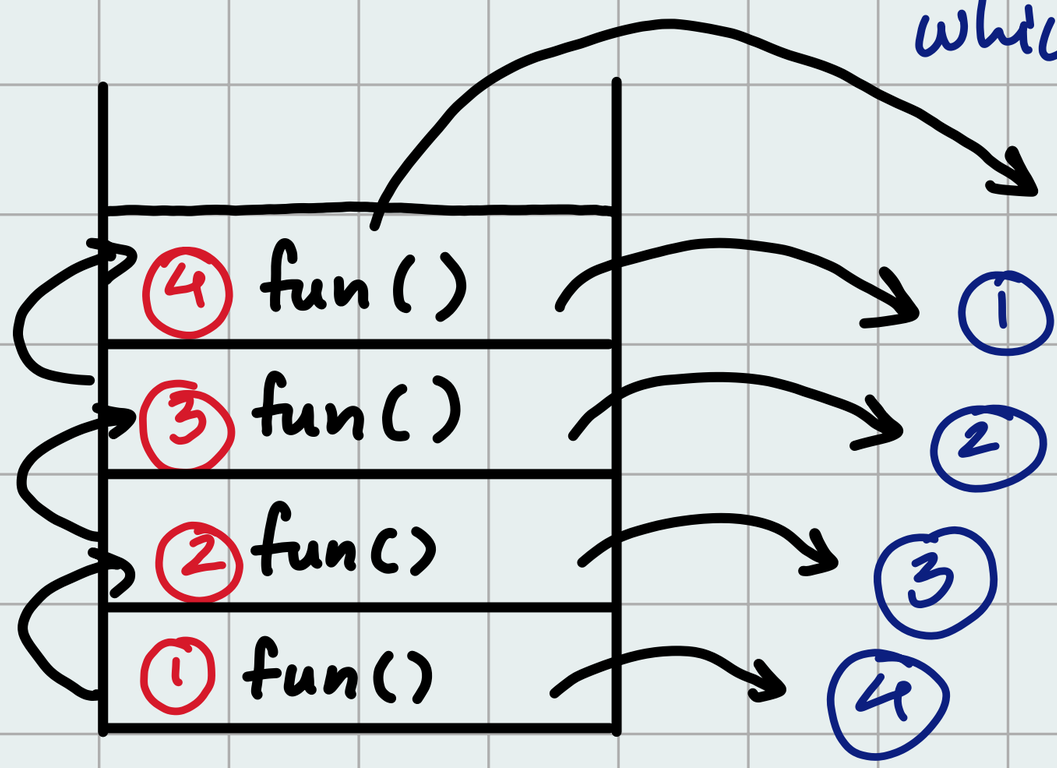
Enjoy the most efficient handwriting experience! Taking notes on the go, whether for inspiration, ideas, knowledge learning, business insights, or even sketches...

NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

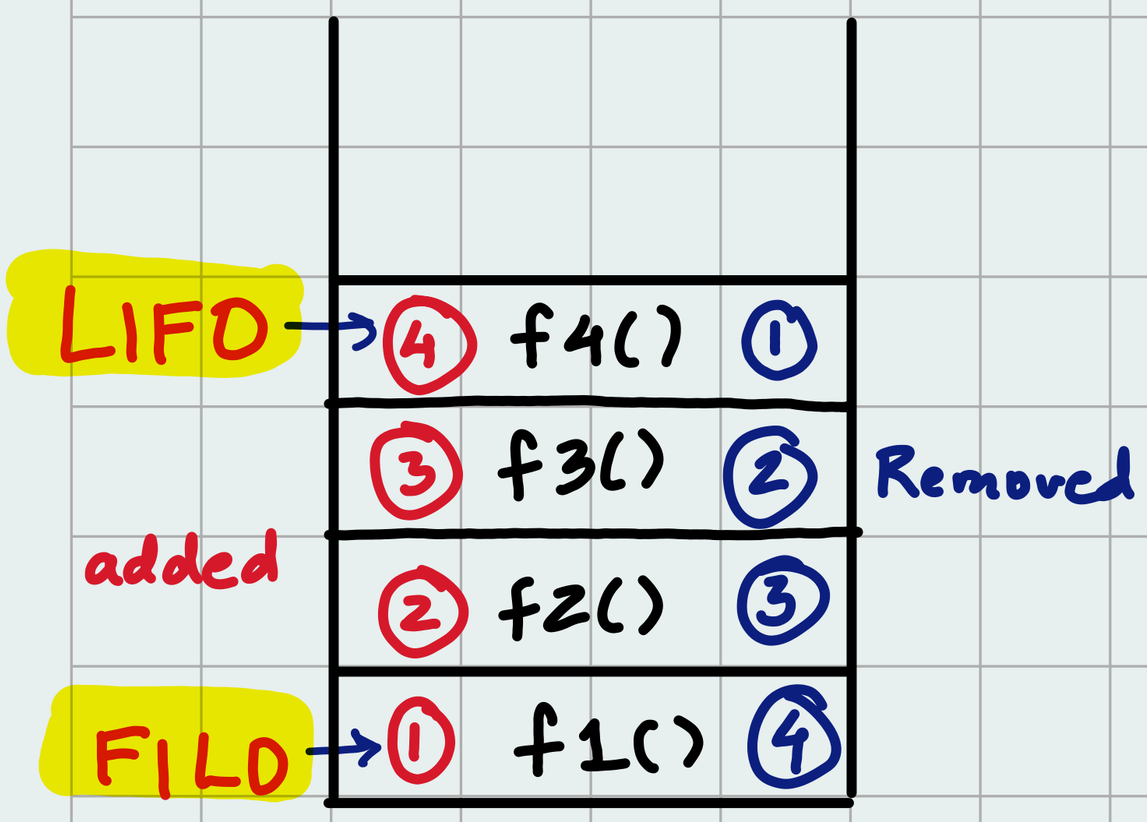
Stacks



which one was removed first?

function call goes over here First function calls another function second function calls third function and so on

When last function called is finished performing it will be popped out of stack All functions will be poped out one by one as their performance is finished



Here we can notice that the item that came in at the very first is the last item that will be removed

This u stack

FILO

First In Last Out

that means first item that was in is the last item that was removed

both are

LIFO

Last In First Out

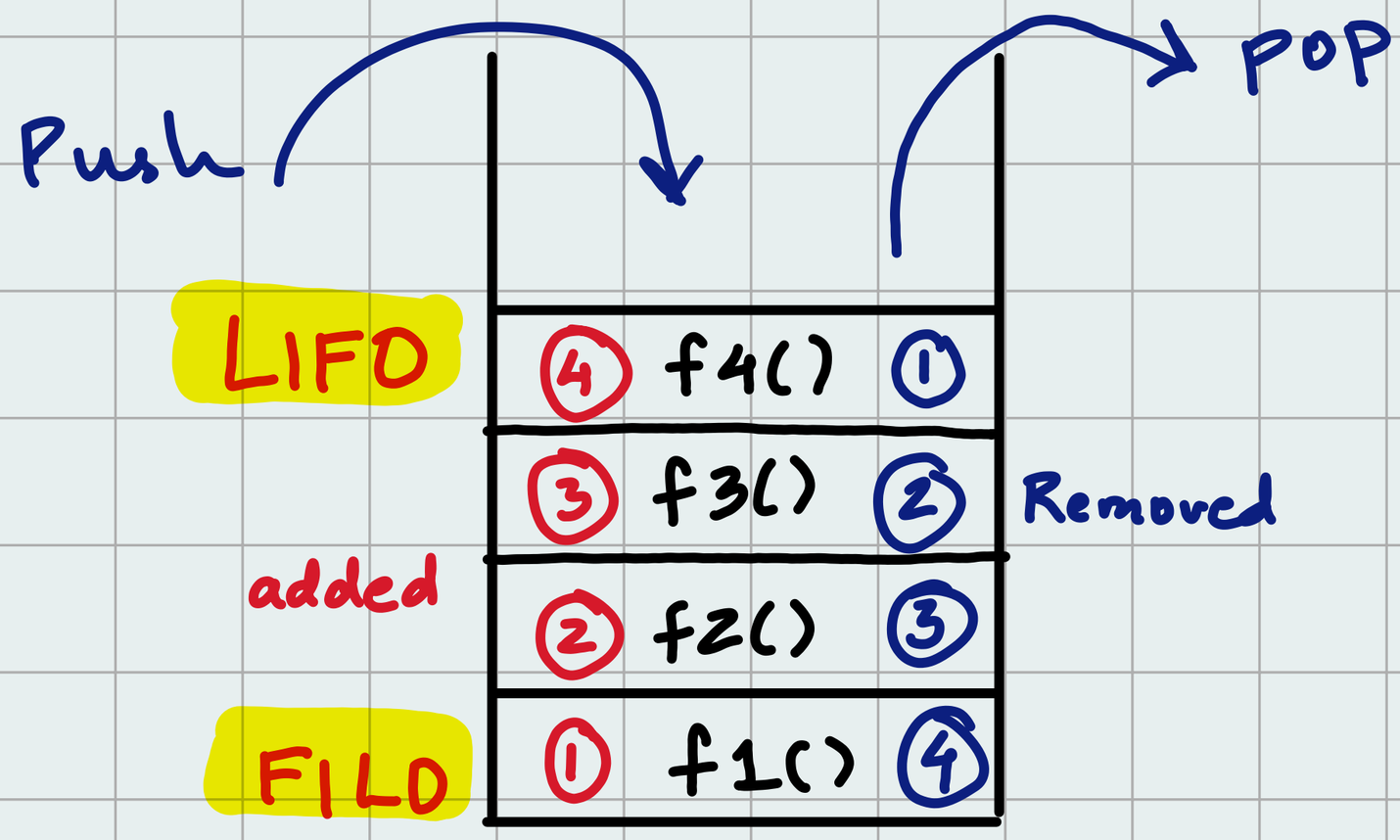
that means Last item that was in is the first item to be removed

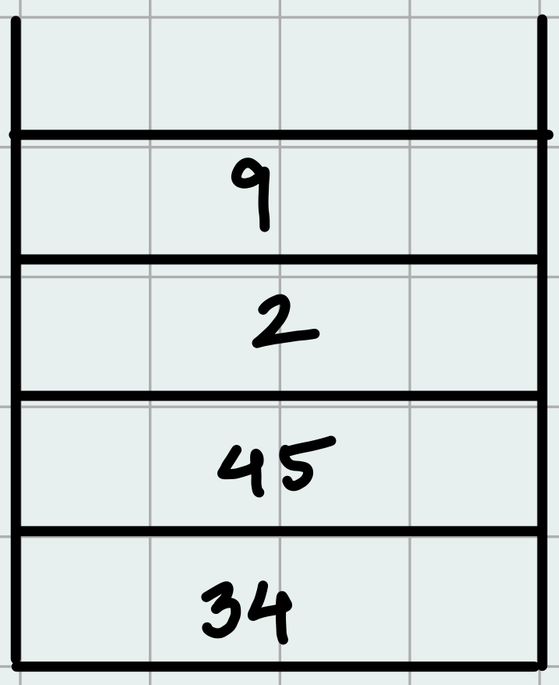
Real-world example:

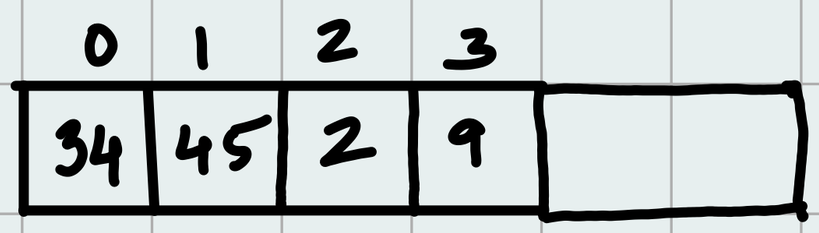
When we go to a buffet plates are stacked on top of each other Last plate to get stacked up is the first one getting removed While first plate to stack up is the last one to get removed

The removal of an object from stack is known as **Pop.**

The insertion of an object is known as **Push.**



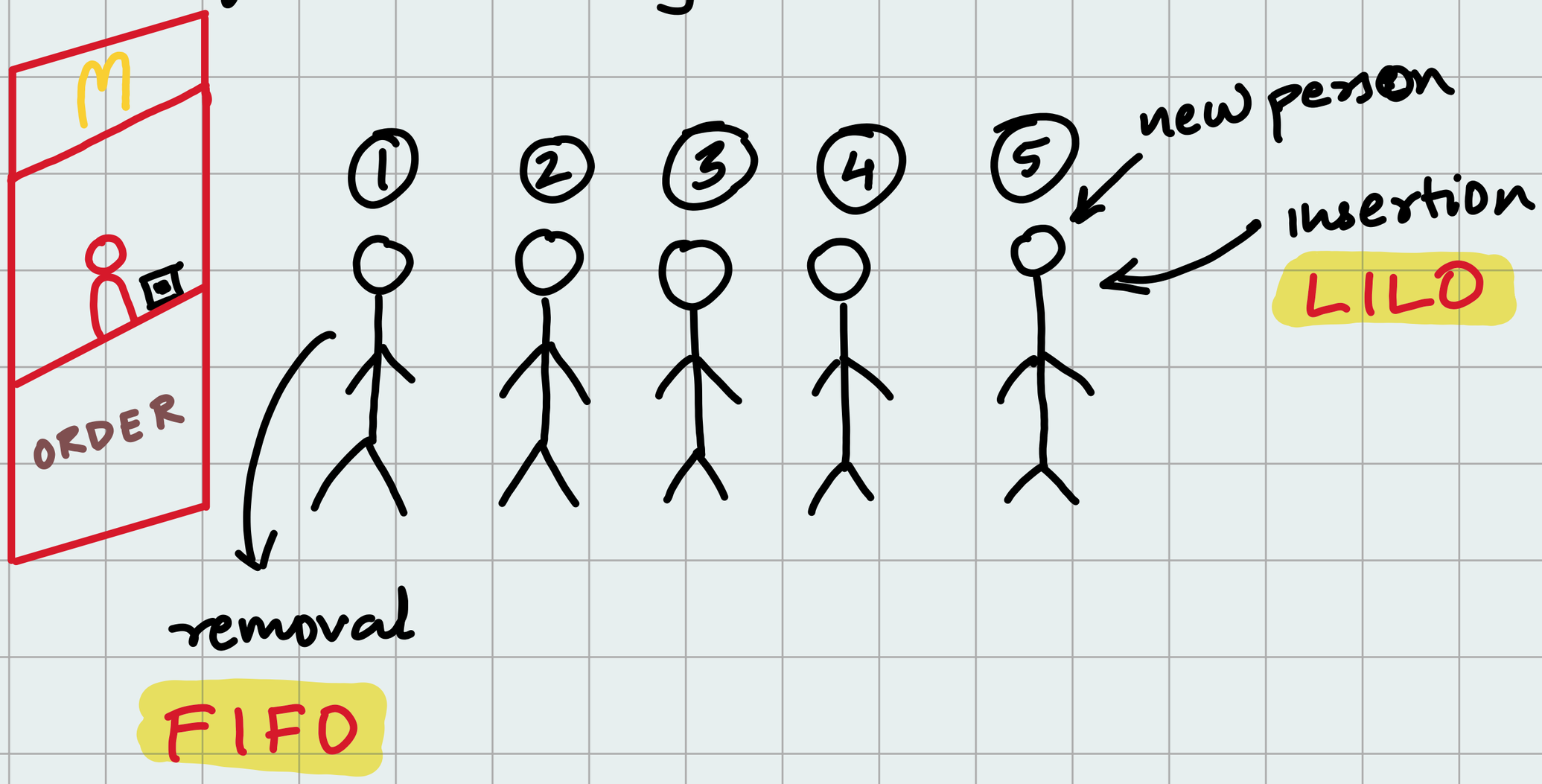




Time complexity Insertion → constant
removal → constant

Queues:

Just like real life Queues When we go to a restaurant There is queue for ordering food



FIFO: First In First Out.

First Object to get In will get out first

LILO: Last In Last Out.

Last Object to get in will get out Last

→ Stack is a class.

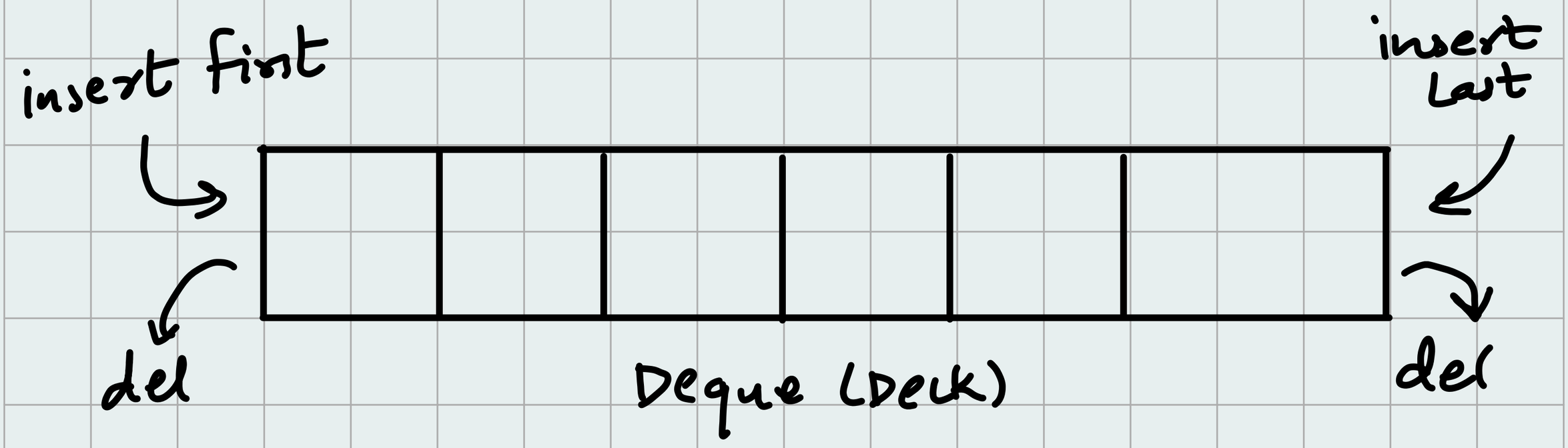
→ Queue is an Interface.

* When do we use Stacks & Queues?

When you want to store answers so far in questions

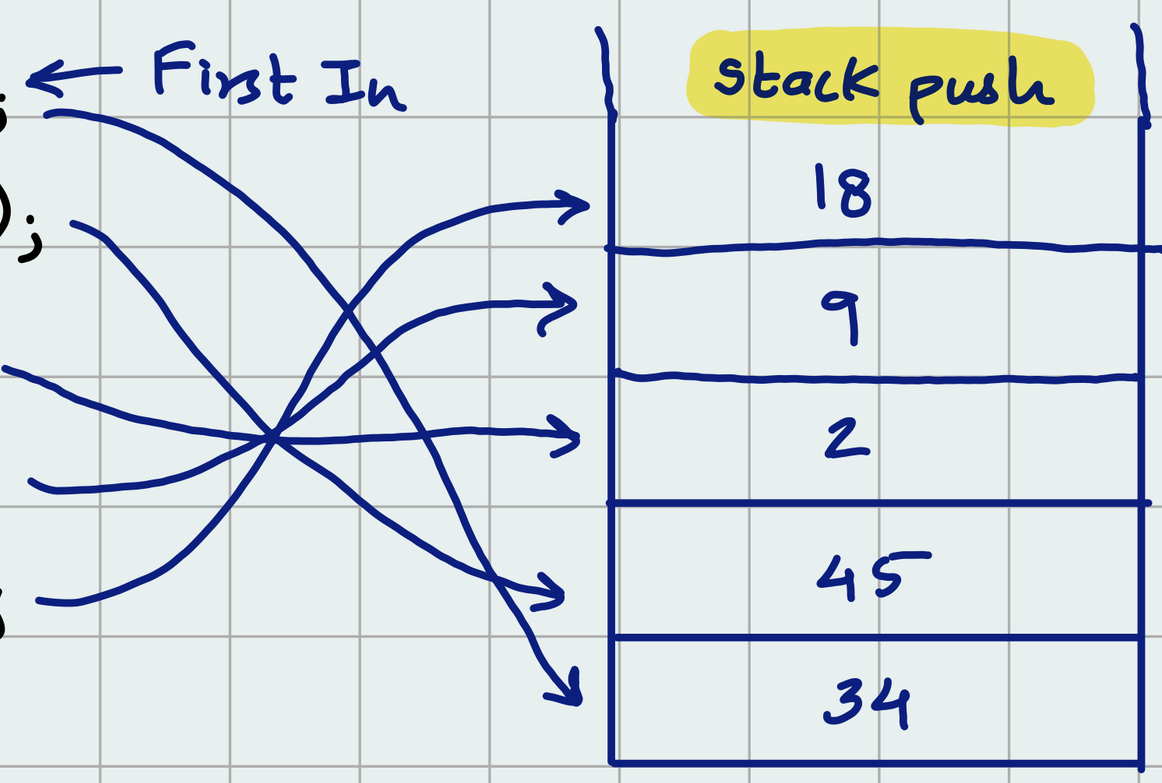
Deque. (Deck)

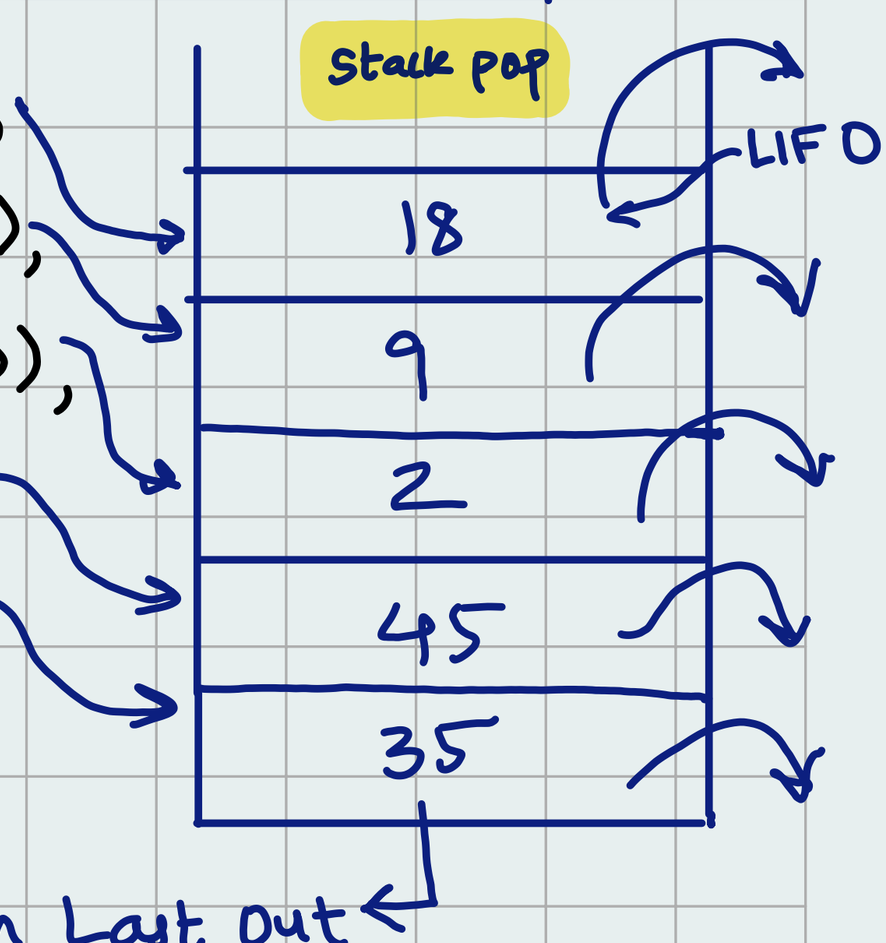
A Queue in which You can insert and remove from both side



* Code for Inbuilt stack

```
public class InbuiltStack {
    public static void main (String [] args) {
        Stack<Integer> stack = new stack<>().
        stack.push (34);
        stack.push (45);
        stack push (2);
        Stack.push (9),
        Stack.push (18);

        System.out println (stack pop()),
        System.out println (stack.pop()),
        System out println (Stack pop()),
        System.out println (stack pop()),
        System.out println (stack pop()),
    }
}
```





Output:

18
9
2
45
35

Time complexity constant → O(1)

★ Code for Inbuilt Queue (Interface of LinkedList).

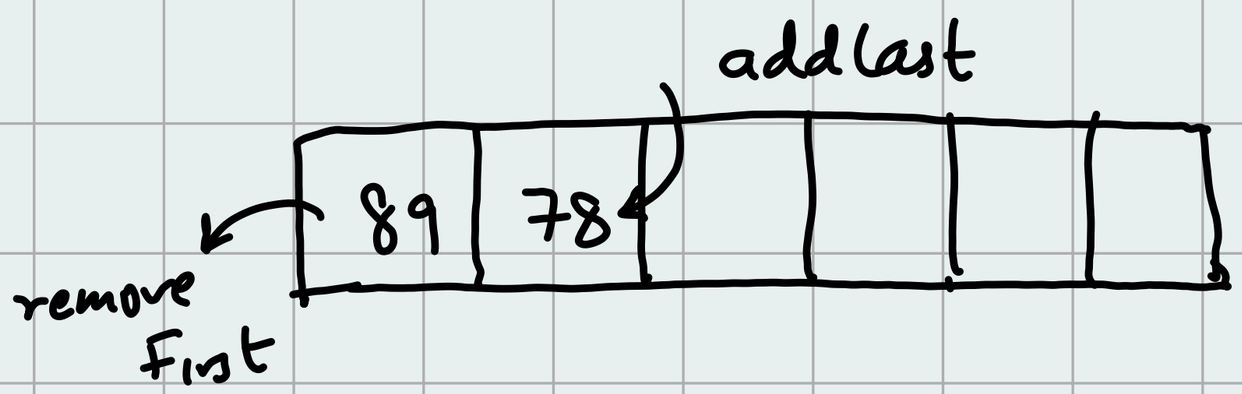
```
public class InbuiltQueue {
    public static void main (String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(3);
        queue.add(6);
        queue.add(5);
        queue.add(19);
        queue.add(1)

        System.out.println(queue.remove());
        System.out.println(queue.remove());
    }
}
```

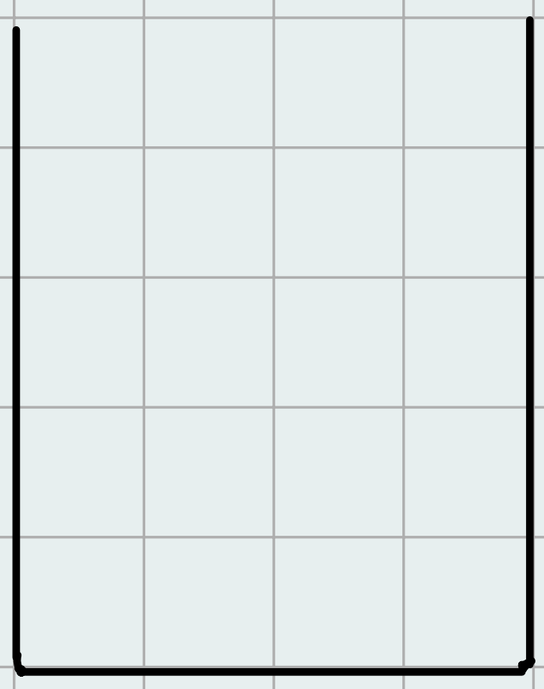
Output:

```
3
6
```

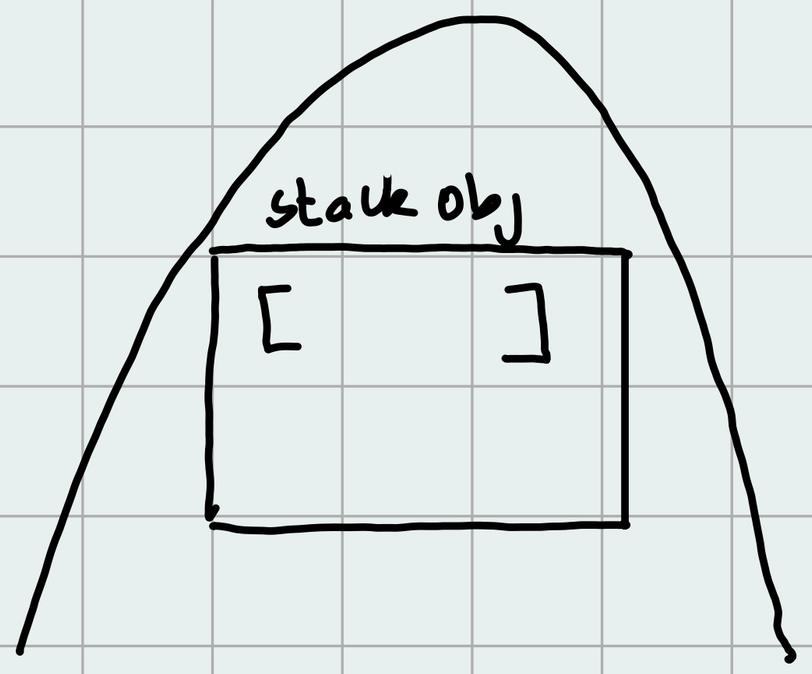
★ Array Deque

```
public class ArrayDeque {
    Public static void main (String [] args) {
        Deque<Integer> deque = new ArrayDeque (),
        deque add (89),
        deque add Last (78),
        deque removeFirst ();
    }
}
```





custom Stack



ptr ~~-1~~ 0

Initially. [0 0 0 0 0 0 0]
0 1 2 3 4 5 6

[18, 37

ptr	
0	18
1	37

★ Custom Stack Implementation:

CustomStack.java

```
public class CustomStack {
    protected int[] data;
    private static final int DEFAULT_SIZE = 10;
    int ptr = -1;
    public CustomStack() {
        this(DEFAULT_SIZE)
    }
// Creating a constructor that takes in size.
    public CustomStack(int size) {
        this.data = new int[size];
    }

    public boolean push(int item) {
        if(isFull()) {
            System.out.println("Stack full!");
            return false;
        }
        ptr++;
        data[ptr] = item;
        return true;
    }

    public int pop() throws StackException {
        if (isEmpty()) {
            throw new Sta Exception("Cannot pop from an empty
                                                    stack");
    }
```

```
        int removed = data[ptr],
        ptr--,
        return removed,
}

    public int peek()  throws  StackException {
        if (isEmpty()){
    throw new StackException ("Cannot peek from an empty
                                        stack!"),
        }
        return data[ptr],
}

    public boolean isFull() {
            return ptr == data.length - 1; //ptr at last index
}

    public boolean isEmpty() {
            return ptr == -1,
        }
}
```

StackException.java

```
public class StackException extends Exception {
    public StackException(String message) {
        super(message),
    }
}
```

StackMain.java

```
public class StackMain {
public static void main (String[] args) throws StackException {
        CustomStack stack = new CustomStack(5),
        stack.push(34);
        stack.push(45),
        stack.push(2),
        stack.push(9),
        stack.push(18),

        System.out.println(stack.pop()),
        System.out.println(Stack.pop()),
        System.out.println(Stack.pop()),
        System.out.println(stack.pop()),
        System.out.println(Stack.pop()),
        System.out.println(stack.pop()),
    }
}
```

* Output

18
9
2
45
34

Dynamic Stack java

```
public class DynamicStack extends CustomStack {
  public DynamicStack() {
      super(), // it will call CustomStack()
}
  public DynamicStack(int size) {
      super(size), // it will call CustomStack (int size)
}
@Override
public boolean push (int item) {
    if (this. isFull()) { // this take care of it being full
       // double the array size
    int [] temp = new int [data length * 2],

  // copy all previous items in new data
 for (int i = 0, i< data length ; i++) {
     temp[i] = data[i],
}
```

```
        data = temp,
    } // at this point we know Array is not full
     // insert item
     return super push(item),
  }
}
```

StackMain.java

```
public class StackMain {
    public static void main (String[] args) throws StackException{
        DynamicStack stack = new DynamicStack (5);
        stack push (34),
        stack push (45),
        stack push (2),
        stack push (9),
        stack push (18);
        stack push (89),

         System out println (stack pop()),
        System.out println (Stack pop()),
         System out println (stack pop()),
        System.out println (Stack pop()),
        System out println (stack pop()),
    }
}
```

data = [2, 9, 1, 8, 7]

temp = [, , , , , 0, 0, 0, 0, 0, 0]

temp = [2, 9, 1, 8, 7, 19. 0, 0, 0, 0, 0]

★ Custom Queue

```
public class CustomQueue {
    pnvate int [ ] data,
    private static final int DEFAULT_SIZE=10,
    int end = 0;
    public CustomQueue() {
       this(DEFAULT_SIZE),
    }
   public CustomQueue( int size) {
      this data = new int [size],
}
  public boolean isFull() {
     return end == data length,
  }
  public boolean isEmpty () {
    return end == 0;
}
  public boolean insert (int item) {
      if (isFull()) {
        return false,
      }
      data [end++] = item;
      return true;
  }
```

* Custom Queue Implementation:

data = [0, 0, 0, 0, 0, 0]
(0 1 2 3 4 5)

end

[34, 45, 0, 0, 0]

~~0~~ 34

~~1~~ 45

2

Removal of an item from Queue

data = [3, 9, 4, 18, 77]
(0 1 2 3 4)

FIFO ↓

data = [9, 4, 18, 77, 0]

O(N)

```
public int remove() throws Exception {
    if (isEmpty()) {
        throw new Exception("Queue is Empty!"),
    }
    int removed = data[0],
// shift elements to left.
for (int i = 1; i < end; i++) {
        data[i-1] = data[i],
    }
end--;
return removed;
```

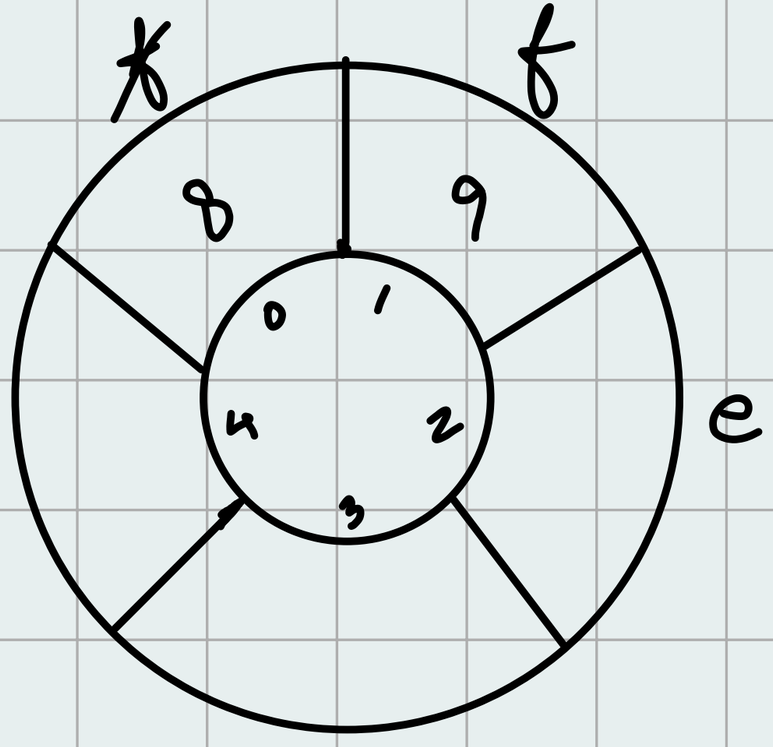
```
    }

    public int front() throws Exception {
        if (isEmpty()) {
            throw new Exception("Queue is empty")
        }
        return data[0];
    }
}
```

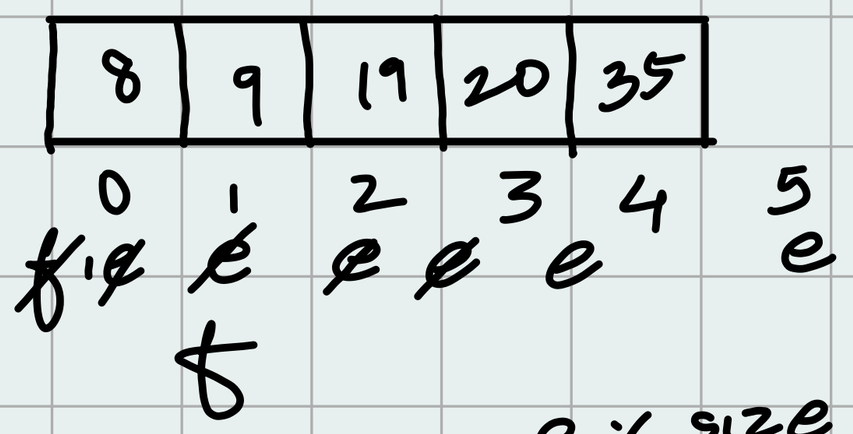
```
public void display() {
    for (int i=0; i<end, i++) {
        System.out.println(data[i] + "<-");
    }
    System.out println("END");
}
```

★ Queue Main java.

```
public class QueueMain {
  public static void main (String[] args) {
CustomQueue queue = new CustomQueue (5),
  queue insert ( 3);
   queue insert ( 6 );
   queue insert ( 5 );
   queue insert ( 19 );
  queue insert ( 1 );

  queue.display (),

  System.out println (queue remove ()),

  queue display (),

 }
}
```

★ Circular Queue Implementation





e % size

5 % 5 = 0

6 % 5 = 1

```
public class CircularQueue {
    protected int [] data,
    private static final int DEFAULT_SIZE=10,
    protected int end = 0;
    protected int front =  0,
    private int size = 0,


 public  CircularQueue() {
     this(DEFAULT_SIZE),
 }

    public CircularQueue(int size) {
        this data = new int [size],
}

  public boolean isFull() {
    return size == data length,
}
```

```
public boolean isEmpty() {
    return size == 0;
}

public boolean insert(int item) {
    if (isFull()) {
        return false,
    }
    data[end++] = item,
    end = end % data.length;
    size++,
    return true;
}

public int remove() throw Exception {
    if (isEmpty()) {
        throw new Exception("Queue is empty");
    }
    int removed = data[front];
    front = front % data.length,
    size--;
    return removed,
}

public int front() throws Exception {
    if (isEmpty()) {
        throw new Exception("Queue is empty"),
    }
    return data[front],
}
```

```
    public void display () {
      if (isEmpty()){
         System.out.println("Empty"),
         return,
    }
    int i = front,
do {
    System.out.println(data[i] + "->");
    i++;
    i %= data.length;
} while (i != end),
 System.out.println("END"),
}
}
```

★ Queue Main java.

```
public class QueueMain {
  public static void main (String[] args) {
 CircularQueue queue = new CircularQueue(5),
  queue insert ( 3);
  queue insert ( 6 );
  queue insert ( 5 );
  queue insert ( 19 );
 queue insert ( 1 );


 queue.display (),

  System.out println(queue remove()),
 queue insert (133),
 queue display(),

  }
}
```

★ Dynamic Queue.

```
public class DynamicQueue extends CircularQueue {
    public DynamicQueue() {
        super();
    }

    public DynamicQueue(int size) {
        super(size);
    }

    @override
    public boolean insert(int item) {
        if (this.isFull()) {
            int[] temp = new int[data.length * 2];
            for (int i=0; i< data.length; i++) {
                temp[i] = data[(front+i) % data.length];
            }
            front = 0;
            end = data.length;
            data = temp;
        }
        return super.insert(item);
    }
```

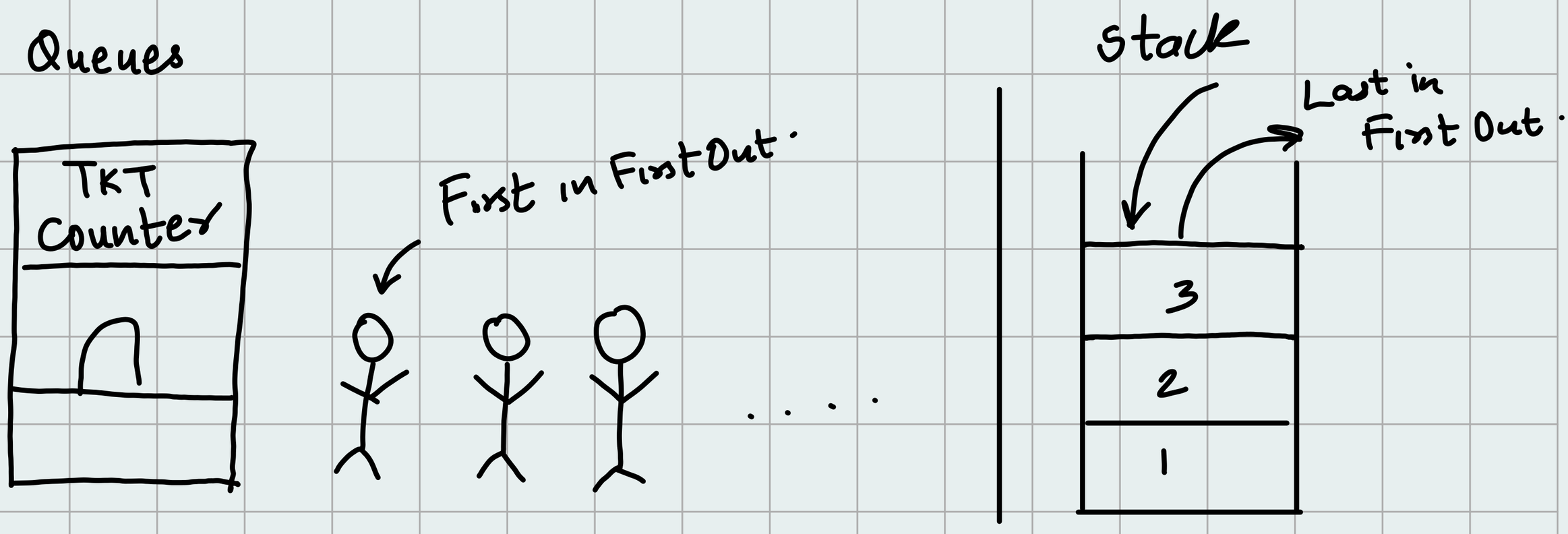
★ FAANG Questions:

In which scenario would you get the hint to use stacks & Queues.

- If you are dealing with ordering, If you are putting stuff in order Then we are going to use stacks & Queues

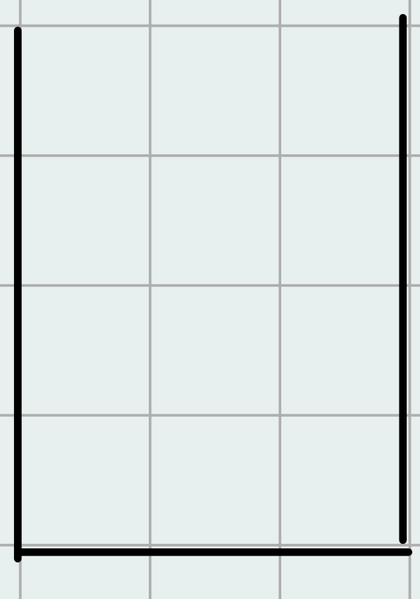
LeetCode · 232 Implement Queue using Stack.

Implement a first in first out (FIFO) queue using only two stacks The implemented queue should support all functions of a normal queue (push, peek, pop, and empty)

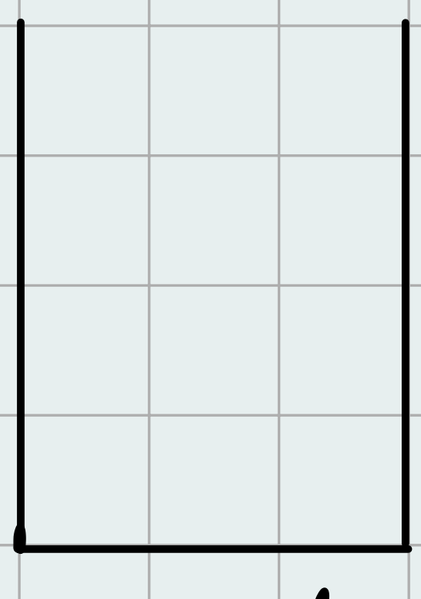


Given Stacks = 2 nos

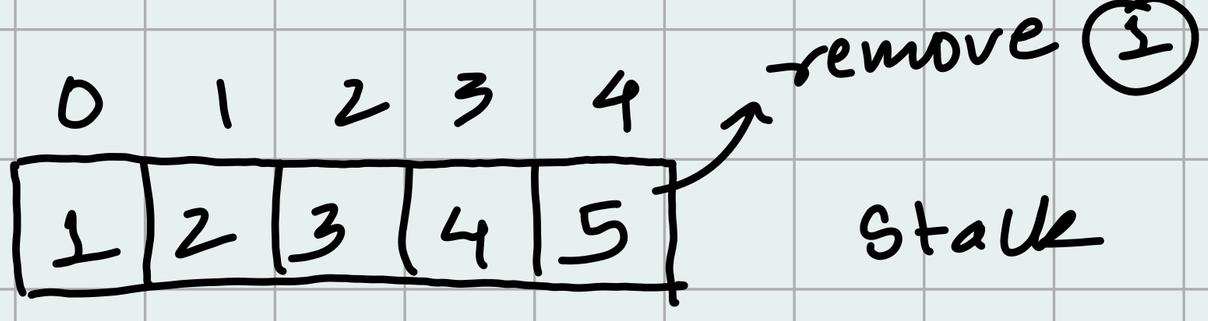
FILO

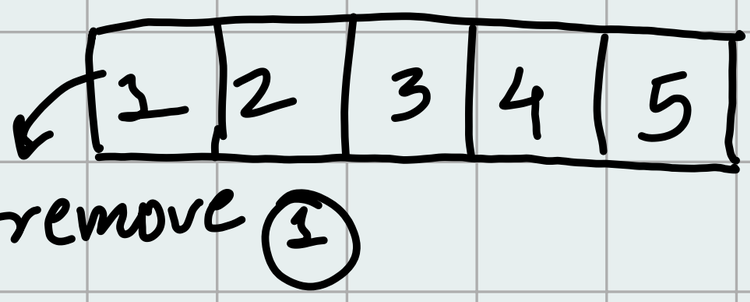


first stack (main)



second. stack (helper)

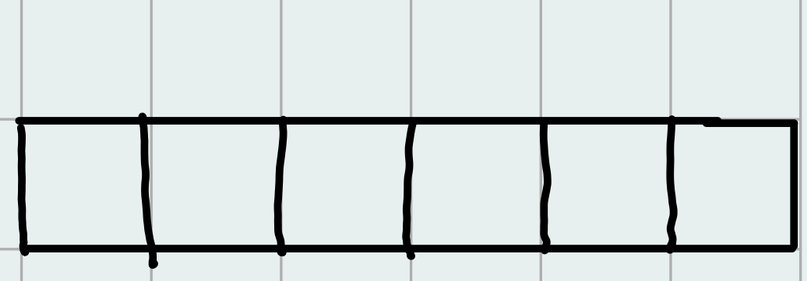




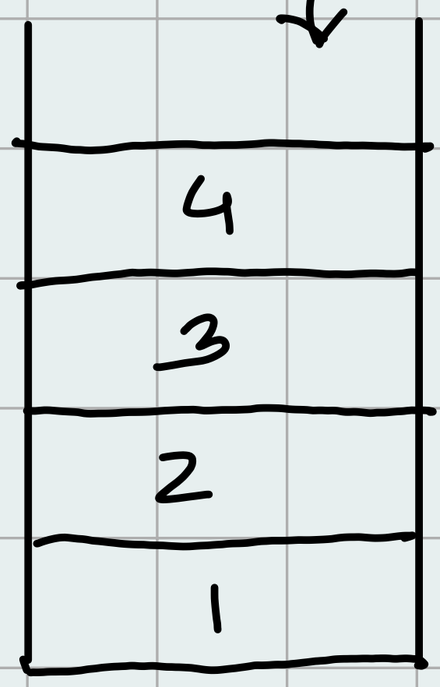
Queue.

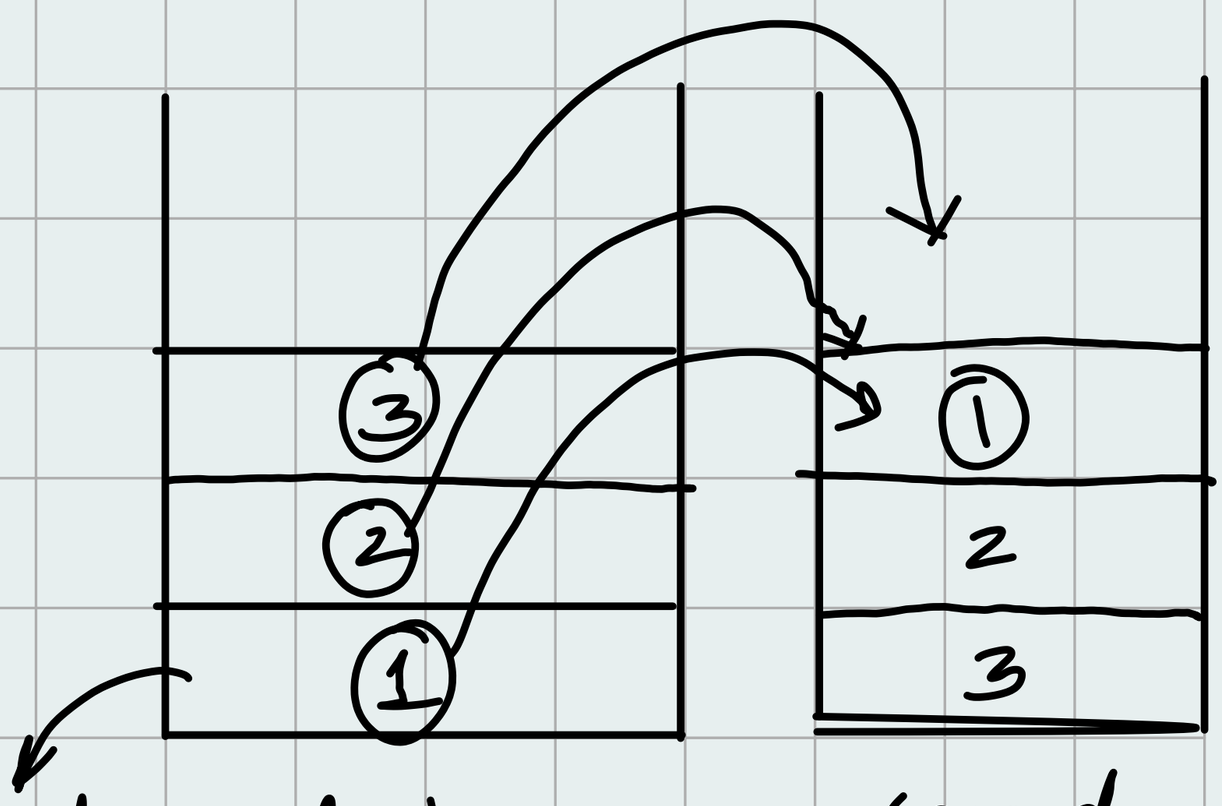
```
add () {
    insert in first,
}
```

Queue.



FIFO



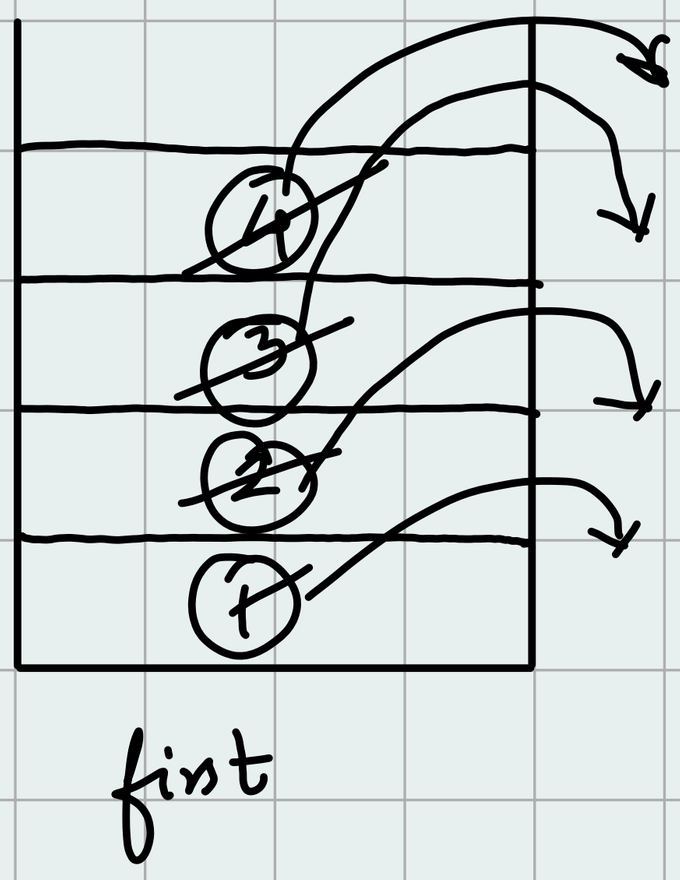


we need to remove first but due to FILO this second

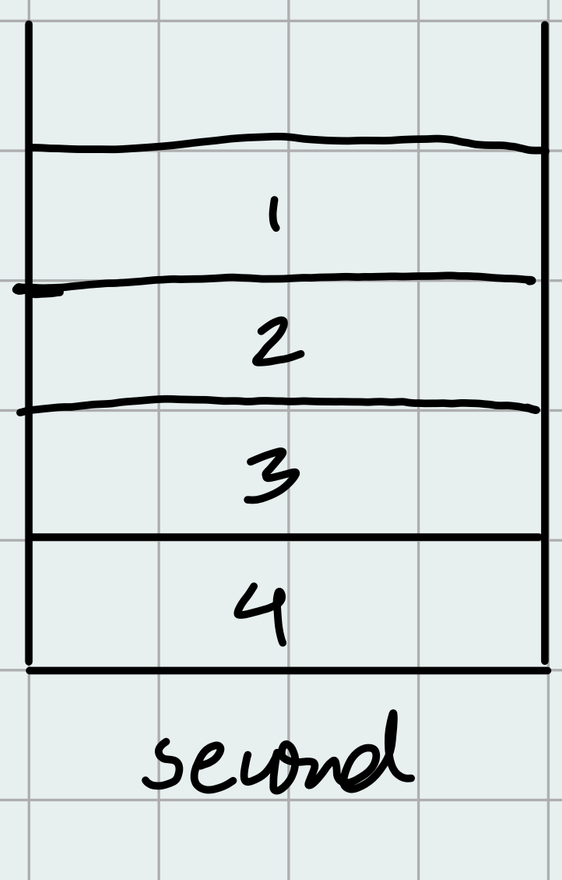
you need to remove above elements

first stack pop () → add second

second pop ()

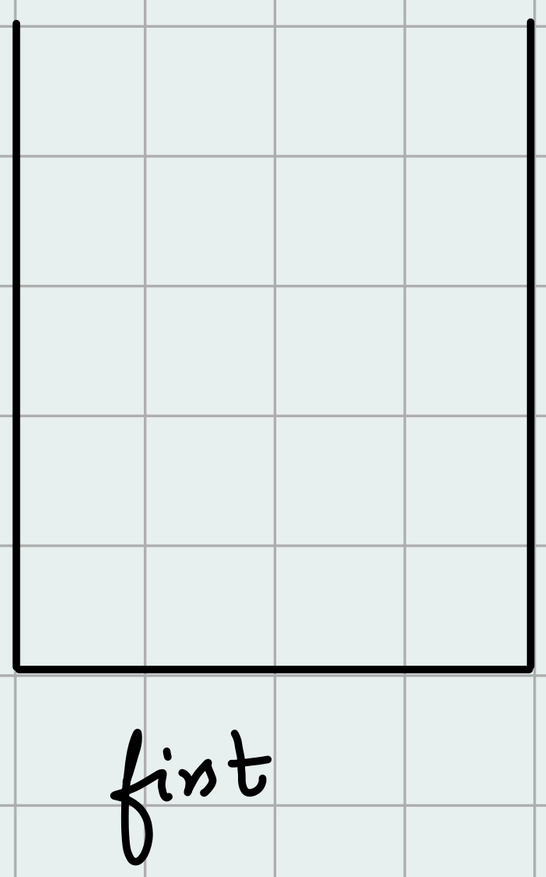


first

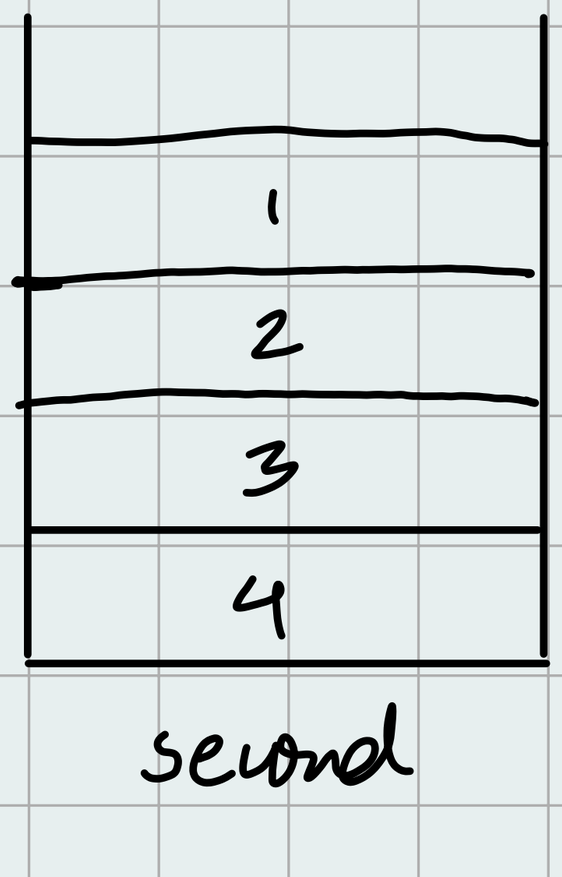


second

After this step first will be empty



first



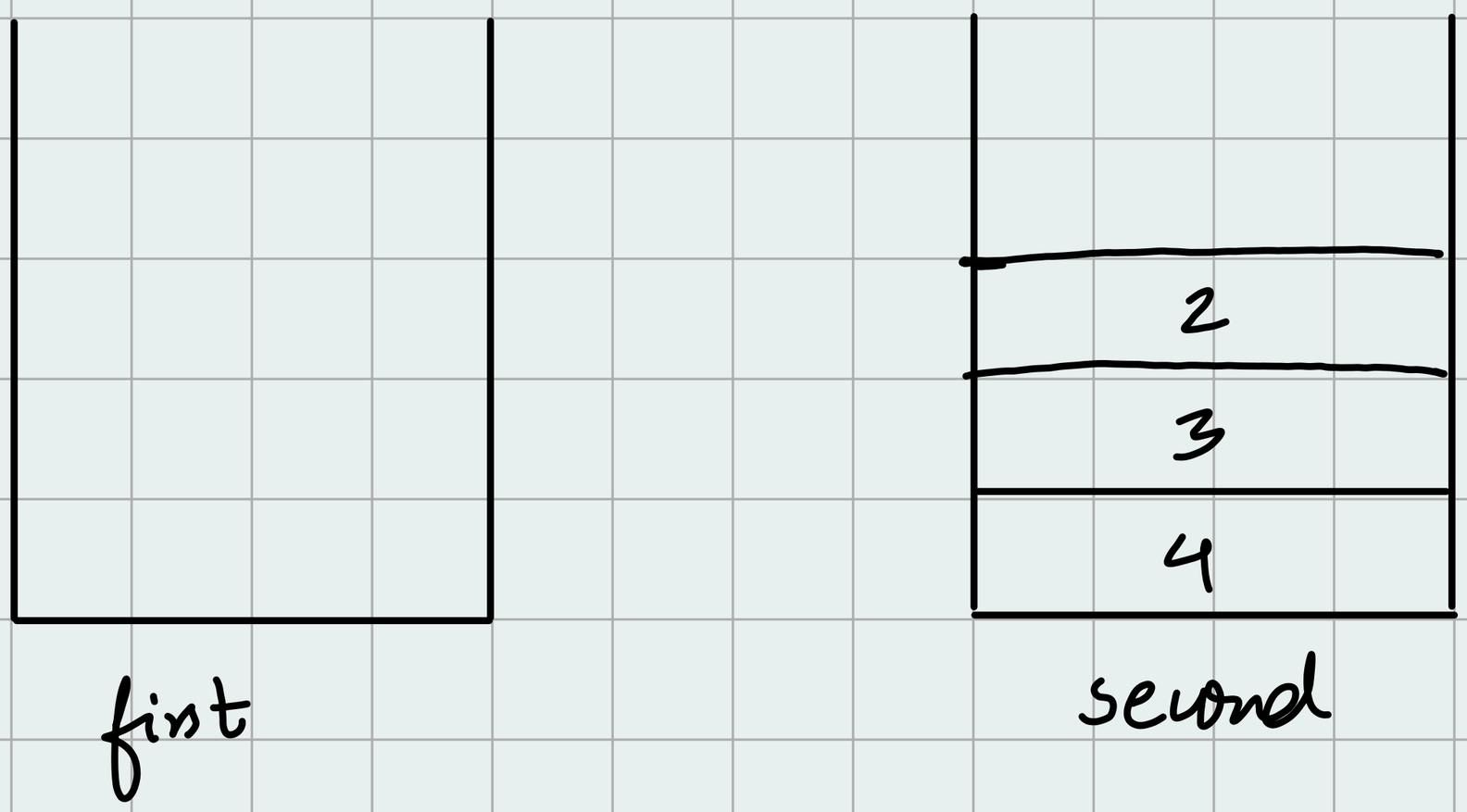
second

second pop() will remove 1

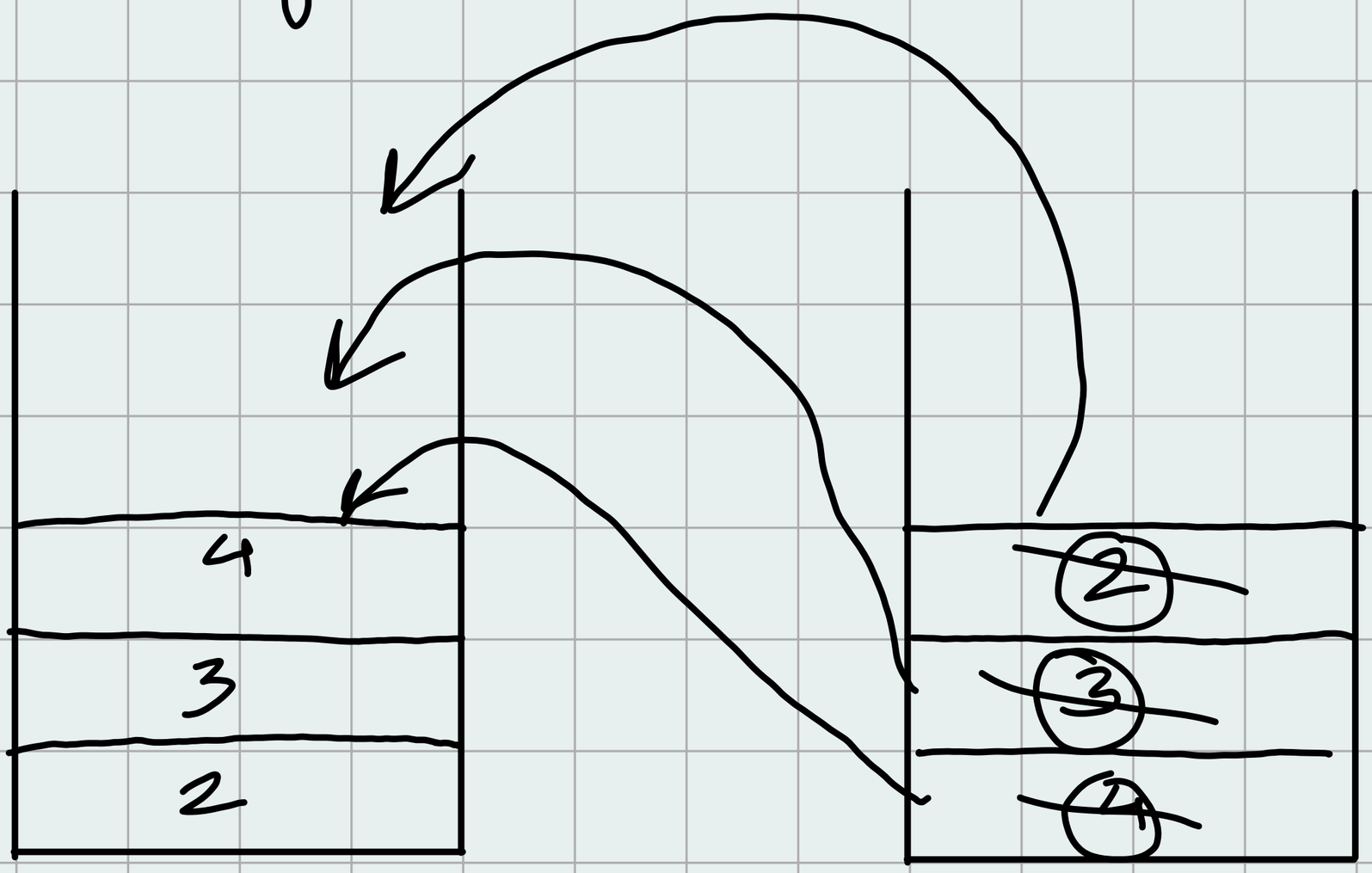
first

second

Now we are left with



Now we need to empty the second stack and put it back in first stack

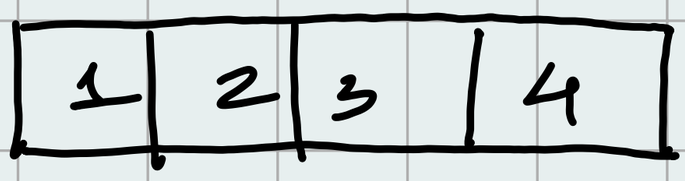


Now these two stacks believe like a queue.

But it is not efficient

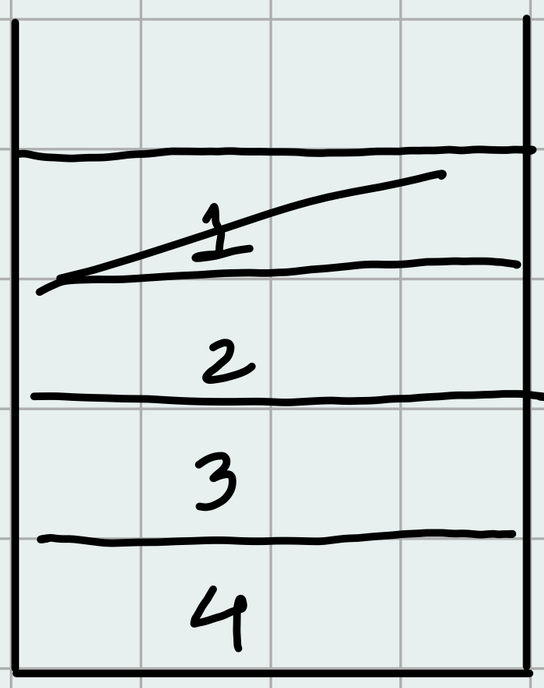
- For insertion the complexity is constant which means it is efficient
- For removal of Item you need to traverse the whole stack hence complexity is O(N)

Hence it is Known as insert efficient Queue

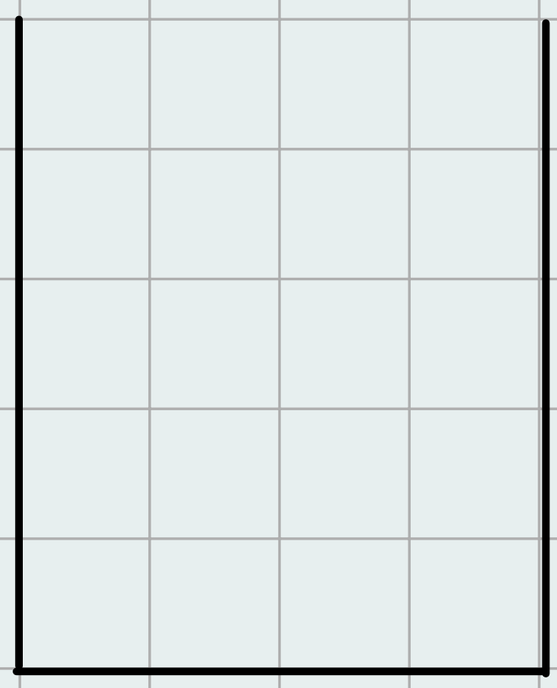


← to make remove efficient we need to put the item to be removed at top

* Remove Efficient.

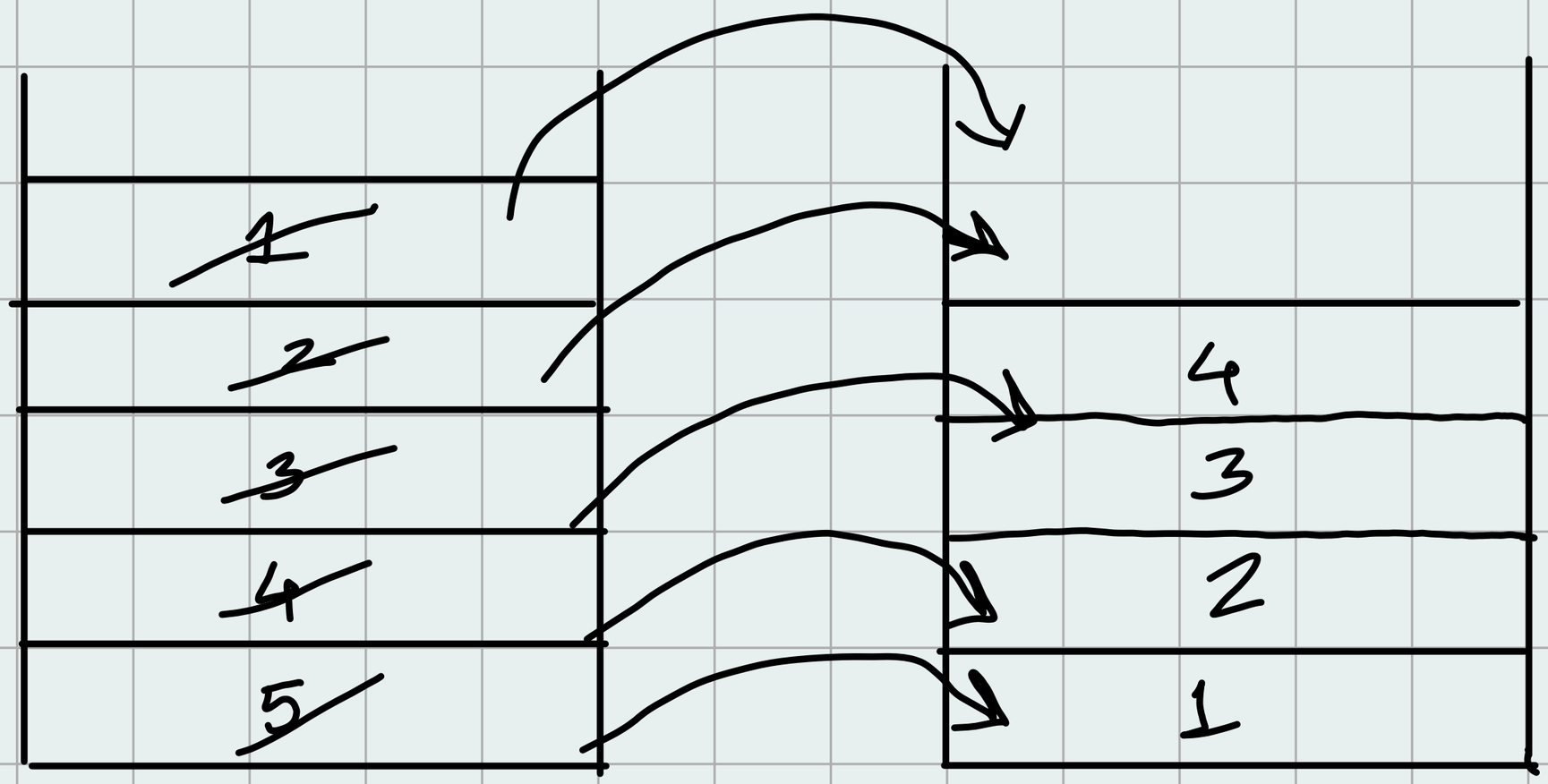


first



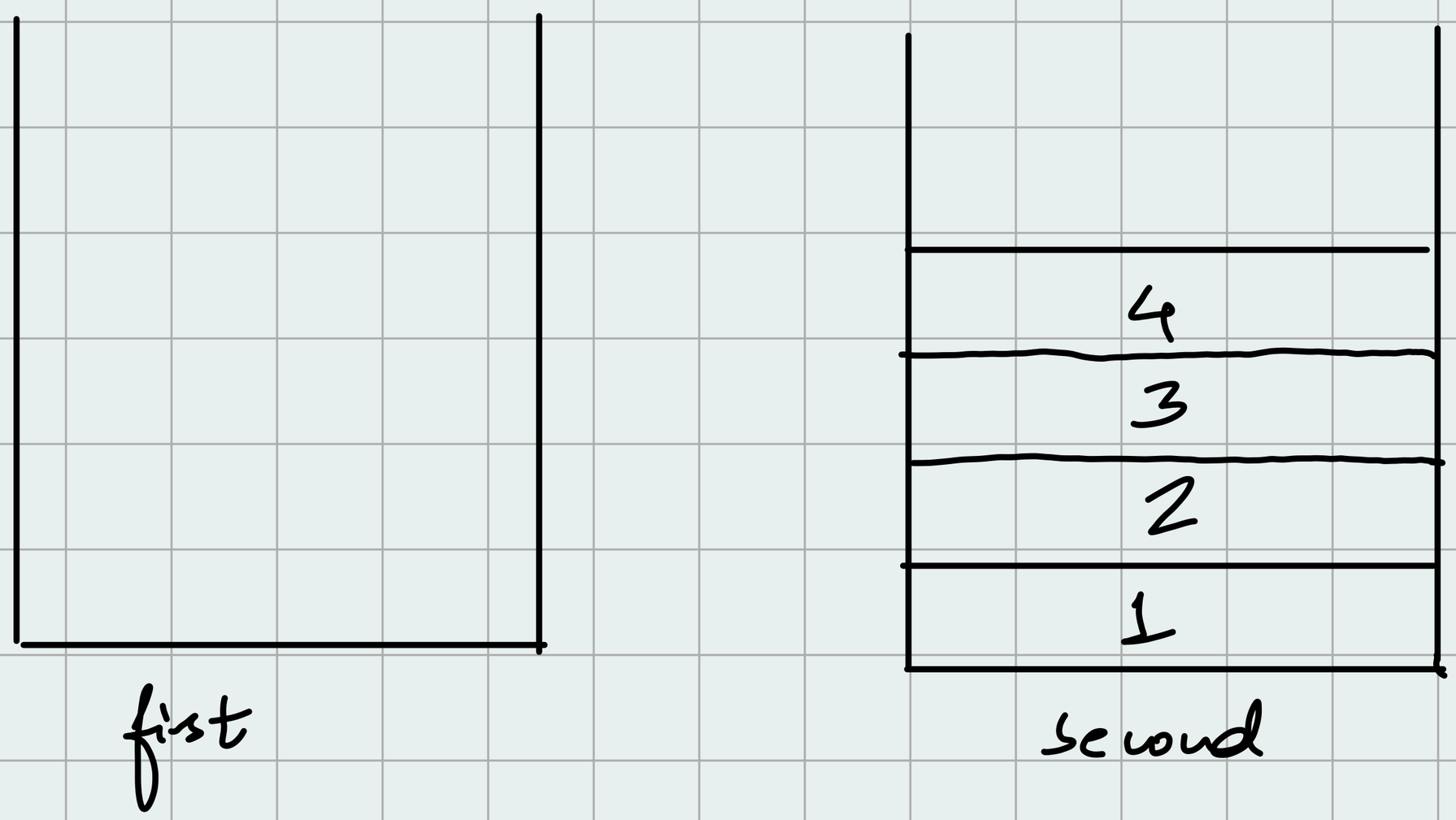
second

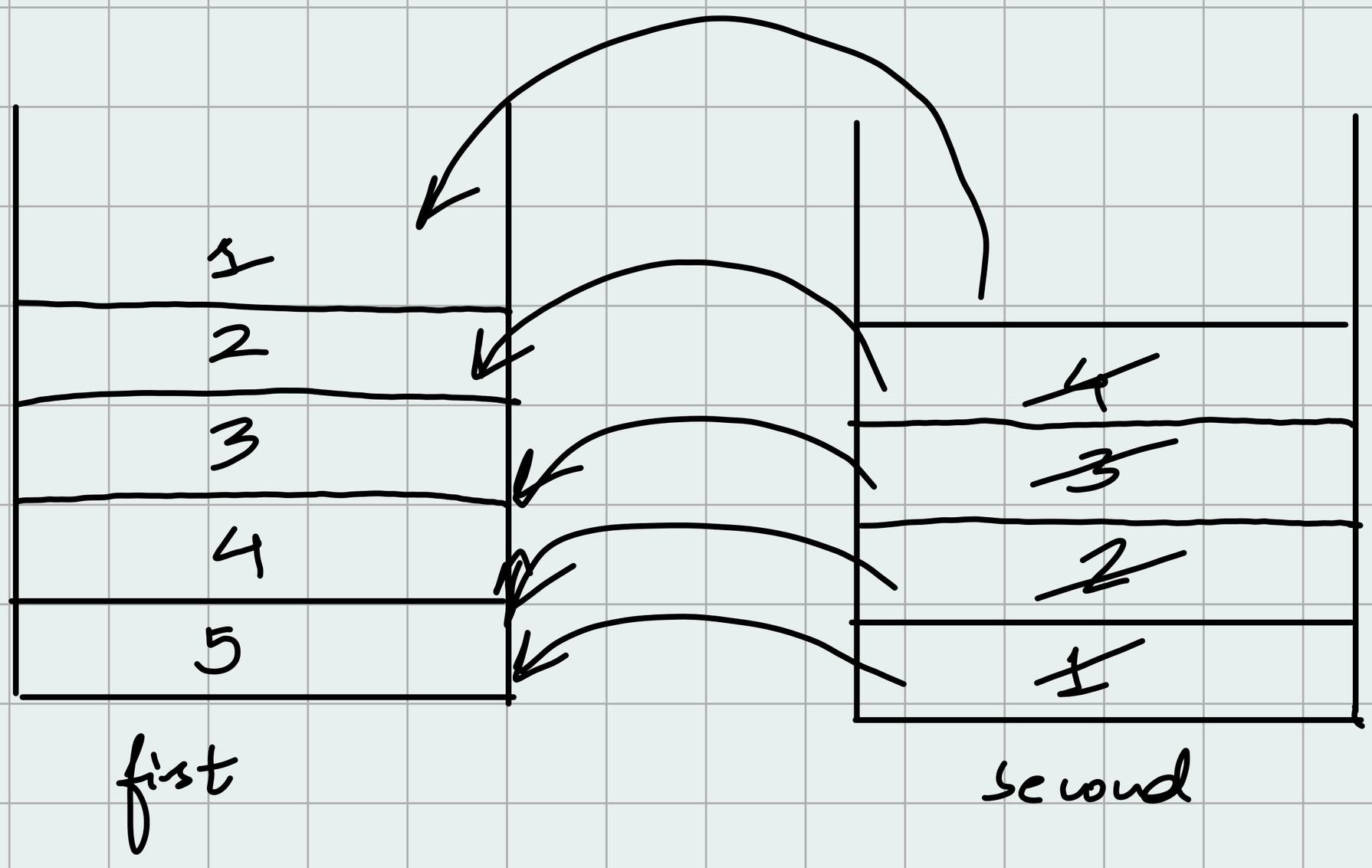
But now question is when you want to insert you have to insert from back lets say 5



first

second





Now removal is O(1) and insertion is O(N)

O(1) // removal

O(N) // insert

```
class Main {
    public static void main (String[] args) {


    }
    class QueueUsingStack {
private Stack<Integer> first,
private Stack<Integer> second,

public QueueUsingStack () {
    first = new Stack();
    second = new Stack();
}
public void add (int item) {
    first push (item),
}
public int remove() throws Exception {
    while (! first isEmpty()) {
        second push (first pop()),
   }
    int removed = second pop(),
    while (! second isEmpty()){
     first.push (second.pop()),
  }
  return removed,
}
```

```
public int peeK() throws Exception {
    while (!first.isEmpty()) {
        second.push(first.pop()),
    }

int removed = second.pop(),
while (!second.isEmpty()){
    first.push(second.pop()),
}

return peeked,

  public boolean isEmpty() {
      return first.isEmpty();
    }
}
```

★ Remove Effuent

```
class QueueUsingStack {
    private Stack<Integer> first,
    private Stack<Integer> second,

    public QueueUsingStack() {
        first = new Stack();
        second = new Stack();
    }
public void add (int item) {
  while (! first isEmpty()) {
        second push (first pop()),
}
    first push (item);
    while (! second . isEmpty()) {
        first push (second pop()),
  }
}

    public int remove() throws Exception {
        return first pop(),
    }
```

```
public int peek() throws Exception{
    return first peek(),
 }

 public boolean isEmpty() {
    return first.isEmpty();
 }
}
```

★ Game of two Stacks Medium

Alexa has two stacks of non negative integers, stack a[n] and stack b[m] where index 0 denotes the top of the stack Alexa challenges Nick to play following game

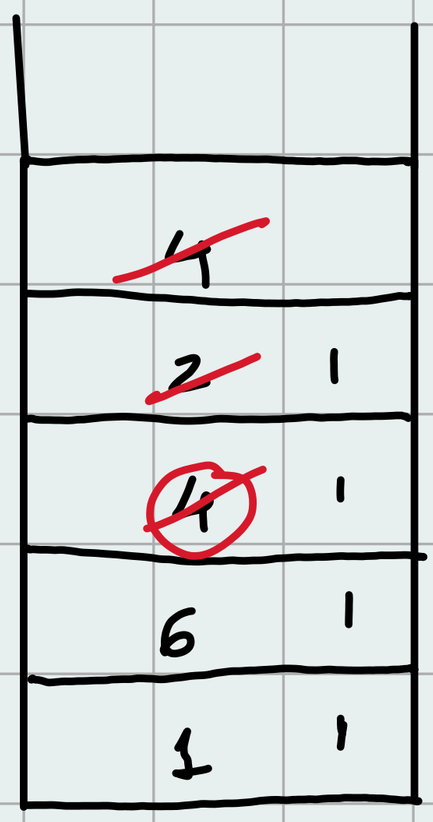
a) In each move, Nick can remove one integer from the top of either stack a or stack b

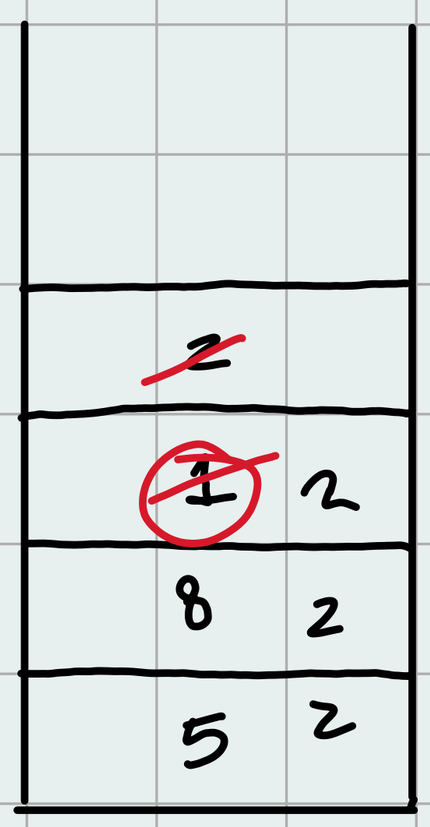
b) Nick keeps a running sum of integers he removes from the two stacks

c) Nick is disqualified from the game if at any point, his running sum becomes greater than some integer maxSum given at the beginning of the game

d) Nicks final score is the total number of integers he has removed from the two stacks

Given a, b and maxSum for g games. find the maximum possible score Nick can achieve





maxSum cannot exceed

10

4+2+2+4 = not possible exceed

4+2+2+1 = possible.

4 → Ans

(a, b, s, m)

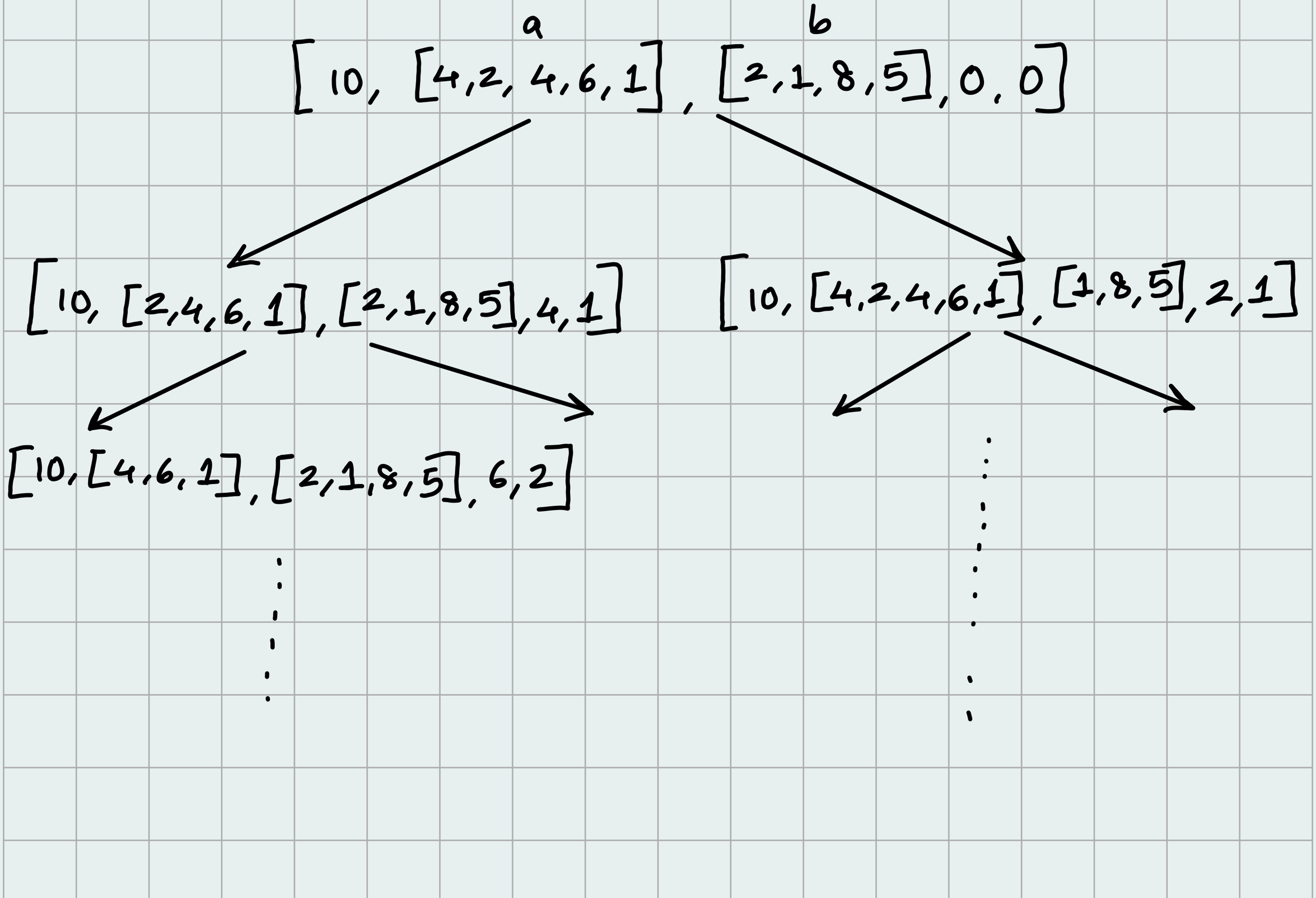
a pop(), b, s+popped,

([4,2,4,6,1], [2,1,8,5], 0, 0)

([4,2,4,6,1], [1,8,5], 2, 1)

([2,4,6,1], [2,1,8,5], 4, 1)

Recursion tree for above question



```
public class TwoStacks {
    static int  twoStacks (int x, int [] a, int [] b){
      return twoStacks( x,a,b,0,0 ) -1;
    }

    private static int twoStacks (int x, int [] a, int [] b,
                                  int sum, int count) {
        if (sum > x) {
        return count,
      }
   if (a length == 0 || b length == 0) {
     return count,
}

 int ans1 = twoStacks (x, Arrays copyOfRange(a,1,a length),
                          b, sum + a[0], count+1),
int ans2 = twoStacks (x, a, Arrays copyOfRange(b,1,b length),
                          sum + a[0], count+1),
 return Math max (ans1, ans2);

}
```

```
public static void main(String[] args) {
  Scanner s = new Scanner(System.in);
  int t = s.nextInt();
  for(int i=0; i<t; i++) {
    int n = s.nextInt();
    int m = s.nextInt();
    int x = s.nextInt();
    int[] a = new int[n];
    int[] b = new int[m];
    for(int j=0; j<n; j++){
      a[j] = s.nextInt();
}

for(int j=0; j<m; j++) {
      b[j] = s.nextInt();
    }

    System.out.println(twoStacks(x, a, b));
  }
 }
}
```

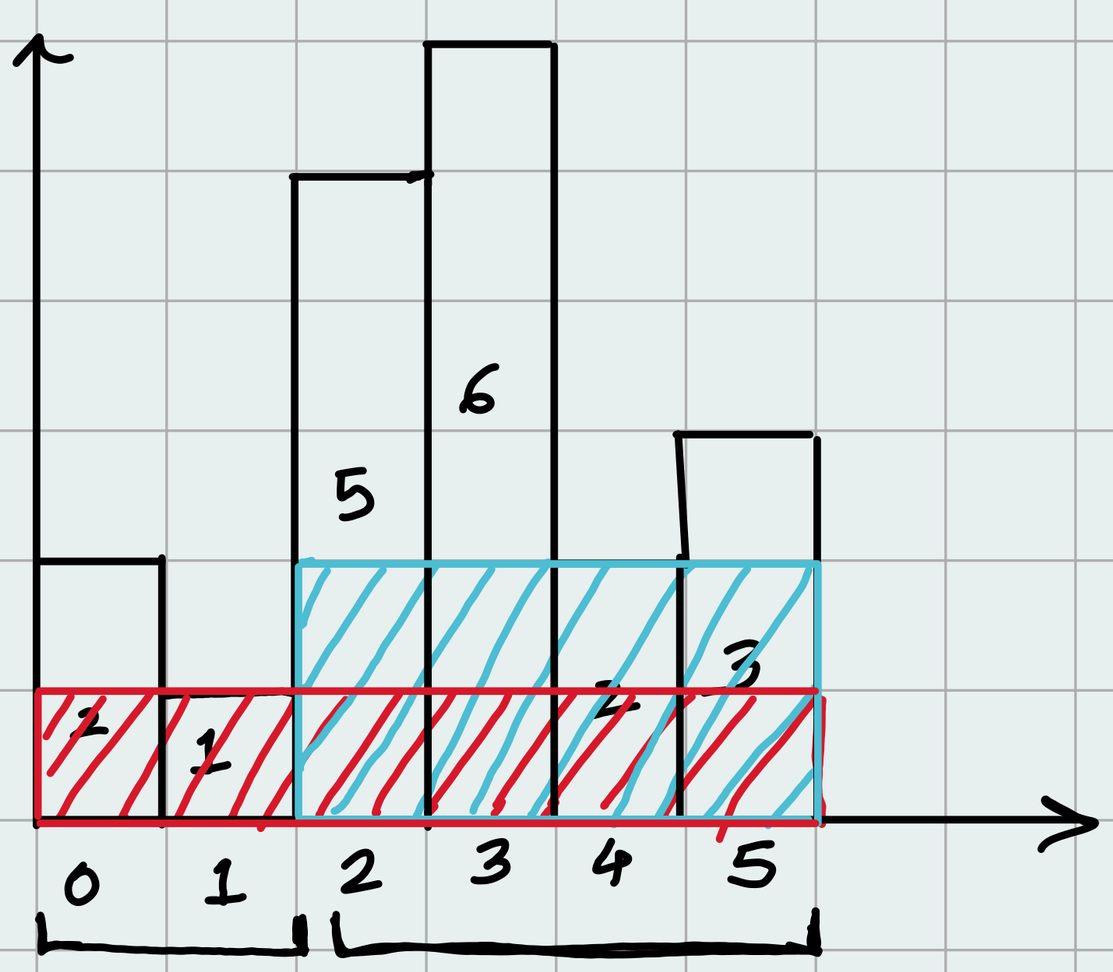
ANSWER SO FAR

↓

STACKS

* Leetcode. 84 Largest Rectangle in Histogram

Given an array of integers heights representing the histogram bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram



Q) Find largest area in histogram

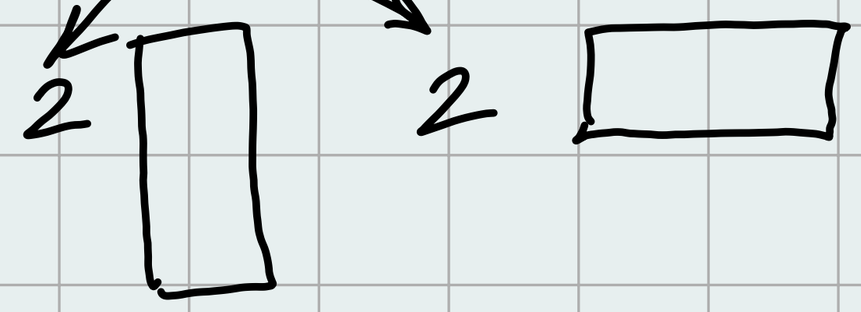
max : 2 5 10

What we need?

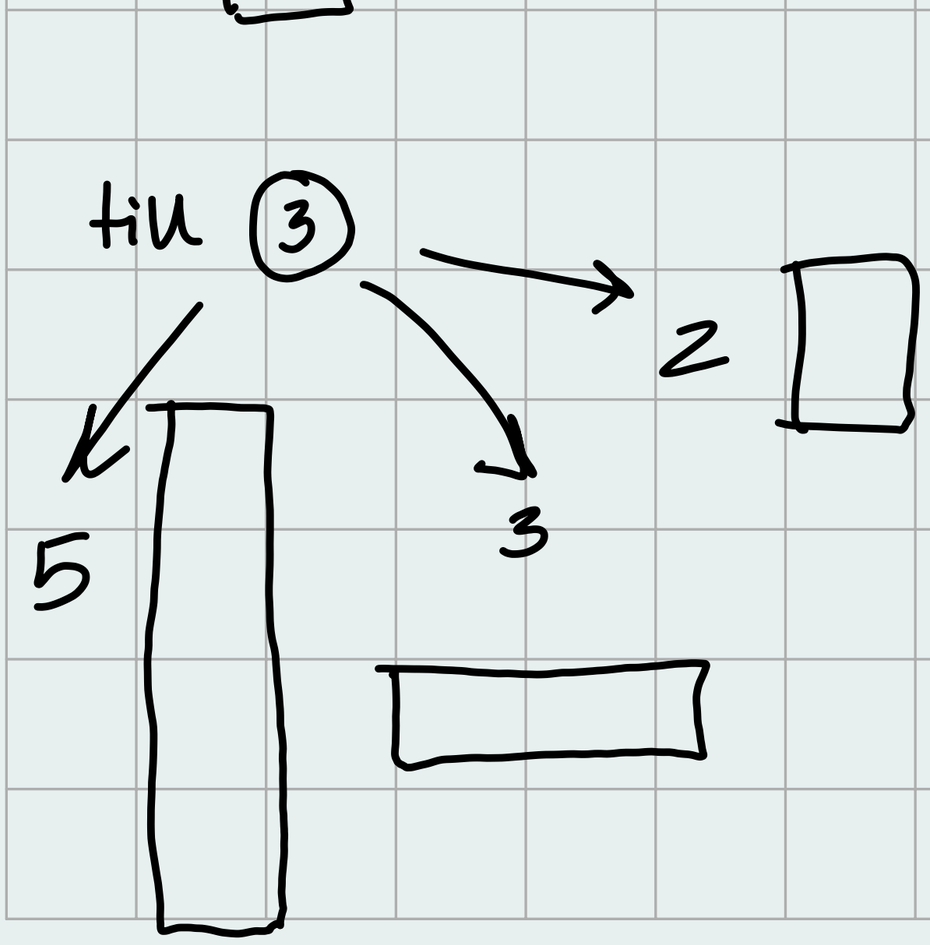
① We need max till each index

till ① → 2

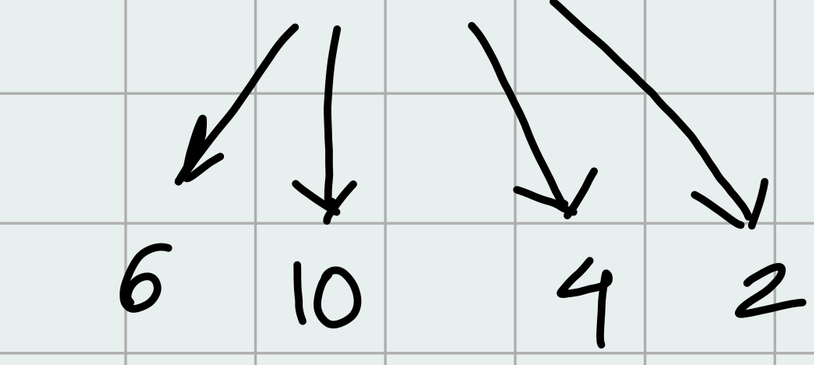
till ②



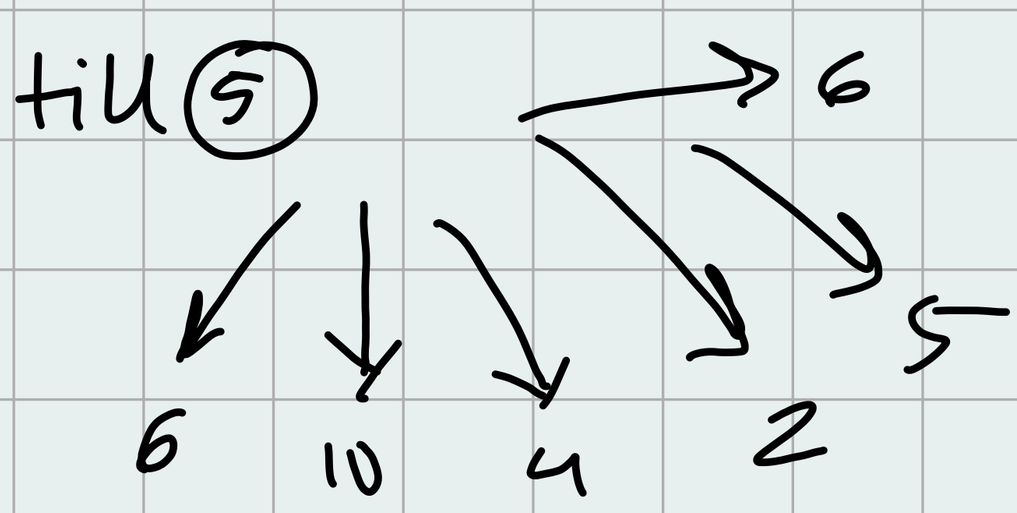
till ③



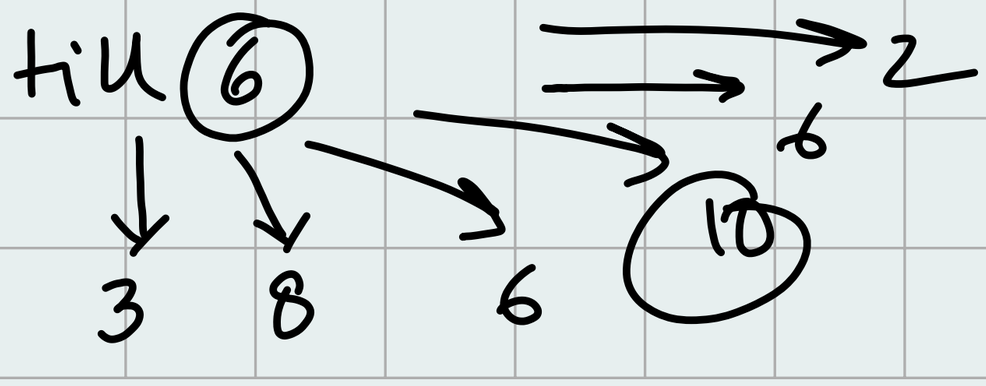
till ④



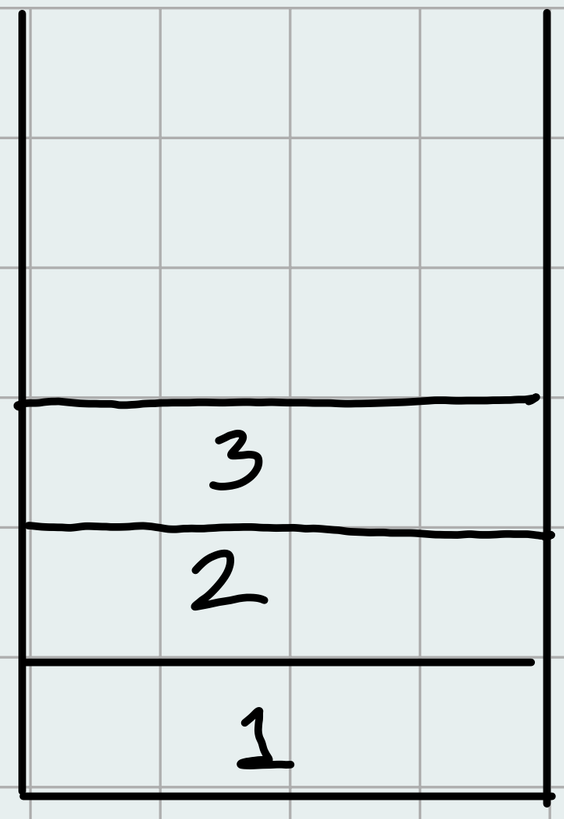
till ⑤



till ⑥



i	max
~~1~~	~~0~~
~~2~~	~~2~~
~~3~~	~~6~~
~~4~~	10
~~5~~ 6	



$6 * (3-2) = 6$

$5 * (3-1) = \underline{\underline{10}}$

$2 * (5-1) = 8$

⊛ LeetCode · 20 Valid Paranthesses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid

An input string is valid if

- Open braces must be closed by same type of braces
- Open braces must be closed in the correct order
- Every closed bracket has a corresponding open bracket of same type.