Queue

describe the functioning of the stack and queue data types correctly using the terms ‘last in last out’ and ‘first in first out’

A **queue** is a first-in first-out (FIFO) abstract data type that is heavily used in computing. Uses for queues involve anything where you want things to happen in the order that they were called, but where the computer can't keep up to speed.

For example:

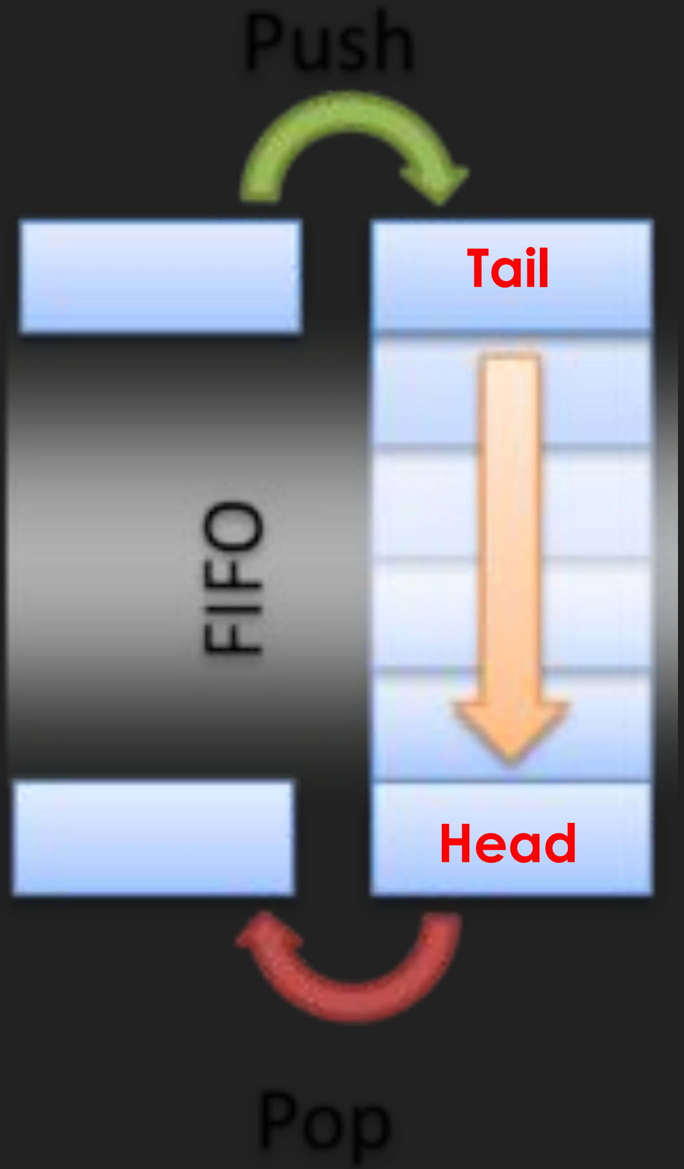
Keyboard Buffer - you want the letters to appear on the screen in the order you press them. You might notice that when your computer is busy the keys you press don't appear on the screen until a little while after you press them. When they do appear they appear in the order you press them.

Printer Queue - you want print jobs to complete in the order you sent them, i.e. page 1, page 2, page 3, page 4 etc. When you are sharing a printer several people may send a print job to the printer and the printer can't print things instantly, so you have to wait a little while, but the items output will be in the order you sent them.

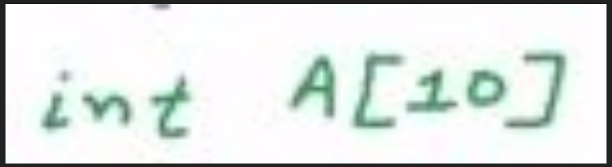
There are several different types of queues such as the ones described below:

1. Linear.

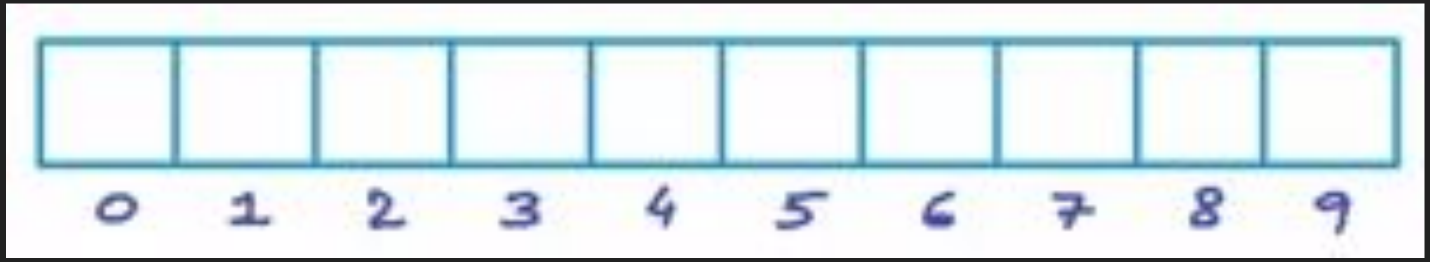
In this queue form, elements are able to join the queue at one end and can exit from the queue at the other end. **First In First Out** (FIFO).

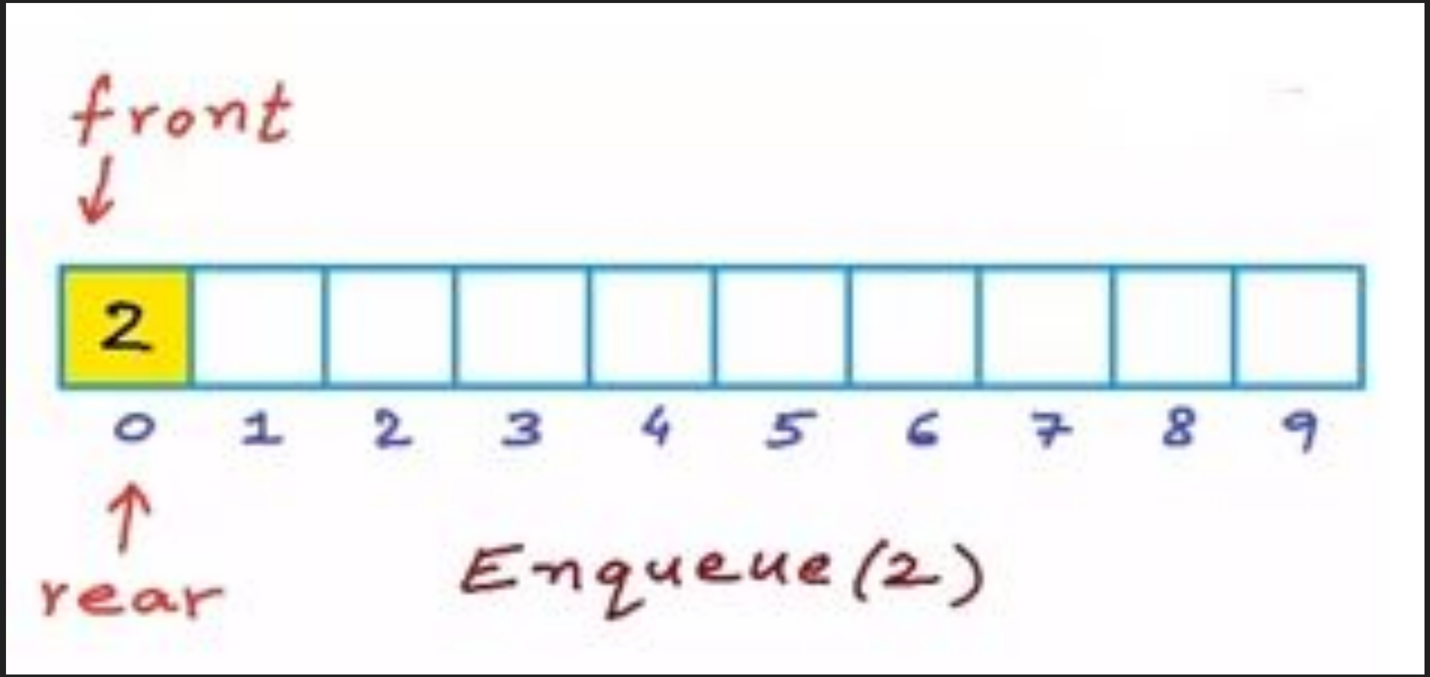


Array implementation

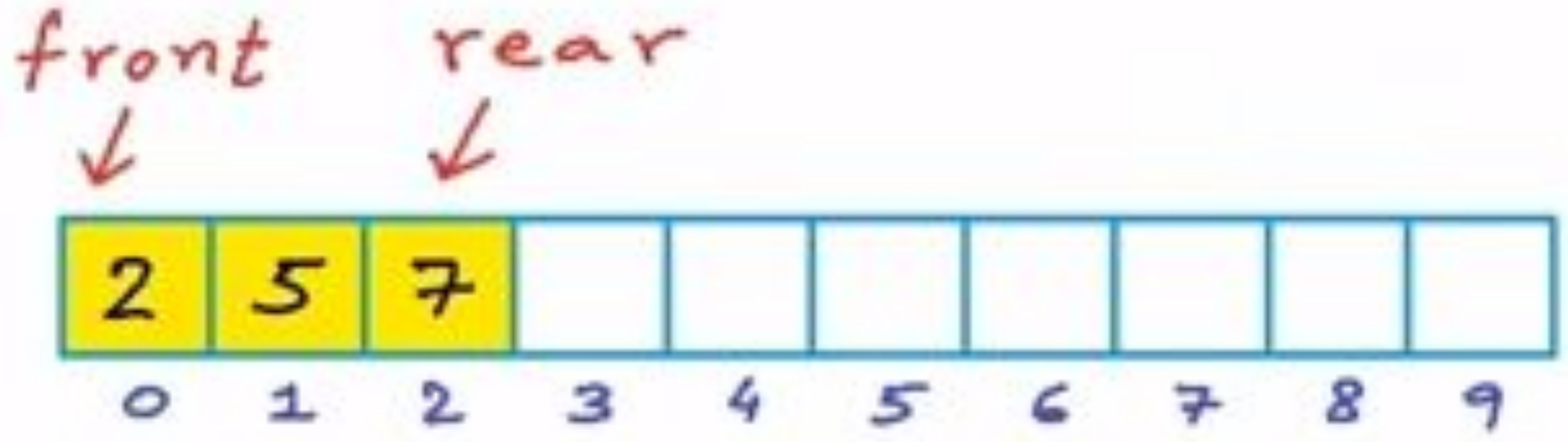


int A[10]



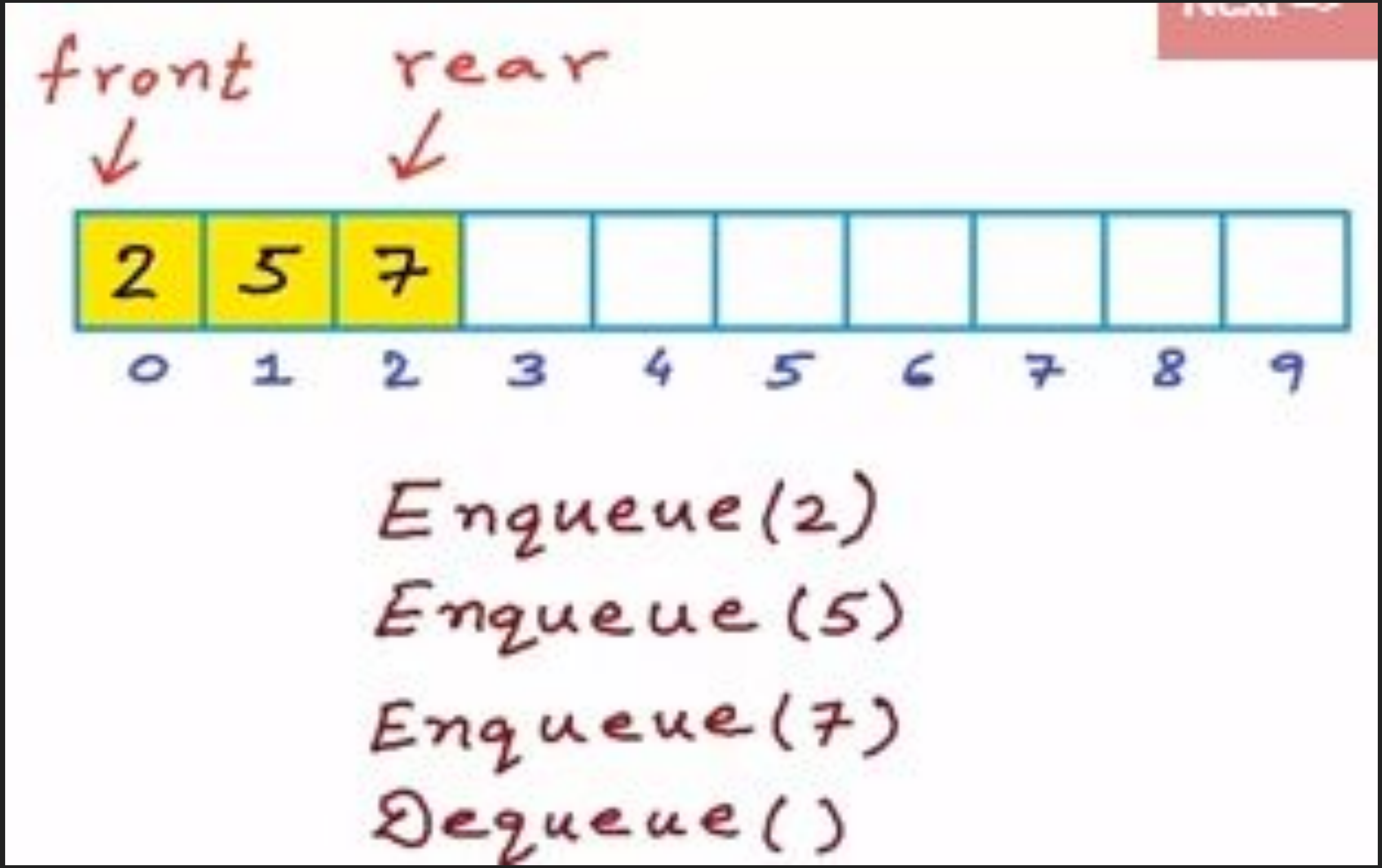


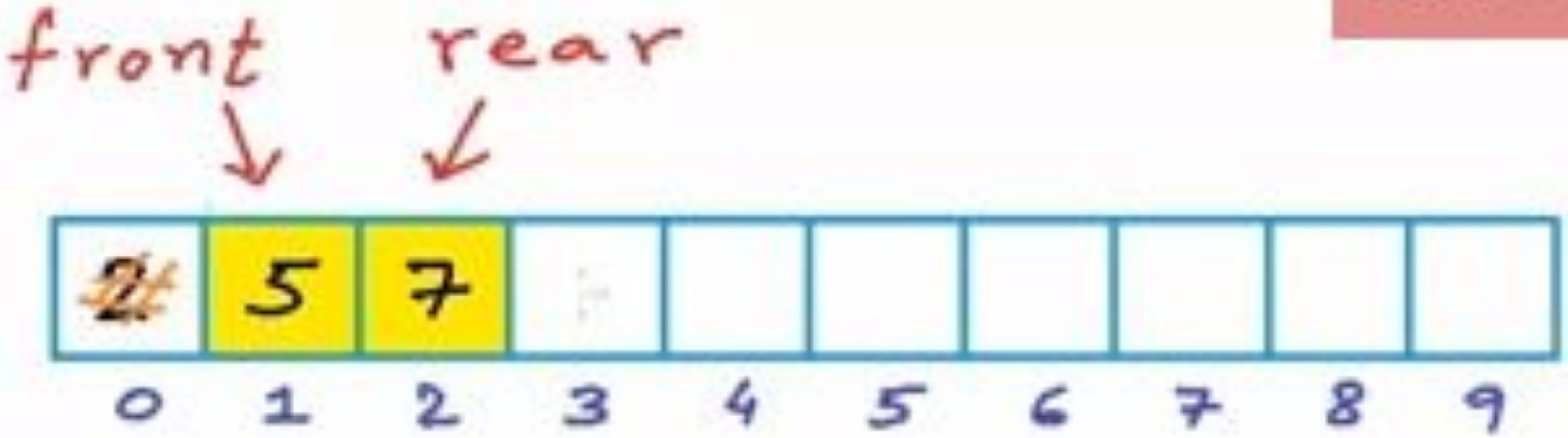
https://www.youtube.com/watch?v=okr-XE8yTO8



Enqueue(2)
Enqueue(5)
Enqueue(7)

?


front
rear
2
5
7
0
1
2
3
4
5
6
7
8
9
Enqueue(2)
Enqueue(5)
Enqueue(7)
Dequeue()

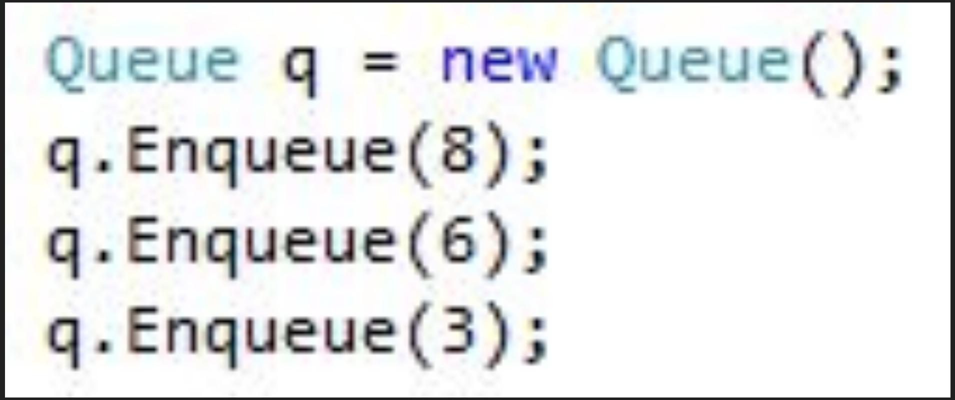


Enqueue(2)
Enqueue(5)
Enqueue(7)
Dequeue()

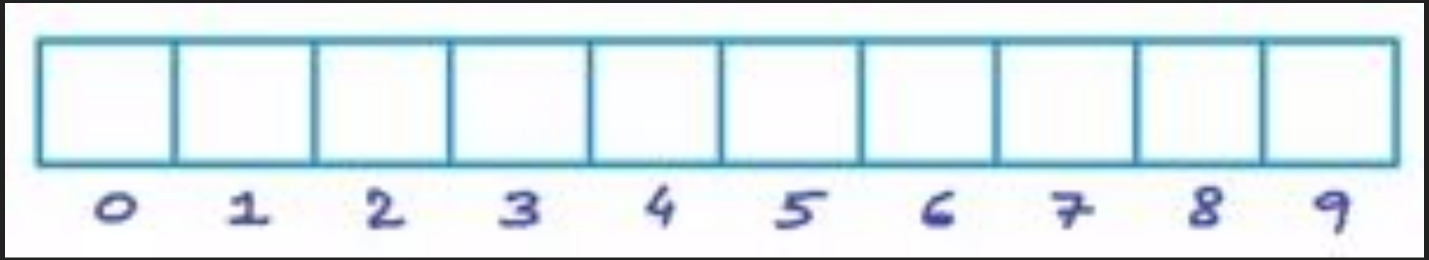
The queue supports the following operations:

push - (Enqueue) - the operation of inserting a new element,
pop - (Dequeue) - the operation to delete a new item,
count - check the number of items in the queue.

?



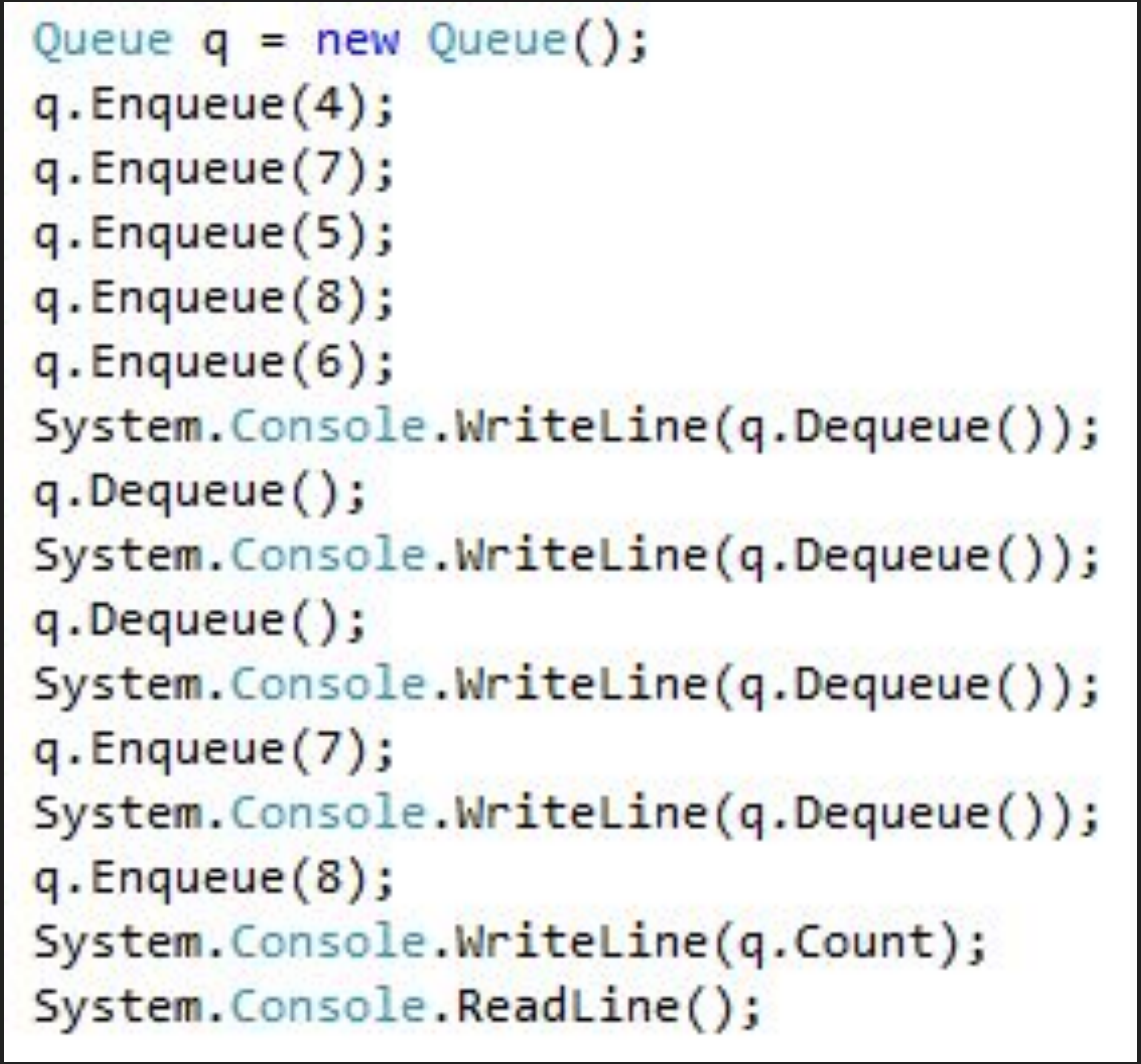
```
Queue q = new Queue();
q.Enqueue(8);
q.Enqueue(6);
q.Enqueue(3);
```



q.Count - ?

What will be printed as a result of this code?

?



```
Queue q = new Queue();
q.Enqueue(4);
q.Enqueue(7);
q.Enqueue(5);
q.Enqueue(8);
q.Enqueue(6);
System.Console.WriteLine(q.Dequeue());
q.Dequeue();
System.Console.WriteLine(q.Dequeue());
q.Dequeue();
System.Console.WriteLine(q.Dequeue());
q.Enqueue(7);
System.Console.WriteLine(q.Dequeue());
q.Enqueue(8);
System.Console.WriteLine(q.Count);
System.Console.ReadLine();
```

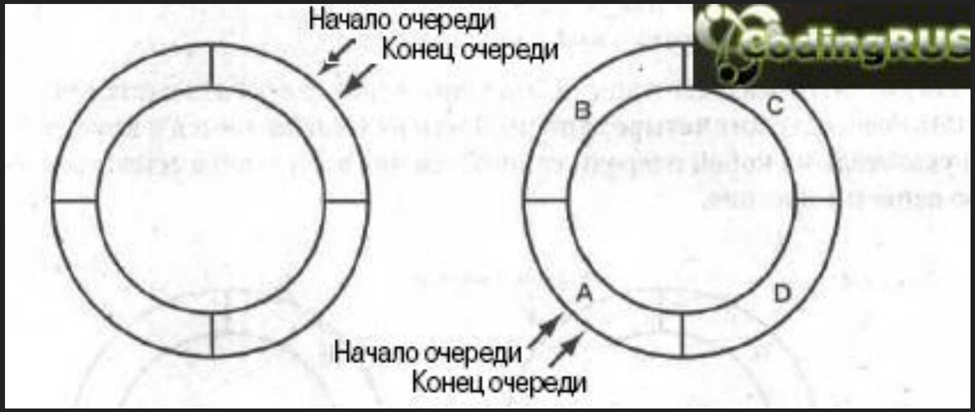
A Linear queue is constantly working its way through memory, it does not re-use memory. As a result if a large number of items are being added and removed from the list this means the lists will grow very large without containing much data. It is not a very efficient solution and will use up large amounts of memory.

2. Circular

Unlike linear queues Circular queues allow for memory to be re-used:

Generally, a circular buffer requires three pointers:

- one to the actual buffer in memory
- one to point to the start of valid data
- one to point to the end of valid data



	+	-
Linear Queue	Fast to implement, with simple code	Uses up large amounts of memory, might overwrite important memory locations
Circular Queue	They reuse memory space, meaning they won't overwrite data or run out of memory (within limits)	Involve storing pointers as well as data, taking up more space than linear queues Limited by the amount of data available in the buffer (array)