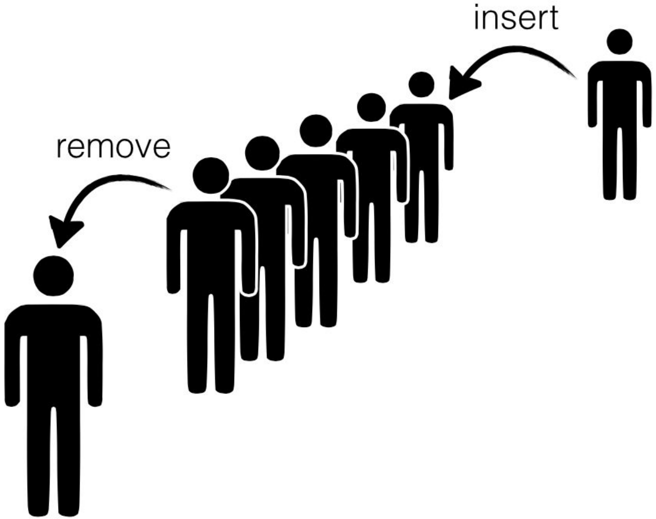


Queues

Lecture 6

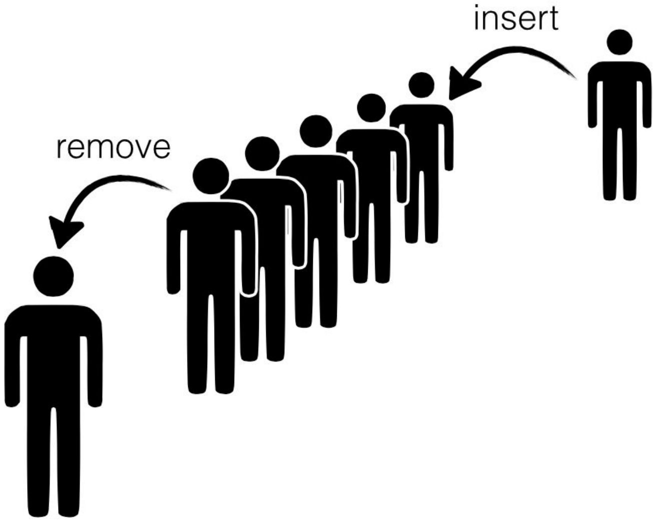
MIC

QUEUES (I'M CHEATING HERE ☺)

The Queue ADT

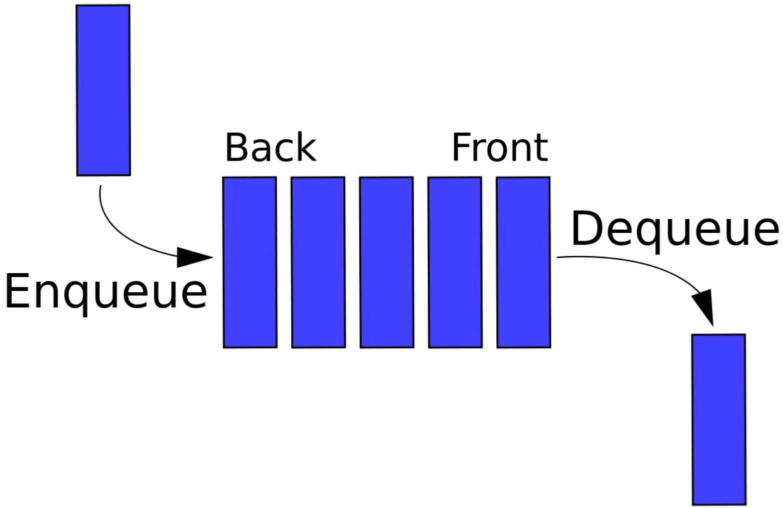
A **Queue** is a collection of objects inserted and removed according to the

First In First Out (FIFO) principle. Think of a queue of people to Rubios



Queue operations

Enqueue (insert) and **Dequeue (remove)** are the two main operations

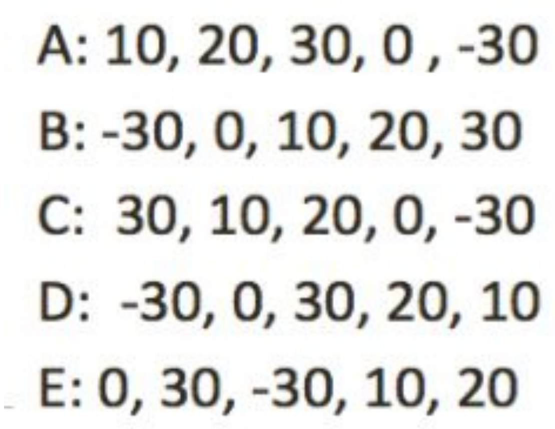


QUESTION

When using enqueue operation to place the following items in a queue:

```
enqueue(10)

enqueue(20)

enqueue(30)

enqueue(0)

enqueue(-30)
```

The output when dequeuing from the queue is:



A: 10, 20, 30, 0 , -30

B: -30, 0, 10, 20, 30

C: 30, 10, 20, 0, -30

D: -30, 0, 30, 20, 10

E: 0, 30, -30, 10, 20

Implementation. Arrays. O(1)

Main update methods:

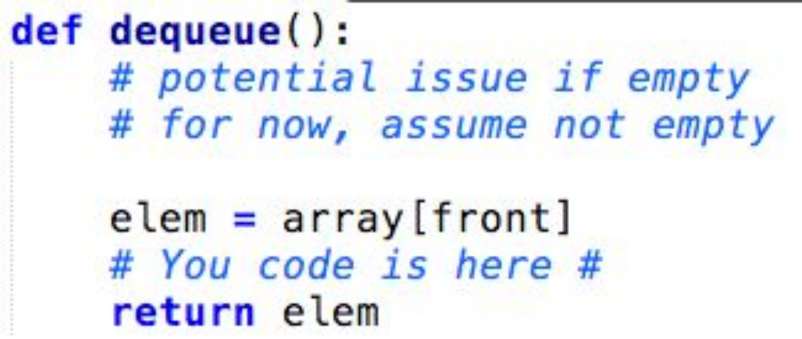
- `Enqueue(e)`
- `Dequeue( )`

Additional useful methods

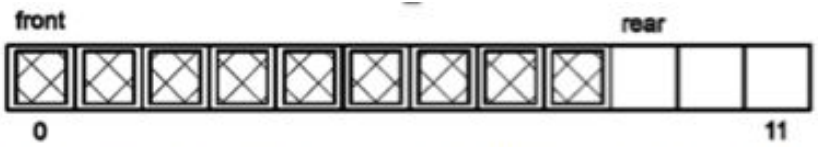
- `Peek():`   Same as dequeue, but does not remove the element
- `Empty():` Boolean, True when the queue is *empty*
- `Size():`   Returns the size of the queue

(Incorrect) Attempts to implement it

Regular array: dequeue



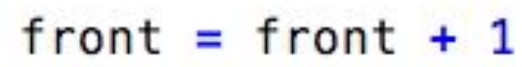
```
def dequeue():
    # potential issue if empty
    # for now, assume not empty

    elem = array[front]
    # You code is here #
    return elem
```



(a) Queue.*front* is always at 0 – shift elements *left* on dequeue().

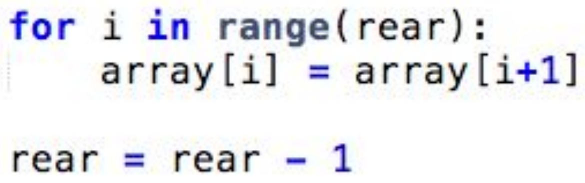
Select the correct code to delete from below:

A:



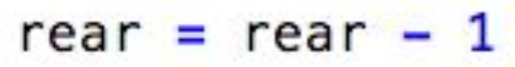
```
front = front + 1
```

C:



```
for i in range(rear):
    array[i] = array[i+1]

rear = rear - 1
```

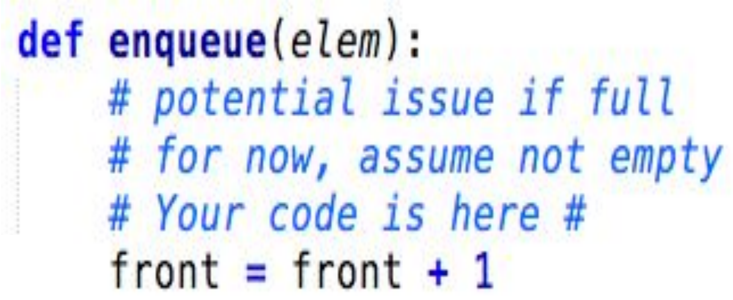
B:



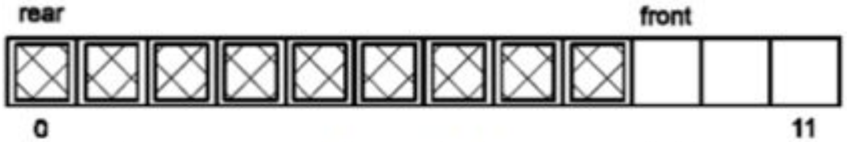
```
rear = rear - 1
```

D: None of these are correct

Regular array: enqueue



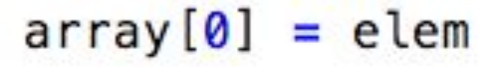
```
def enqueue(elem):
    # potential issue if full
    # for now, assume not empty
    # Your code is here #
    front = front + 1
```



(b) Queue.*rear* is always at 0 – shift elements *right* on enqueue().

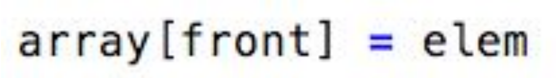
Select the correct code to insert from below:

A:



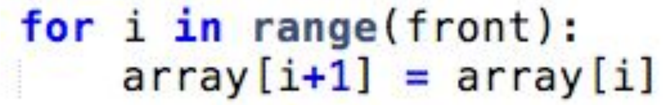
```
array[0] = elem
```

B:

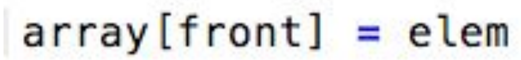


```
array[front] = elem
```

C:



```
for i in range(front):
    array[i+1] = array[i]
```

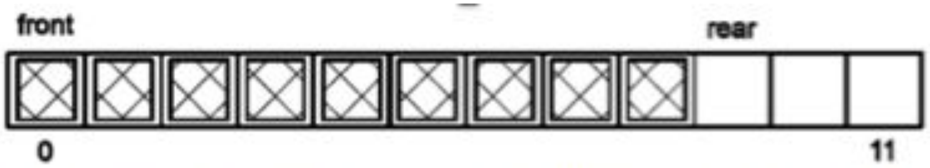


```
array[front] = elem
```

D: None of these are correct

Issues

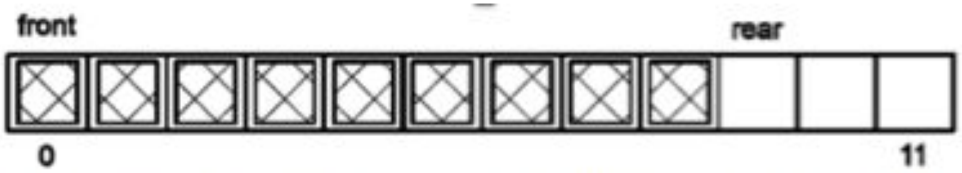
Dequeue:



(a) Queue.*front* is always at 0 – shift elements *left* on dequeue().

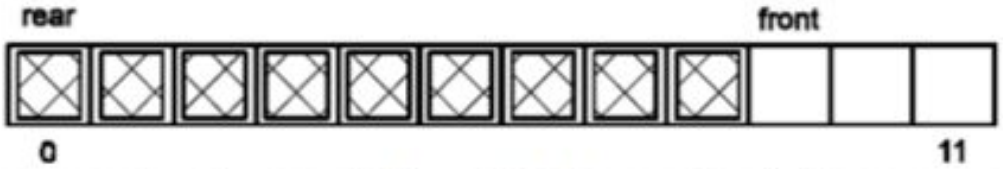
Issues

Dequeue:



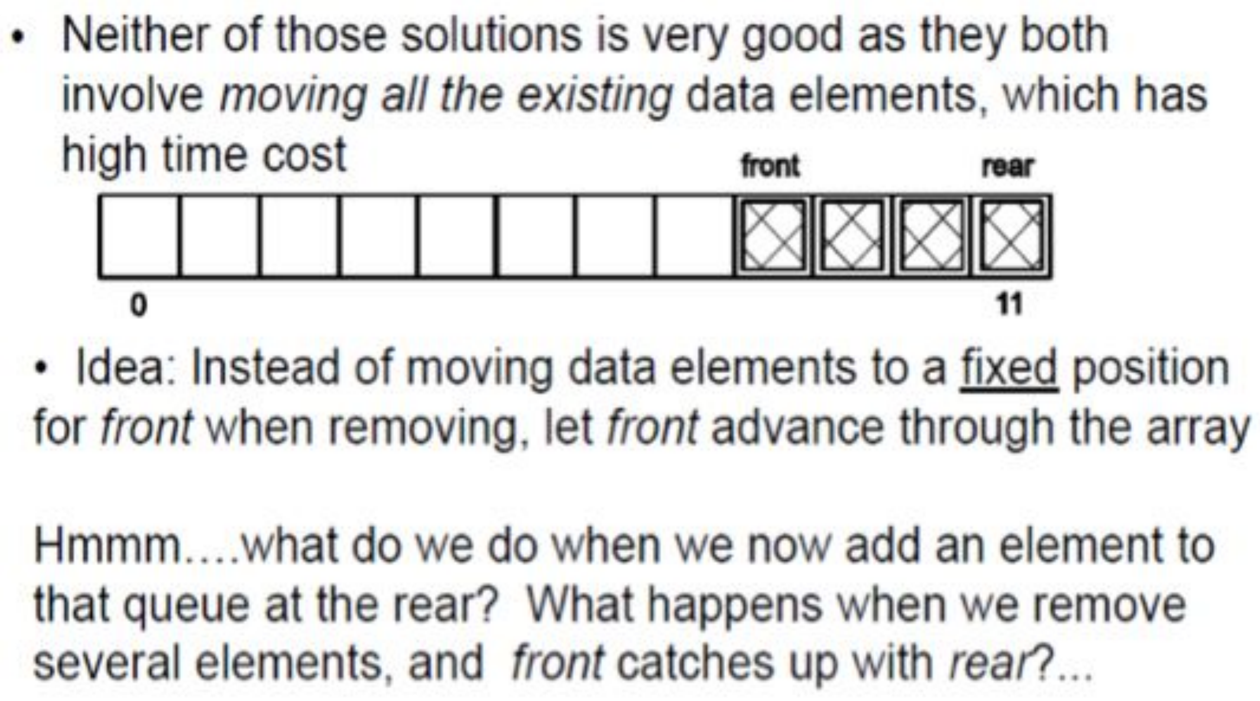
(a) Queue.*front* is always at 0 – shift elements *left* on dequeue().

Enqueue:



(b) Queue.*rear* is always at 0 – shift elements *right* on enqueue().

Regular Array



- Neither of those solutions is very good as they both involve *moving all the existing* data elements, which has high time cost

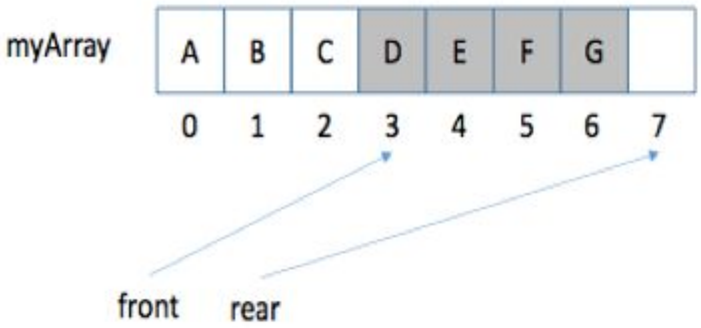
front
rear
0
11

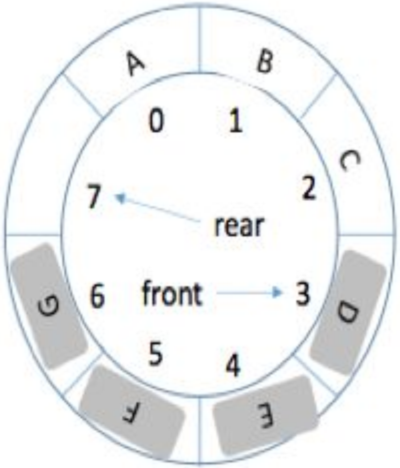
- Idea: Instead of moving data elements to a fixed position for *front* when removing, let *front* advance through the array

Hmmm….what do we do when we now add an element to that queue at the rear? What happens when we remove several elements, and *front* catches up with *rear*?...

Making a linear array appear circular

front==head
rear == tail





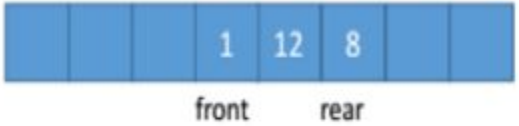
front 3

rear 7

Design decisions: Where do front and rear point?

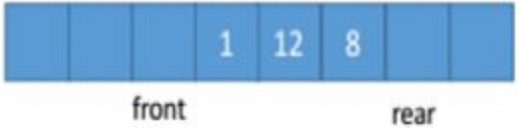
Which of these choices will work?

A



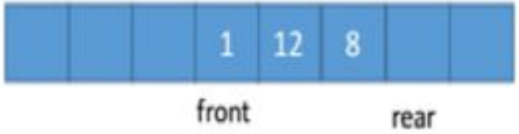
			1	12	8			
			front		rear			

B



			1	12	8			
		front				rear		

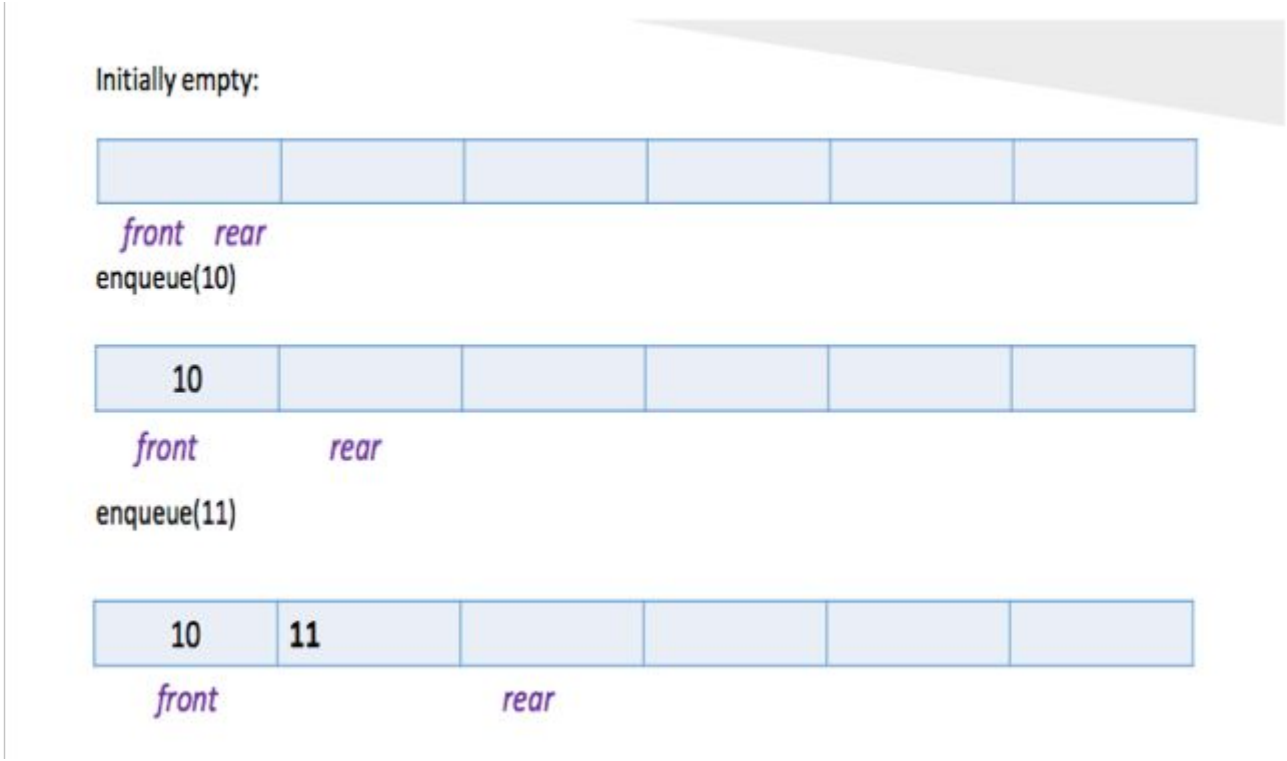
C



			1	12	8			
			front			rear		

D Any of these could work

Queues using circular arrays

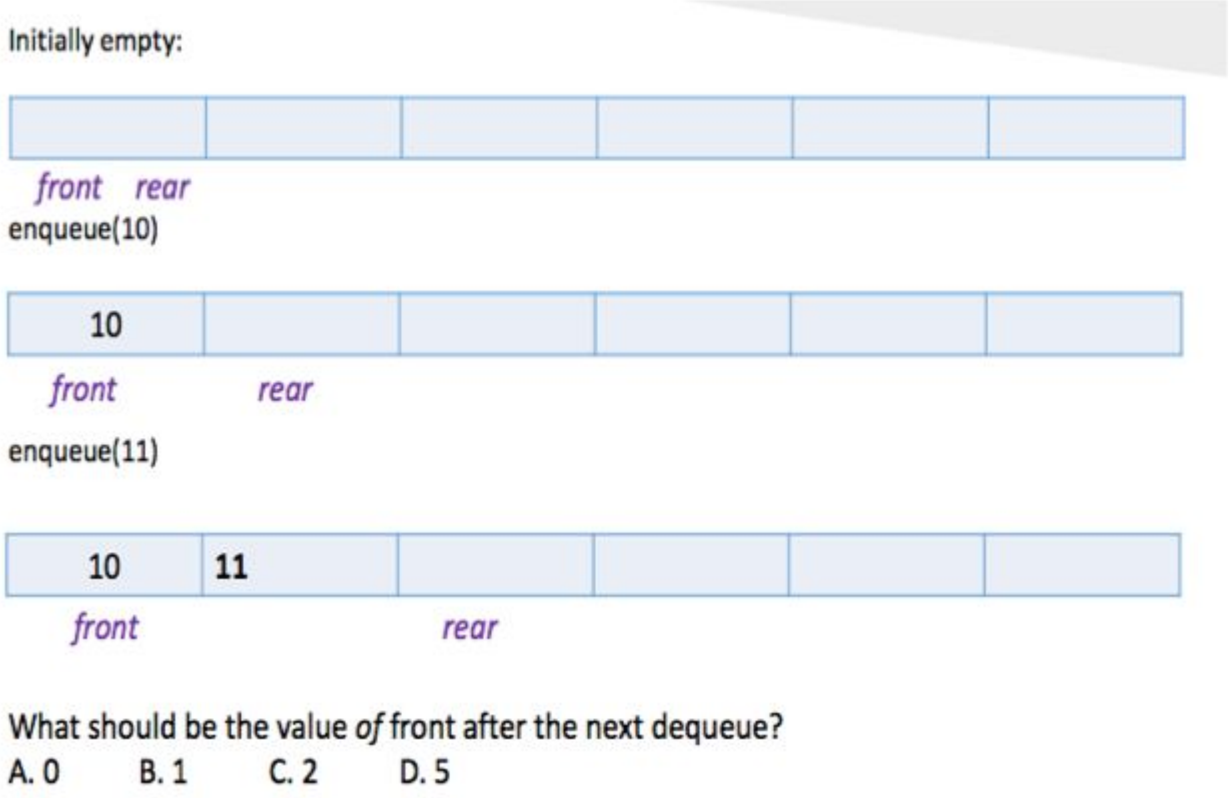


Initially empty:

enqueue(10)

enqueue(11)

Queues using circular arrays



Initially empty:

front *rear*

enqueue(10)

10					

front *rear*

enqueue(11)

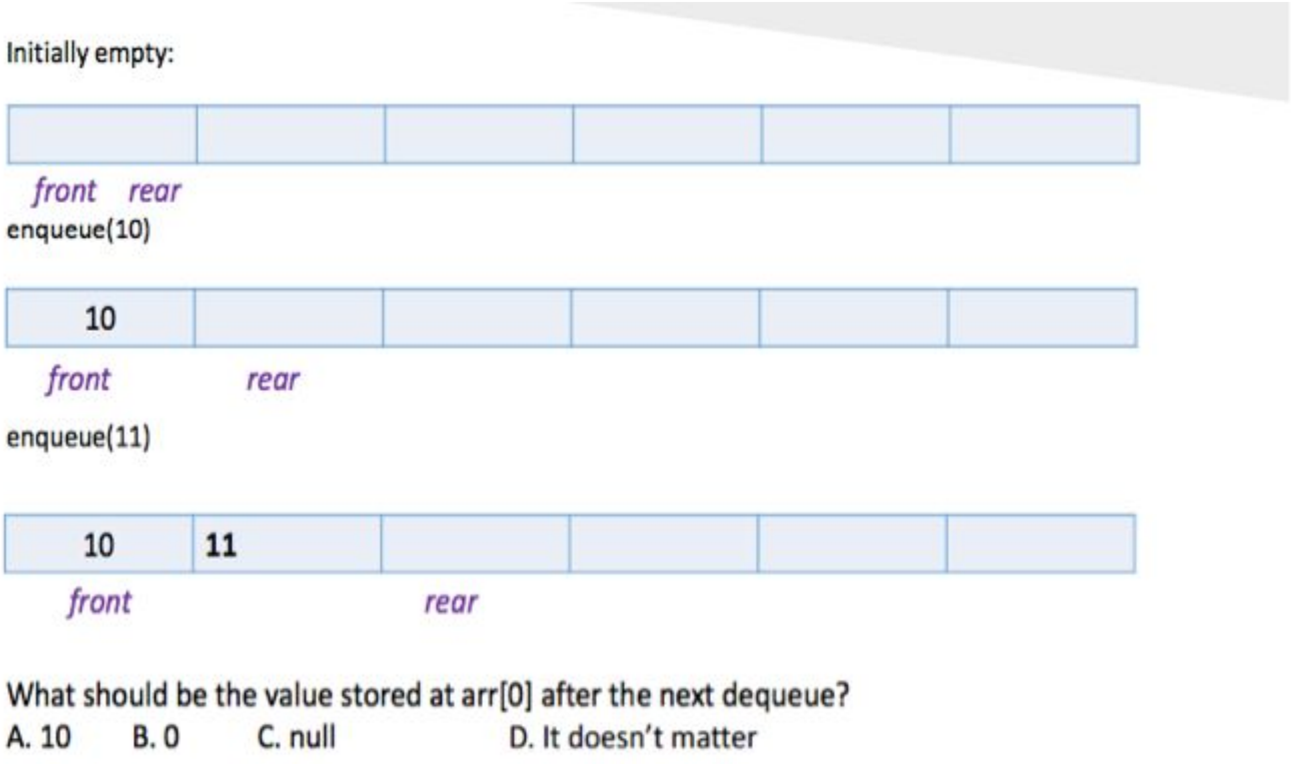
10	11				

front *rear*

What should be the value *of* front after the next dequeue?

A. 0 B. 1 C. 2 D. 5

Queues using circular arrays



Initially empty:

front *rear*

enqueue(10)

10					

front *rear*

enqueue(11)

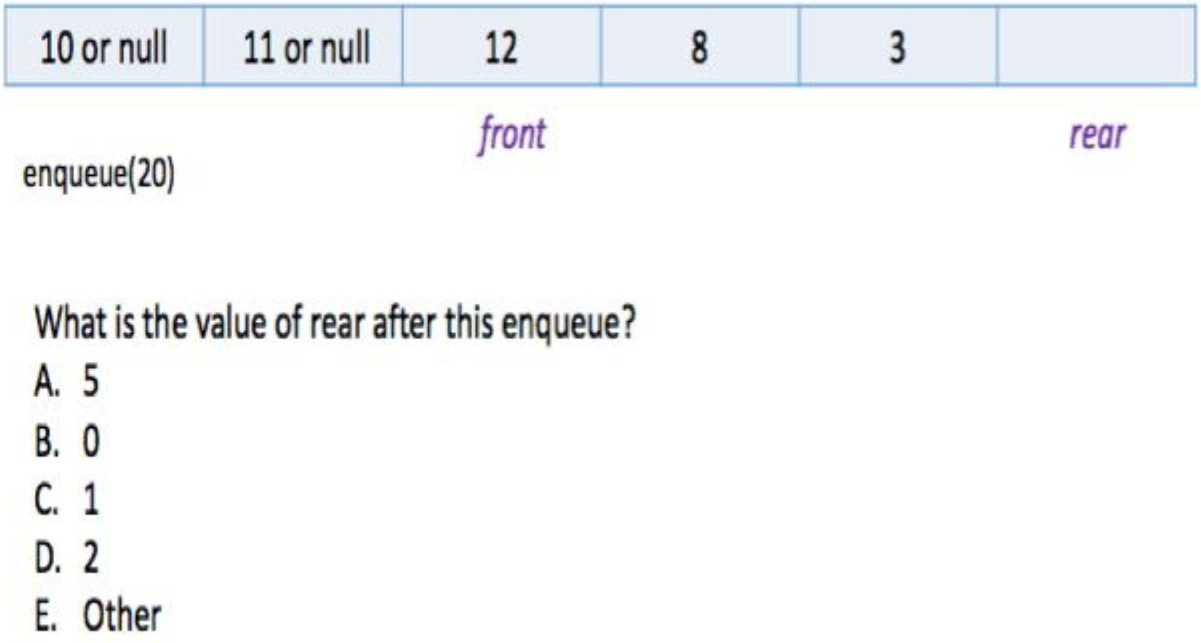
10	11				

front *rear*

What should be the value stored at arr[0] after the next dequeue?

A. 10 B. 0 C. null D. It doesn’t matter

Queues using circular arrays



10 or null	11 or null	12	8	3	
		front			*rear*

enqueue(20)

What is the value of rear after this enqueue?

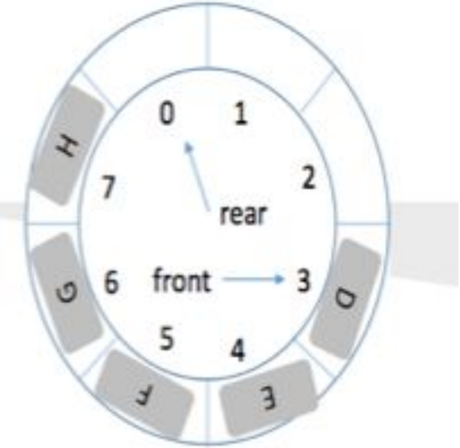
A. 5

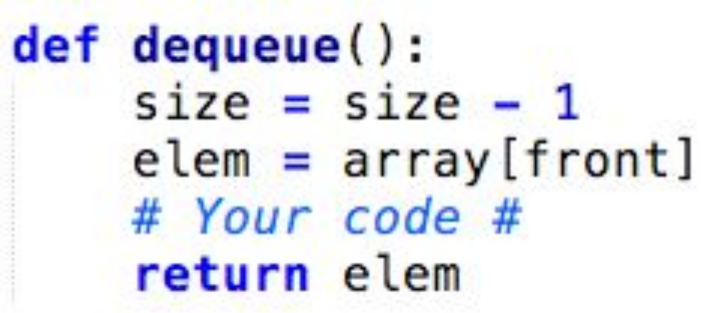
B. 0

C. 1

D. 2

E. Other

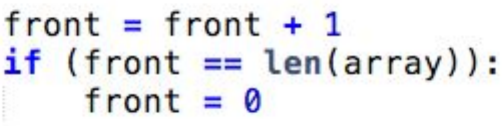




```
def dequeue():
    size = size - 1
    elem = array[front]
    # Your code #
    return elem
```

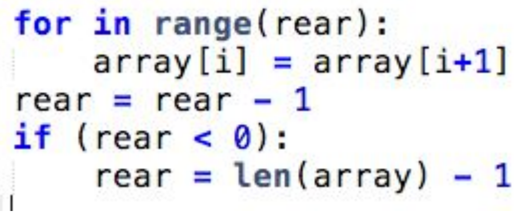
Select the correct code to insert from below:

A:



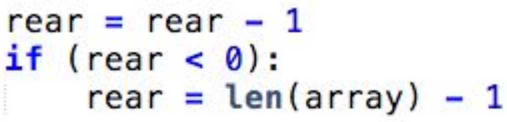
```
front = front + 1
if (front == len(array)):
    front = 0
```

C:



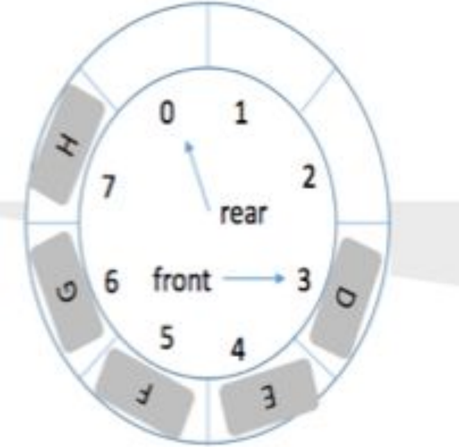
```
for in range(rear):
    array[i] = array[i+1]
rear = rear - 1
if (rear < 0):
    rear = len(array) - 1
```

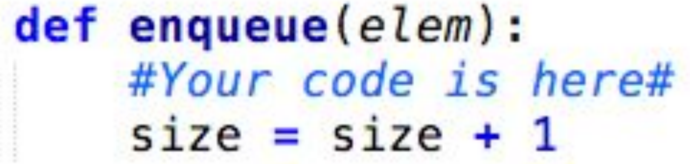
B:



```
rear = rear - 1
if (rear < 0):
    rear = len(array) - 1
```

D: None of these are correct

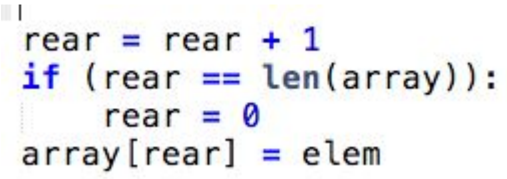




```
def enqueue(elem):
    #Your code is here#
    size = size + 1
```

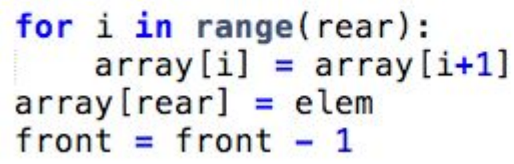
Select the correct code to insert from below:

A:



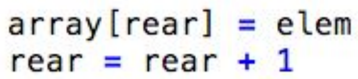
```
rear = rear + 1
if (rear == len(array)):
    rear = 0
array[rear] = elem
```

C:



```
for i in range(rear):
    array[i] = array[i+1]
array[rear] = elem
front = front - 1
```

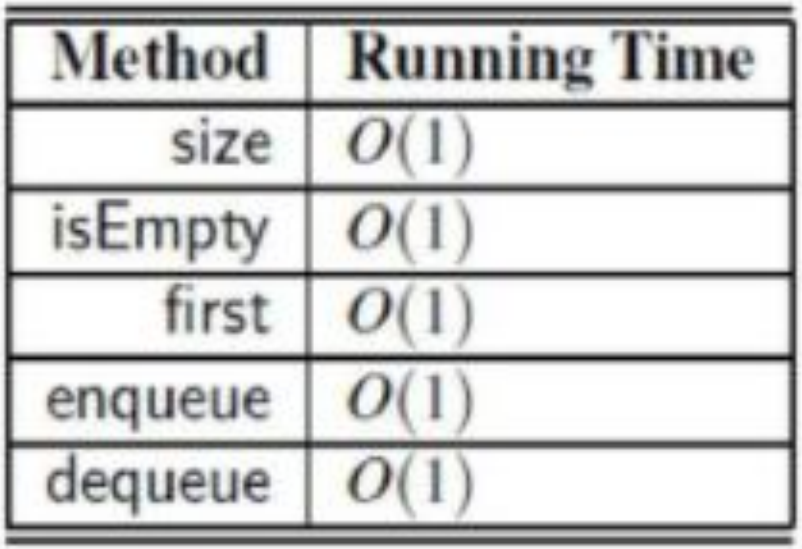
B:



```
array[rear] = elem
rear = rear + 1
```

D: None of these are correct

Complexity of an Array based queues



Method	Running Time
size	$O(1)$
isEmpty	$O(1)$
first	$O(1)$
enqueue	$O(1)$
dequeue	$O(1)$