Queue

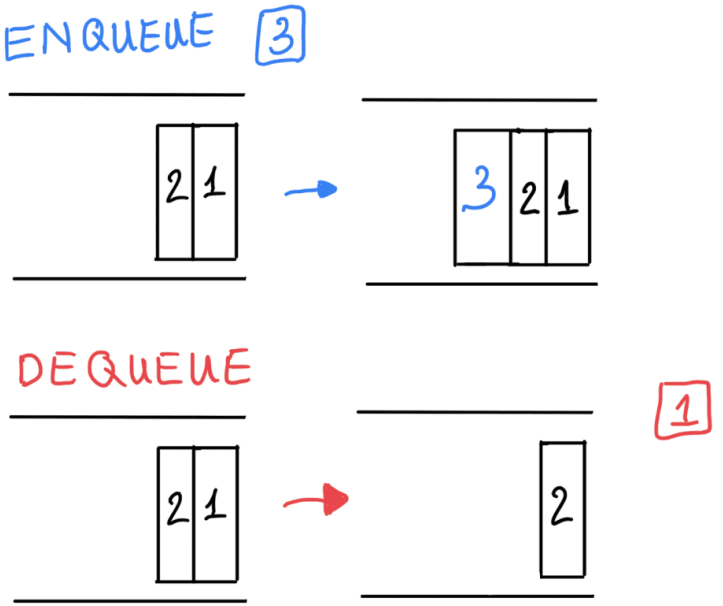
BY

DR. TISHA

COURTESY: DR. MYERS' BOOK

What is a Queue?

- Data Structure
- First in first out (FIFO)
- Queue is an **Interface** in Java
- Example:
  - CPU scheduling
  - Print Queues
  - Network packet scheduling



Functions for Queue

- **offer:** add a new item to the end of the queue. This method may also be called add.
- **poll:** remove and return the next item from the front of the queue. May also be called pop or remove.
- **peek:** return the first item without removing it.

Practice

Let’s try to offer/add some elements to the queue and poll it and also peek it.

```
1  import java.util.Queue;
2
3  class Main {
4    public static void main(String[]
   args) {
5      Queue <String> que = new Queue<>();
6      que.offer("a");
7      que.offer("b");
8
9      System.out.println(que.element());
10
11     que.poll();
12     System.out.println(que.peek());
13   }
14 }
```

Error! Why?

Because..

Queue is an interface not a Class.

We can use LinkedList or PriorityQueue or ArrayDeque class.

Interface?

- Java interface functions like a contract: a class that **implements** a particular interface guarantees to other classes that it will provide certain behaviors.

Sample code for Queue interface and LinkedList class

```
// A simplified version of Queue interface

public interface Queue<E> {

   boolean offer(E e); // abstract method-no body
   E poll();

   E peek();

}
```

```
import java.util.*;

public class LinkedList<E> implements Queue<E> {

   @Override
   public boolean offer(E e) {
      // for LinkedList, same as add
      return add(e);
   }

   @Override
   public E poll() {
      return isEmpty() ? null : super.removeFirst();
   }

    @Override
   public E peek() {
      return isEmpty() ? null : getFirst();
   }
}
```

```java
import java.util.Queue;
import java.util.LinkedList;

class Main {
  public static void main(String[] args) {
    Queue <String> que = new LinkedList<>();
    que.offer("a");
    que.offer("b");

    System.out.println(que.element());

    que.poll();
    System.out.println(que.peek());
  }
}
```

Right code will be

Try it yourself (bonus 10/6)

I added an element to a stack and a Queue. What will be the O() for each of the data structure to remove/pop/poll the exact element?

Think that 1,2,3,4…n already added to both Stack and Queue. Now we add one more item. What would be O()s for removing the lastly added item from both Stack and Queue?

Answer

Answer: Let's think that there were already *n* items on both stack and queue. Now we add one more item.

For stack, the item added to the top. Now to remove/pop that item we just need *1* step. So, the complexity is O(1).

For Queue, the new item added to the end of the queue. We cannot remove that item until it reaches to the front of the queue. That means *n* items will be removed in *n* steps and then additional *1* step required to remove our new item. So, big-O will be O(n+1) ~ O(n).

Try it yourself

Take a look at the documentation for Java's Queue class and you'll see that it includes **offer** and **poll** methods as well as **add** and **remove**. What is the difference between the two sets of methods?

The *offer* method returns true if the operation completed successfully or false if the new item couldn't be added (for example, if the queue is full and can't be expanded), Generally, doesn't return false as queue has dynamic memory allocation. The *add* method throws an exception if the operation couldn't complete. The other two methods are similar: *poll* returns *null* if the queue is empty, but *remove* throws an exception

Different Classes that implements Queue

In Java, the `Queue` interface is implemented by several classes, including `LinkedList`, `ArrayDeque`, and `PriorityQueue`. Each of these classes represents a different implementation of a queue, and they have distinct characteristics and use cases.

1. LinkedList class:

- The `LinkedList` class in Java implements the `Queue` interface and represents a doubly linked list. It allows for efficient insertion and deletion at both ends of the list (front and back).

- Advantages:

  - Efficient for adding and removing elements at both ends of the queue (`offer`, `poll`, `peek` operations).

  - Suitable for cases where you need a simple queue and flexibility for adding and removing elements at both ends.

Different Classes that implements Queue cont. ..

2. ArrayDeque class:

- The `ArrayDeque` class in Java also implements the `Queue` interface and represents a resizable array. It provides efficient operations for adding and removing elements from both ends of the deque.

- Advantages:

  - Efficient for adding and removing elements at both ends of the queue (`offer`, `poll`, `peek` operations).

  - Generally more efficient in terms of memory usage and performance compared to `LinkedList`.

  - Suitable for implementing double-ended queues (deque) and as a general-purpose queue.

Different Classes that implements Queue cont ..

3. PriorityQueue class:

- The `PriorityQueue` class in Java also implements the `Queue` interface but differs in that it maintains the elements in a sorted order based on their natural ordering or a custom comparator.

- Elements are ordered based on their priority (e.g., smallest element first for natural ordering).

- Advantages:

  - Efficient for retrieving and removing the element with the highest priority (`poll`, `peek` operations).

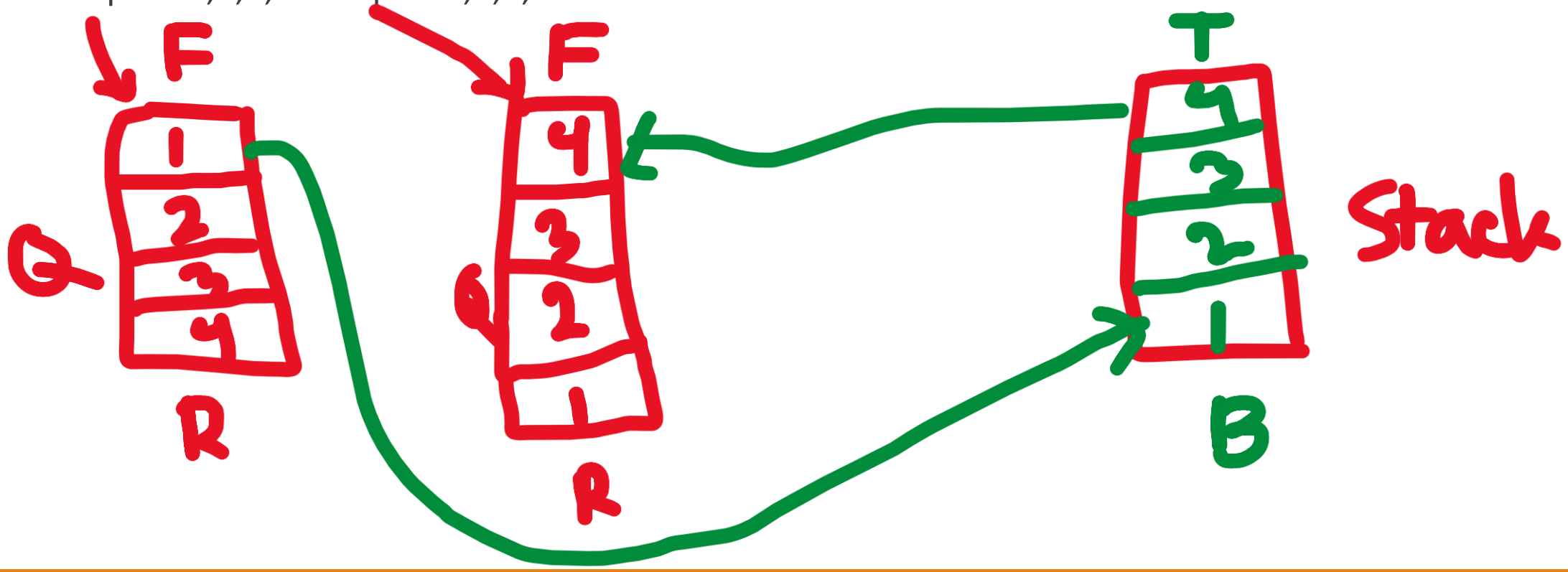
  - Suitable for applications where elements need to be processed in a specific order of priority.

In summary, `LinkedList` and `ArrayDeque` are general-purpose queue implementations, with `LinkedList` providing flexibility for efficient insertion and deletion at both ends, while `ArrayDeque` often offers better performance and memory usage. On the other hand, `PriorityQueue` maintains elements in a sorted order based on their priority, making it suitable for applications where element retrieval by priority is essential. Choose the appropriate queue implementation based on your specific use case and performance requirements.

Practice Problem 1

Reverse a Queue using a Stack. What is the O()?

Input : 1,2,3,4 output: 4,3,2,1



Practice Problem 2

Given two queues, merge them into a single queue by alternating elements.

Queue1: [1, 3, 5]

Queue2: [2, 4, 6]

Output: [1, 2, 3, 4, 5, 6]

Practice Problem 3

Given a queue and a number *k*, rotate the queue by moving the first *k* elements to the back.

Input: [10, 20, 30, 40, 50],

k = 2

Output: [30, 40, 50, 10, 20]

Deque

A queue that allows adding or removing from either end in $O(1)$ time is called a double-ended queues or “deque”.

The name is pronounced as “deck”

It is also an interface

Code for Deque

```
import java.util.ArrayDeque;
import java.util.Deque;


public class Example {
public static void main(String[] args) {
		Deque<Integer> deque = new ArrayDeque<>();
		deque.addFirst(1);
		deque.addLast(2);
		int first = deque.removeFirst();
		int last = deque.removeLast();
		System.out.println("First: " + first + ", Last: " + last);
}
}
```

Advantage of Deque

1. Double-Ended: The main advantage of the Deque interface is that it provides a double-ended queue, which allows elements to be added and removed from both ends of the queue. This makes it a good choice for scenarios where you need to insert or remove elements at both the front and end of the queue.
2. Flexibility: The Deque interface provides a number of methods for adding, removing, and retrieving elements from both ends of the queue, giving you a great deal of flexibility in how you use it.
3. Blocking Operations: The Deque interface provides blocking methods, such as takeFirst and takeLast, that allow you to wait for elements to become available or for space to become available in the queue. This makes it a good choice for concurrent and multithreaded applications.

https://www.geeksforgeeks.org/deque-interface-java-example/#

Disadvantages of Deque

1. Performance: The performance of a Deque can be slower than other data structures, such as a linked list or an array, because it provides more functionality.
2. Implementation Dependent: The behavior of a Deque can depend on the implementation you use. For example, some implementations may provide thread-safe operations, while others may not. It's important to choose an appropriate implementation and understand its behavior before using a Deque.

Home Practice Problem

- Implement stack using two queues
- Implement Queue using two stacks

Deque implementation using Linked List

Home practice

Functions:

addFirst()

addLast()

removeFirst()

removeLast()

peekFirst()

peekLast()

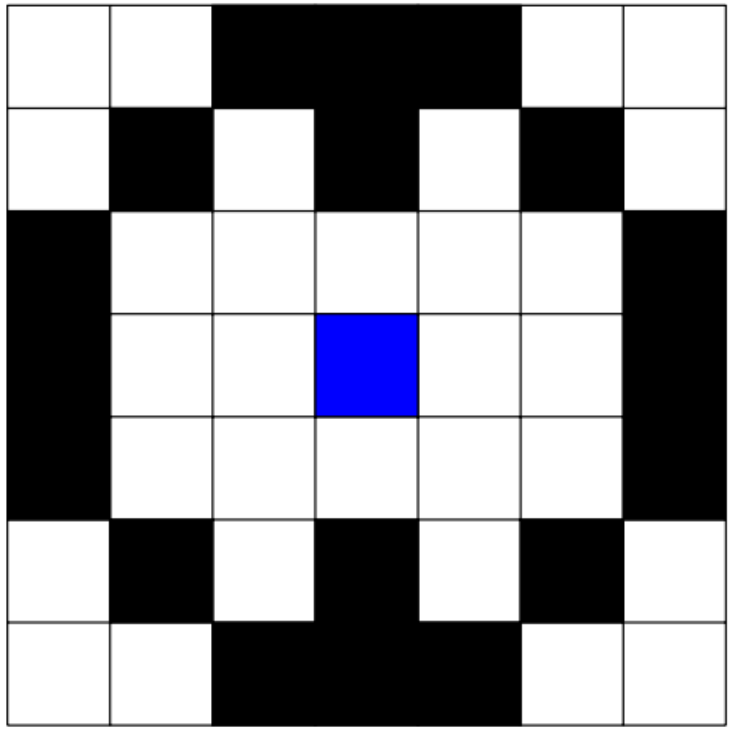
isEmpty()

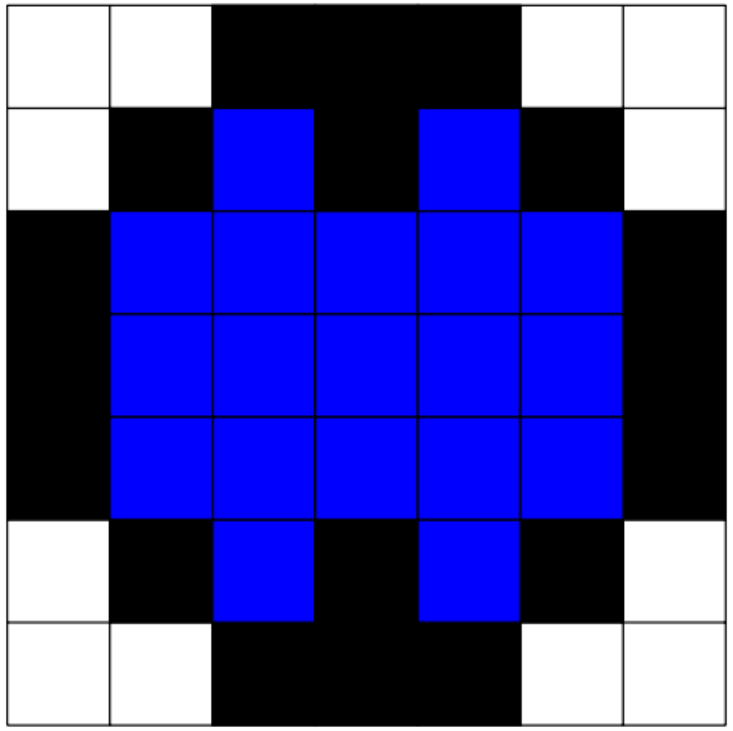
Size()

why an ArrayList is a bad choice for a queue or deque implementation?

ArrayList can append to its end in average $O(1)$ time by increasing the size of the backing array when it becomes full. However, interacting with the head of the list is still difficult. Adding or removing at the first array position requires shifting all of the other elements, which is an $O(n)$ operation. Therefore, the standard ArrayList can't provide $O(1)$ access to both ends.

Flood filling algorithm: Application of Queue

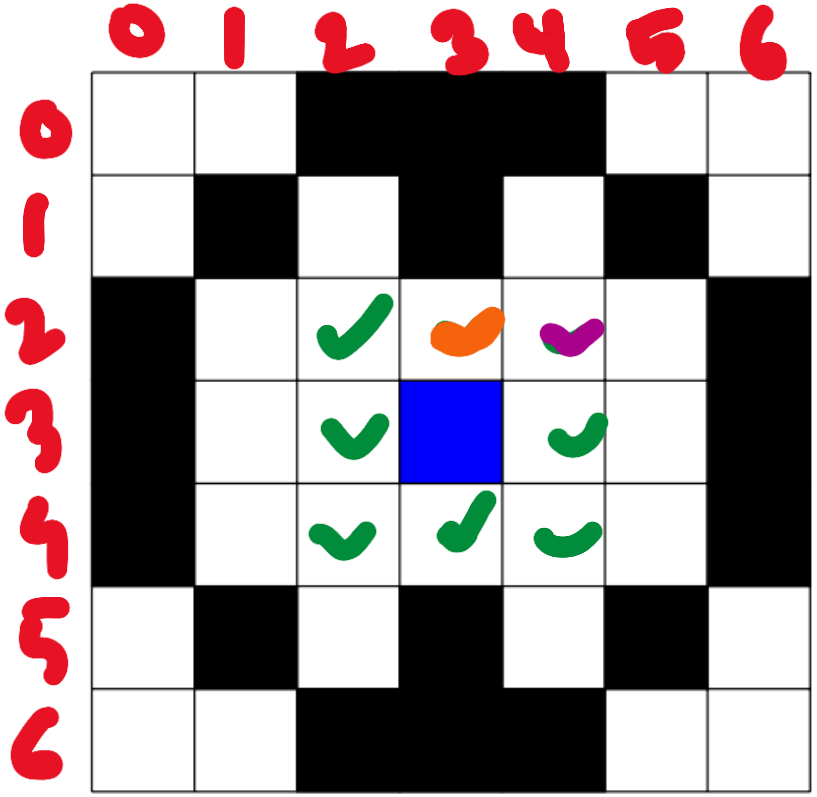




Breadth-first flooding algorithm

- Initialize a queue that contains the starting square.
- While the queue is not empty, remove the next square, fill it, then add its unfilled neighbors to the queue.
- Keep track of squares that have already been added to the queue so you don’t attempt to visit the same square more than once.

Flood filling (cont.)

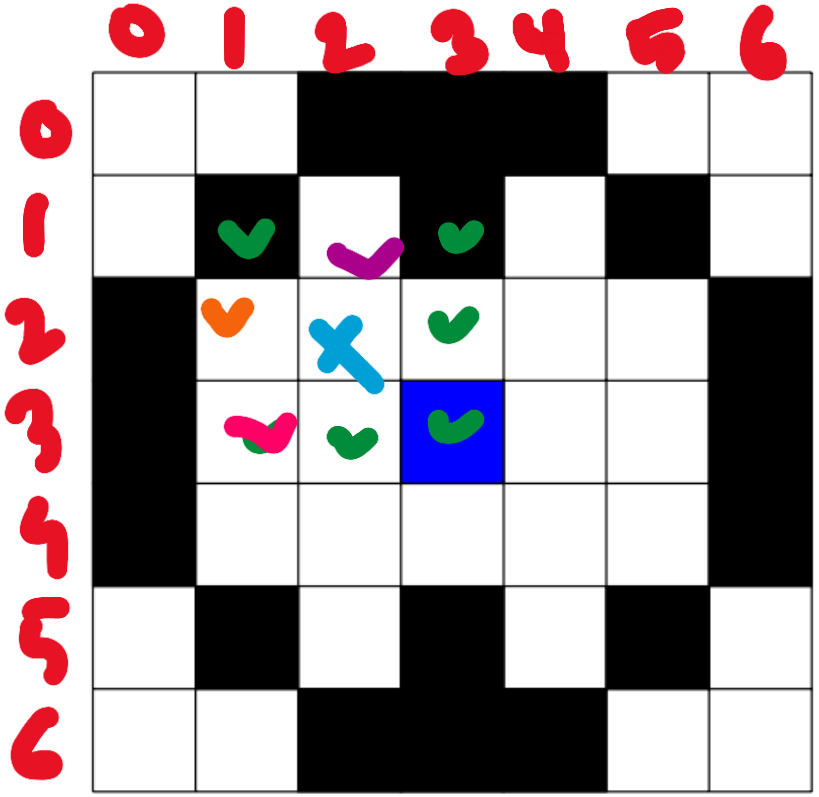


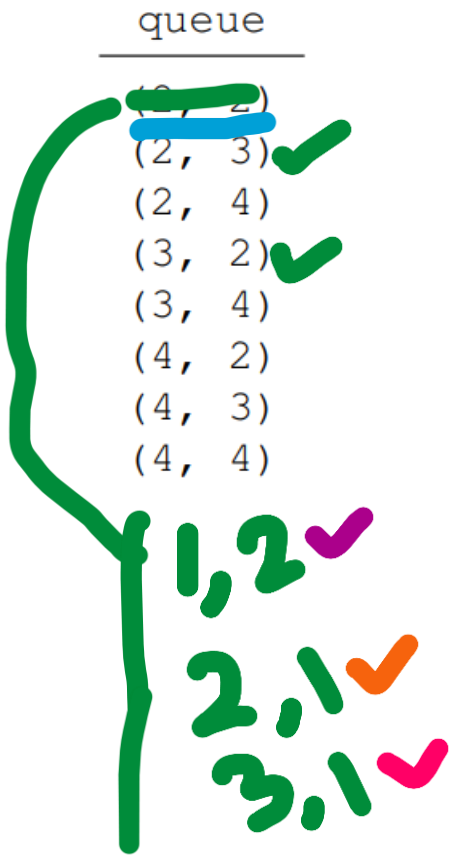
row

Column

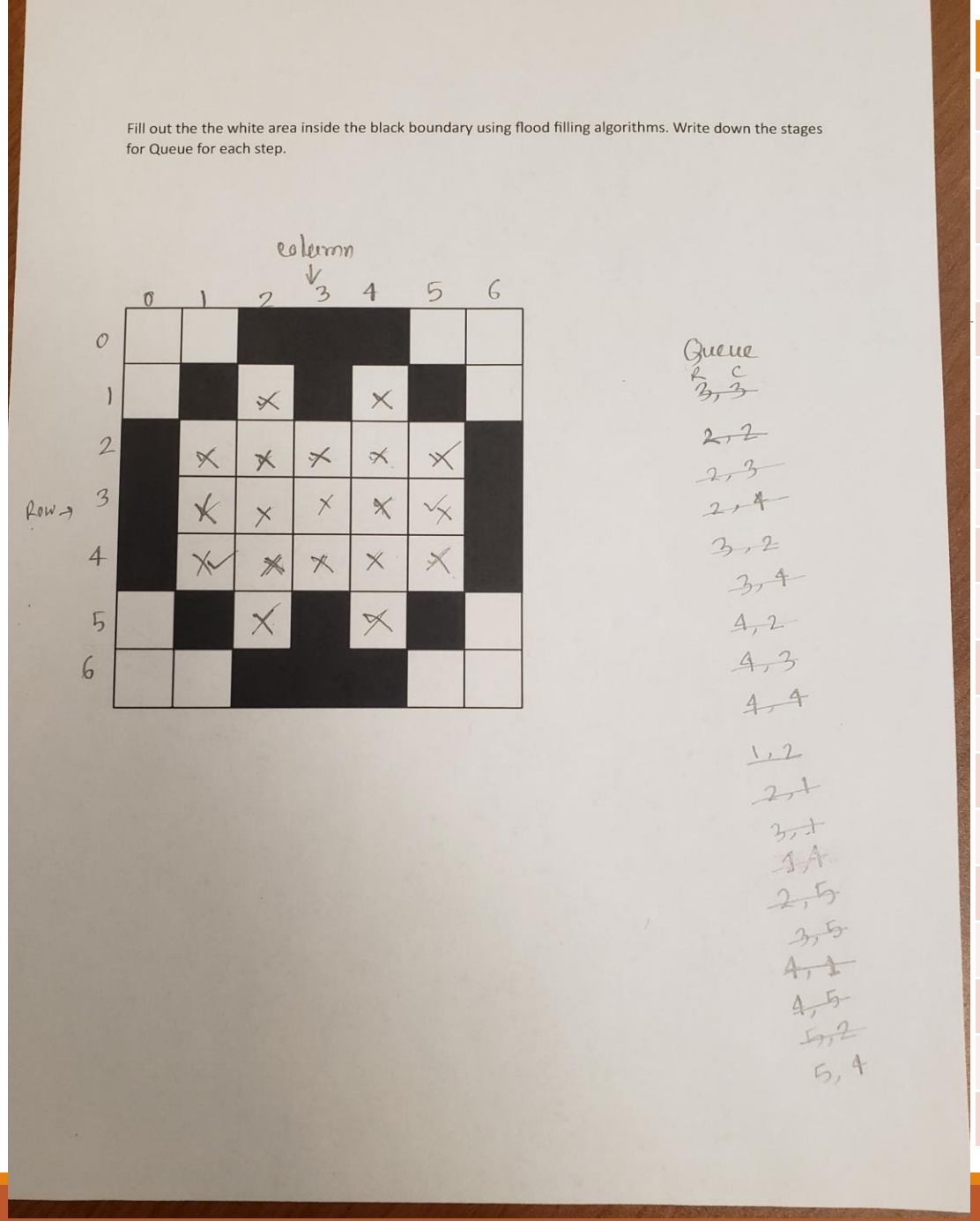
Queue
(2, 2)
(2, 3)
(2, 4)
(3, 2)
(3, 4)
(4, 2)
(4, 3)
(4, 4)

Flood filling (cont.)





Try for the rest!!



Queue
3,3
2,2
2,3
2,4
3,2
3,4
4,2
4,3
4,4
1,2
2,1
3,1
1,4
2,5
3,5
4,1
4,5
5,2
5,4

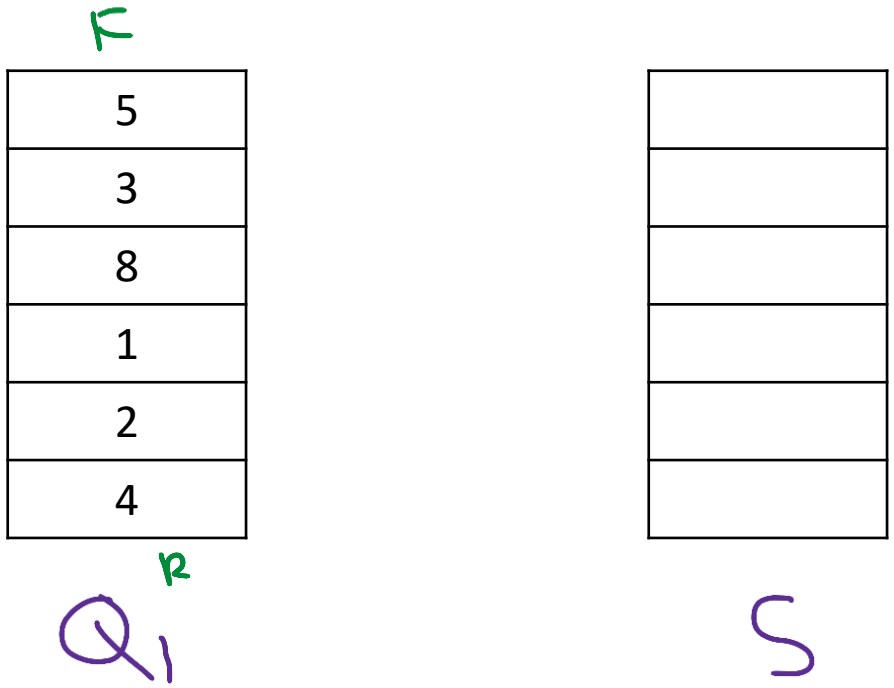
Final outcome for floodFilling

Example code
On Canvas

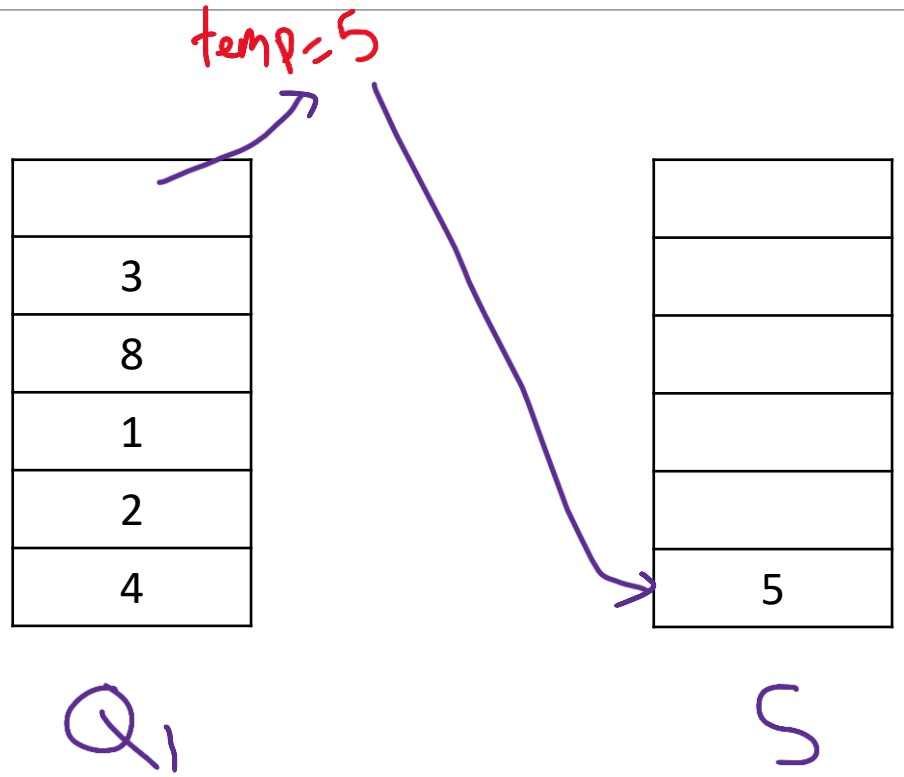
Some example problems

https://www.geeksforgeeks.org/top-50-problems-on-queue-data-structure-asked-in-sde-interviews/

Sort a queue with a Stack



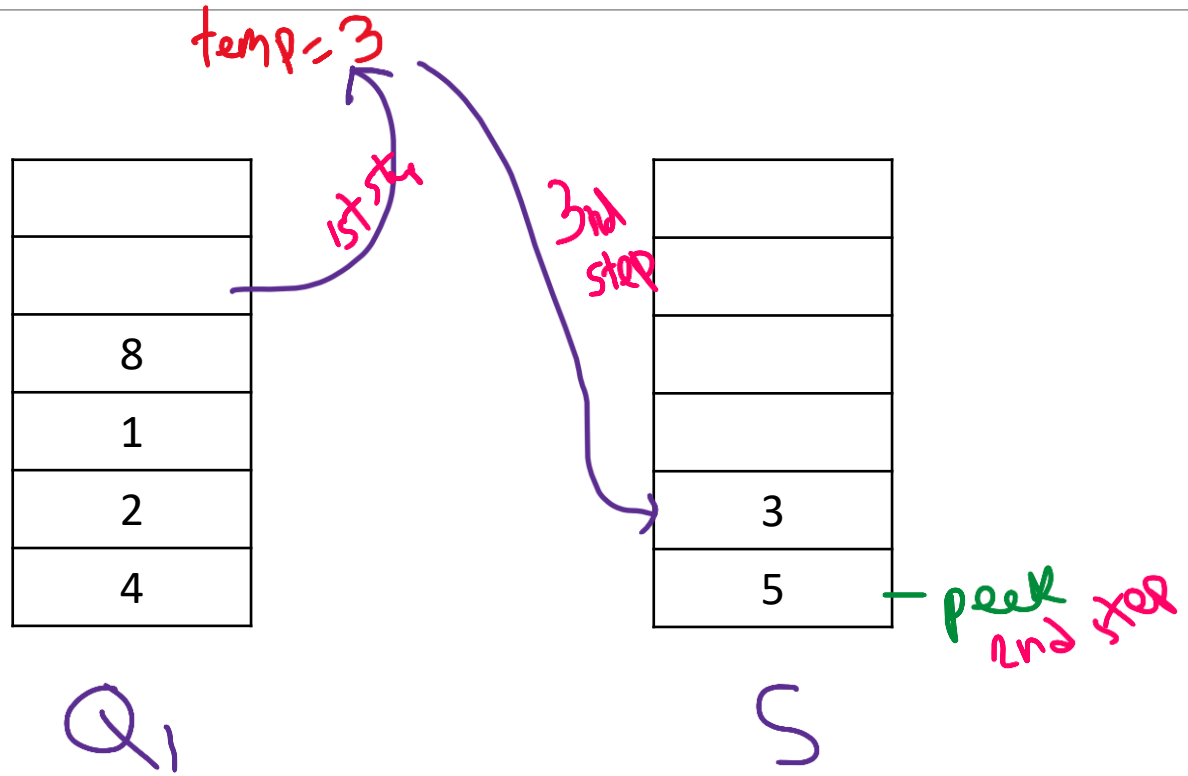
Sort a queue with a Stack



roll an element from the queue and store it in temp.

Push temp to the stack

Sort a queue with a Stack



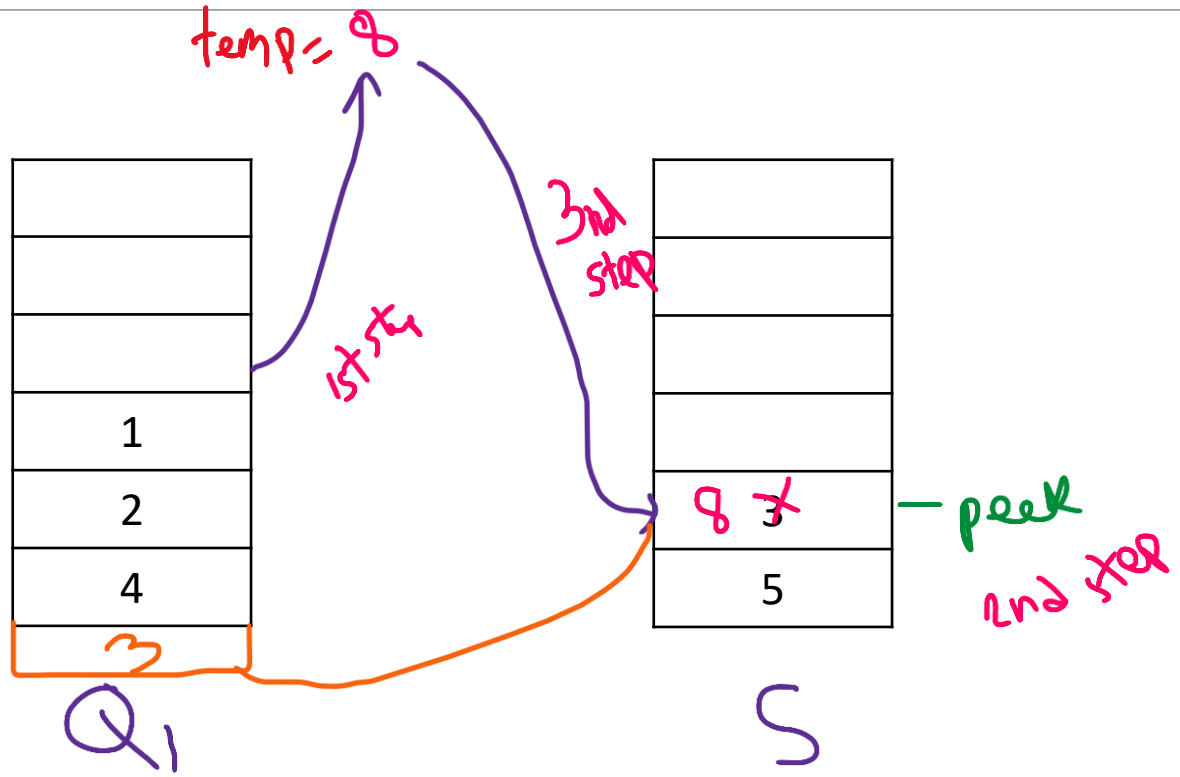
roll an element from the queue and store it in temp.

Peek at stack to check if that is less than temp. if yes, pop and offer it to the queue.

NO

Push temp to the stack

Sort a queue with a Stack

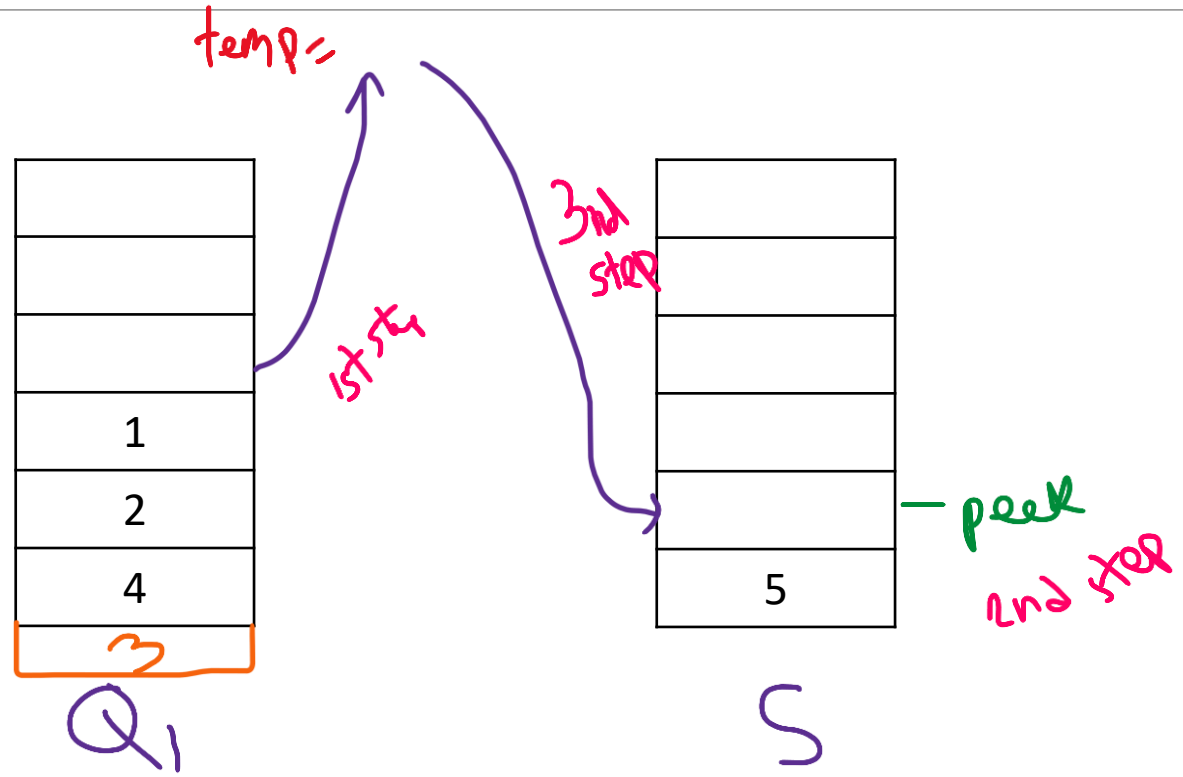


roll an element from the queue and store it in temp.

Peek at stack to check if that is less than temp. if yes, pop and offer it to the queue. Yes

Push temp to the stack

Sort a queue with a Stack(Do the rest)



roll an element from the queue and store it in temp.

Peek at stack to check if that is less than temp. if yes, pop and offer it to the queue.

Yes

Push temp to the stack

Follow the step until the Queue get empty.

Home Practice Dump

Code: Sort a Queue with a Stack