Lecture 10

QUEUES (I'M CHEATING HERE ☺)

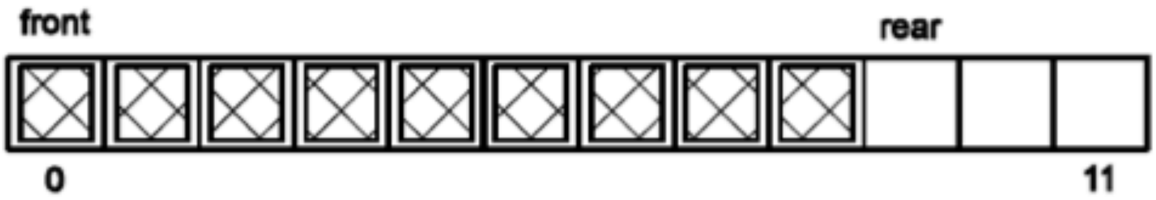
Queues

- **Operations**:
  - enqueue – append (add) an element to the queue.
  - dequeue – remove element from the queue.
  - peek – return element that would be *dequeued* with the next call to dequeue.
- **Enqueue** and **Dequeue** operate at **opposite** ends of the Queue.
- FIFO data structure
  - First In, First Out

Queues, warm up question

- It is a good idea to implement a queue with a class that **extends** LinkedList

- A: Sure, it's fine
- B: No, that's not a great idea



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

```
public E dequeue(){
 // potential issue if empty, for now, assume not empty
 E e = array[front];
 <YOUR CODE HERE>
 return e;
}
```

Select the correct code to insert from below:

A

```
front+ +;
```

B

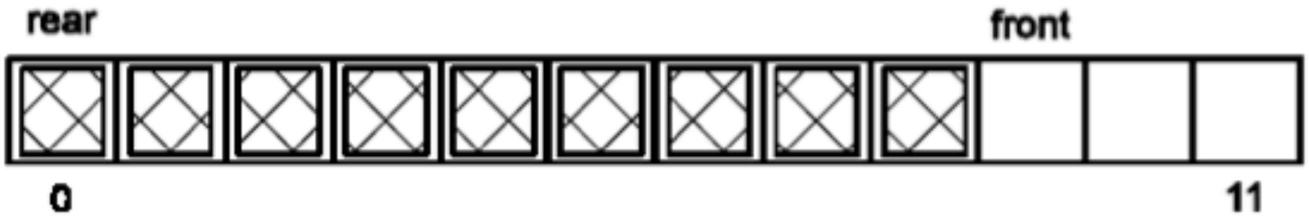
```
rear = rear-1;
```

C

```
for(int i= 0; i<rear; i++) {
  array[i] = array[i+1];
}
rear = rear -1;
```

D

None of these are correct



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

```
public void enqueue( E element){
  // potential issue if full, for now, assume room
  <YOUR CODE HERE>
  front++;
}
```

Select the correct code to insert from below:

A

```
array[0] = e;
```

B

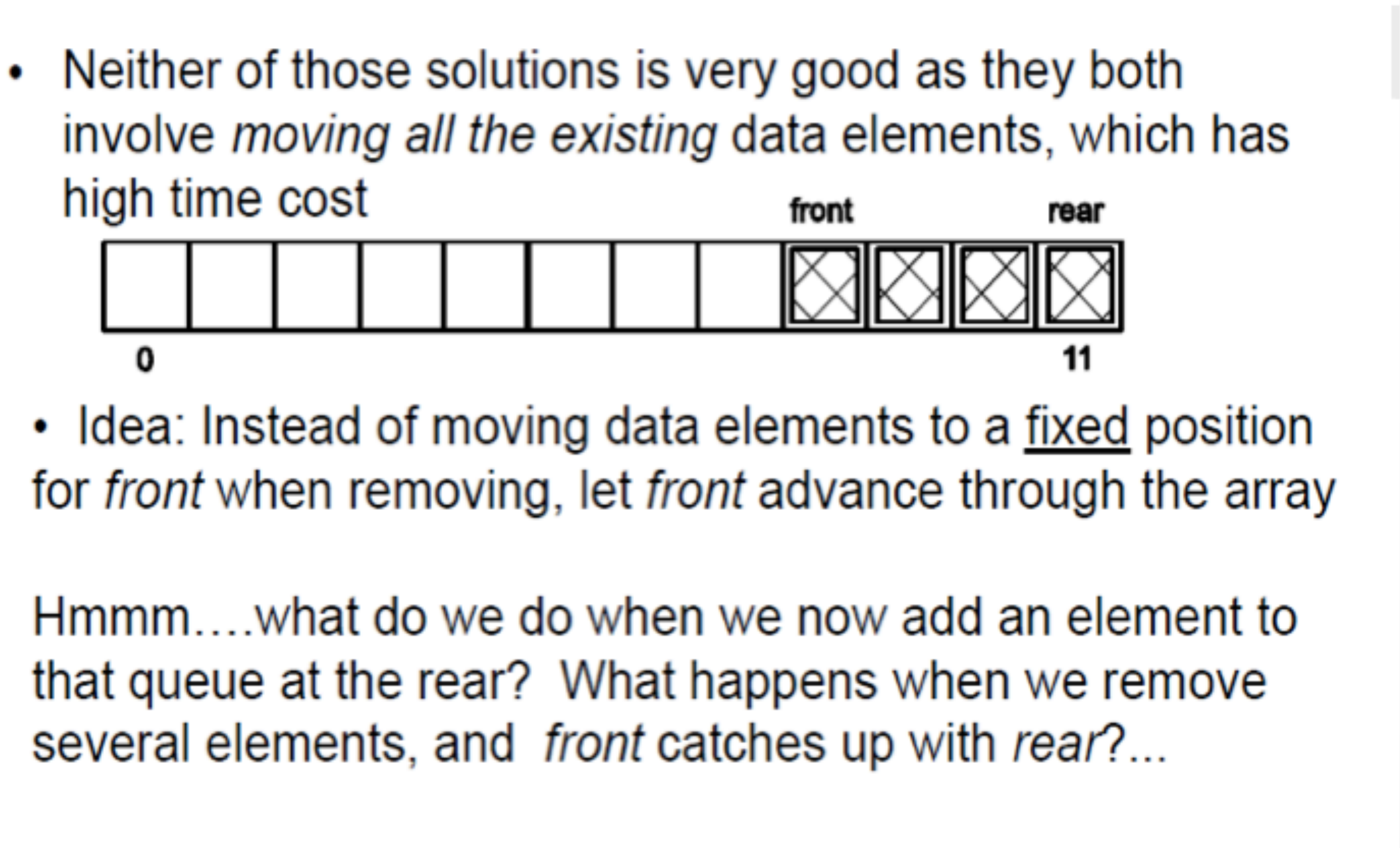
```
array[front] = e;
```

C

```
for(int i= 0; i<front; i++) {
  array[i+1] = array[i];
}
array[front] = e;
```

D None of these are correct

ArrayQueue: another option



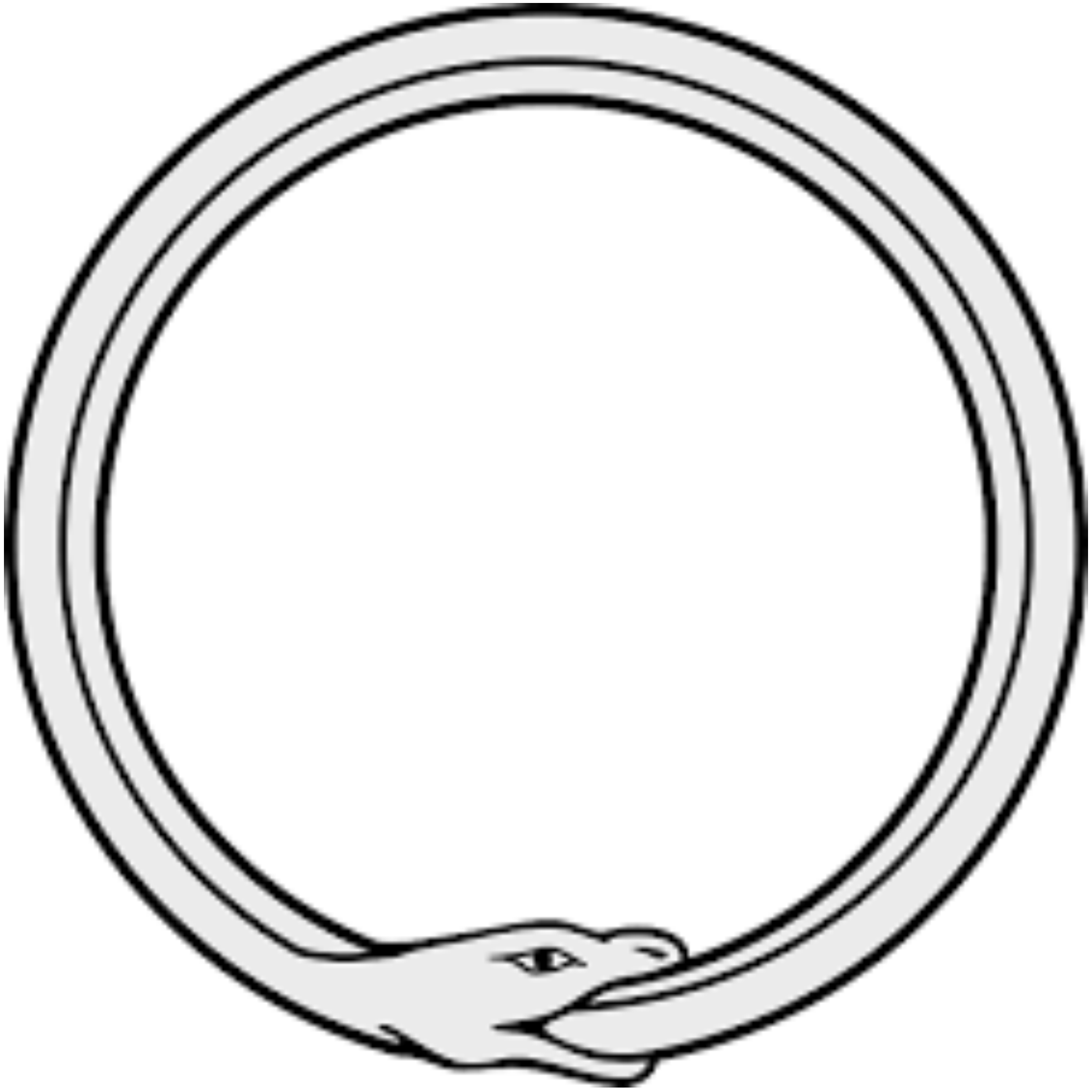
- Neither of those solutions is very good as they both involve *moving all the existing* data elements, which has high time cost

front
rear
0
11

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

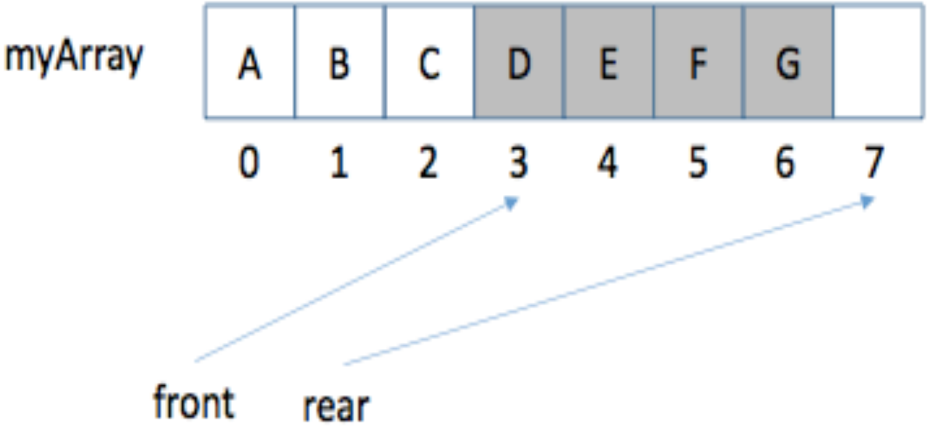
Hmmm….what do we do when we now add an element to that queue at the rear? What happens when we remove several elements, and *front* catches up with *rear*?...

Circular array (Ring buffer)

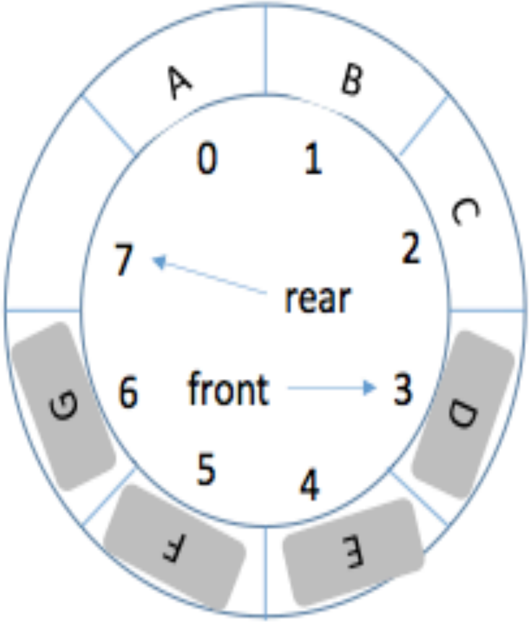


Making a linear array appear circular

front==head
rear == tail



myArray	A	B	C	D	E	F	G	
	0	1	2	3	4	5	6	7

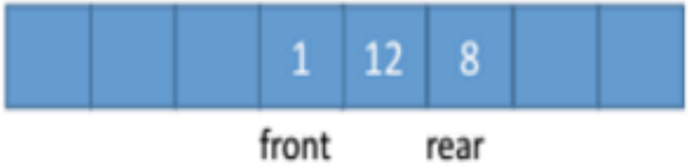


front	3
rear	7

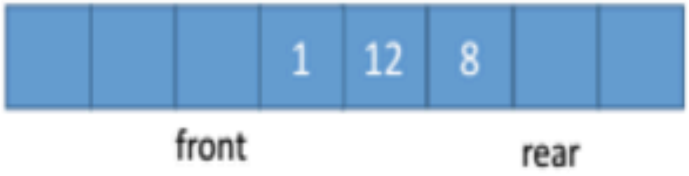
Design decisions: Where do front and rear point?

Which of these choices will work?

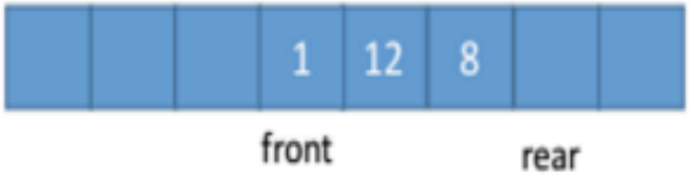
A



B



C

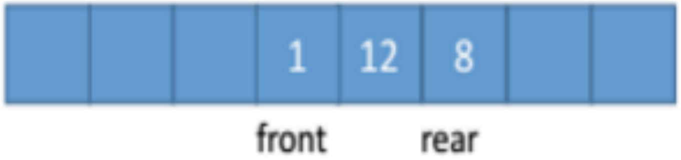


D Any of these could work

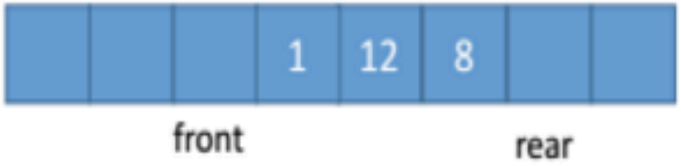
Design decisions: Where do front and rear point?

Which of these choices will work?

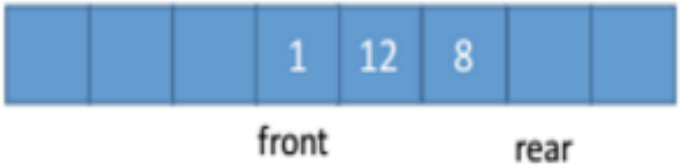
A



B



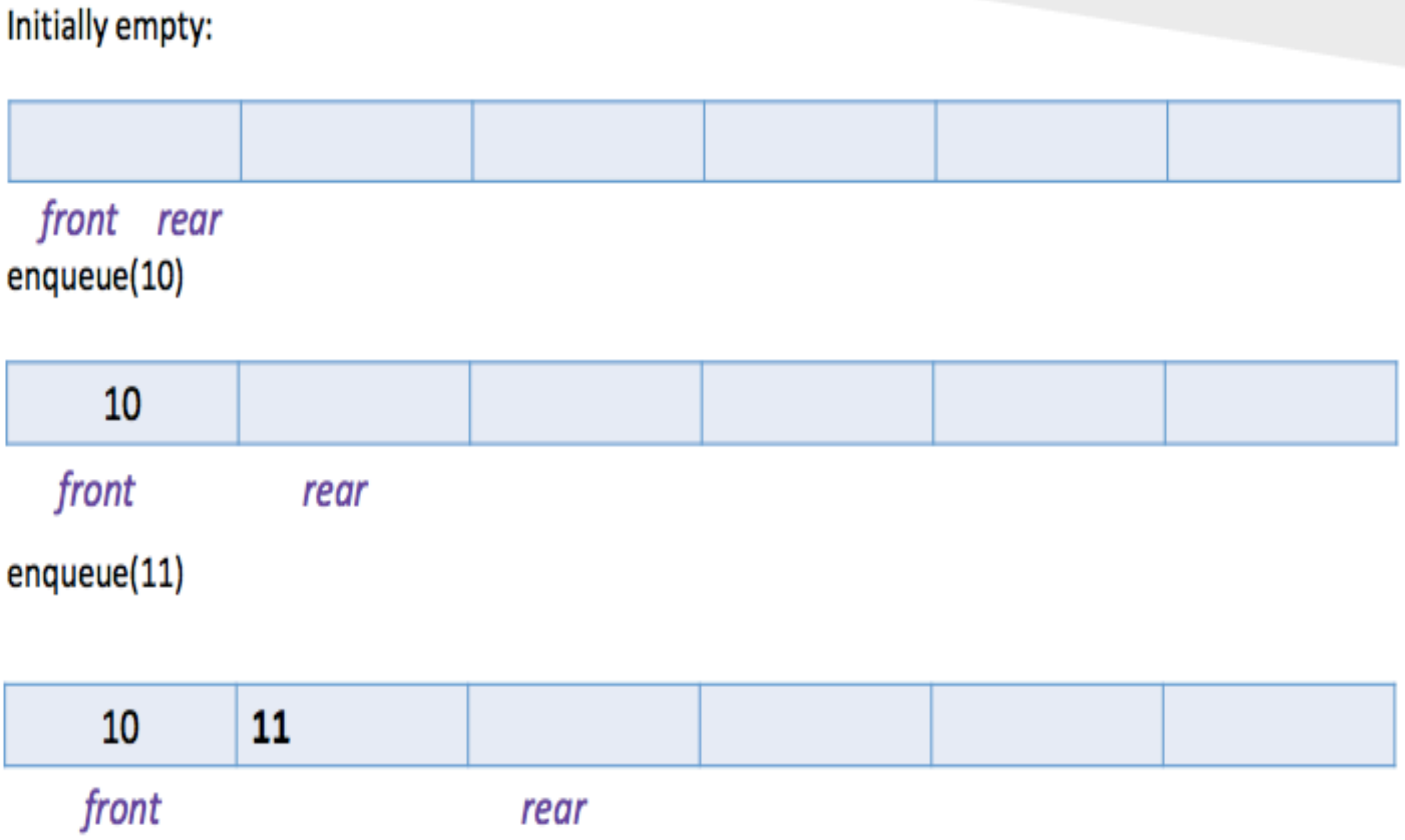
C



It's your choice, but make sure you know what you're doing!

D Any of these could work

Queues using circular Array



Initially empty:

front *rear*

enqueue(10)

10					

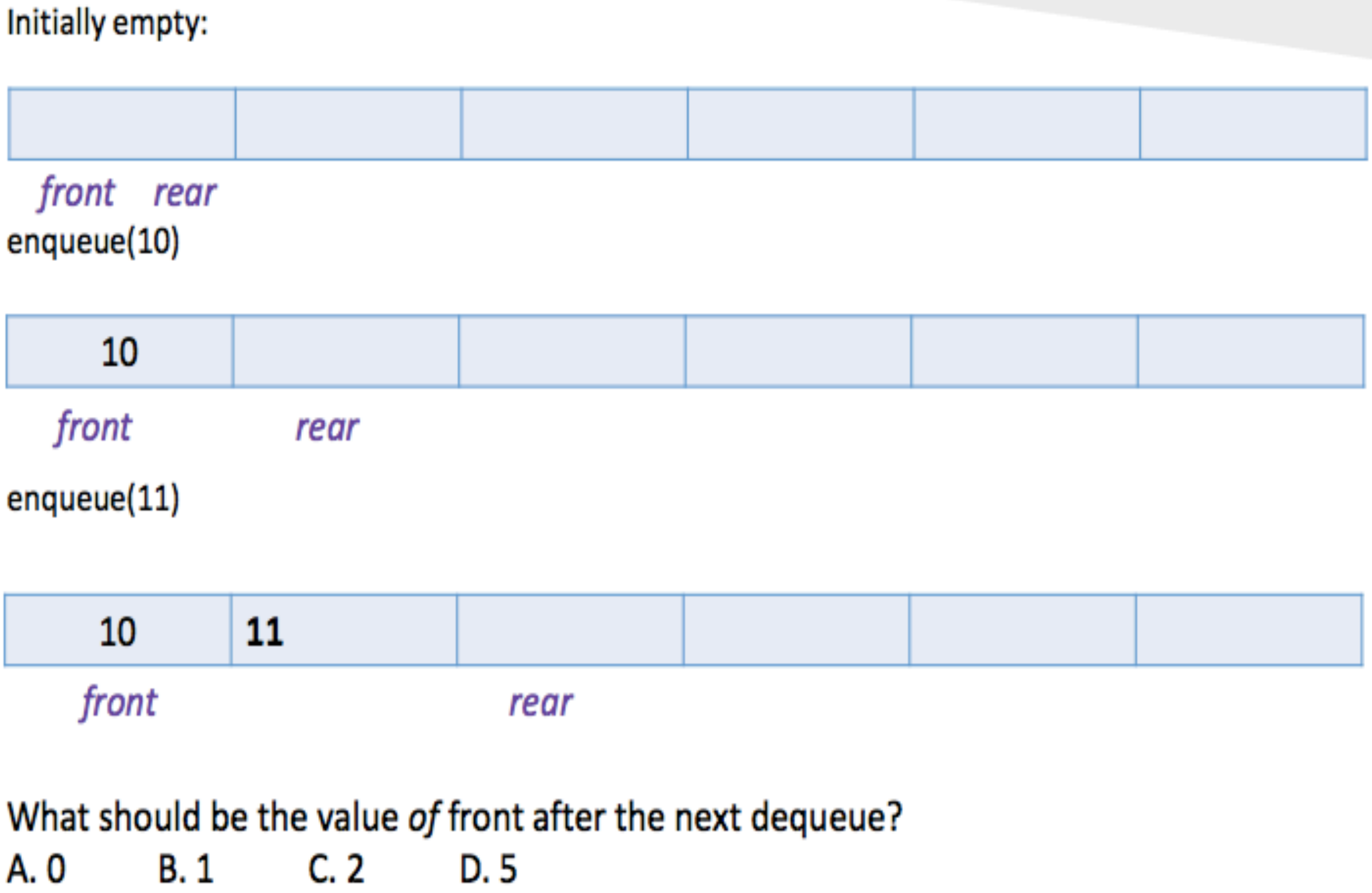
front *rear*

enqueue(11)

10	11				

front *rear*

Queues using circular Array



Initially empty:

front *rear*

enqueue(10)

10					

front *rear*

enqueue(11)

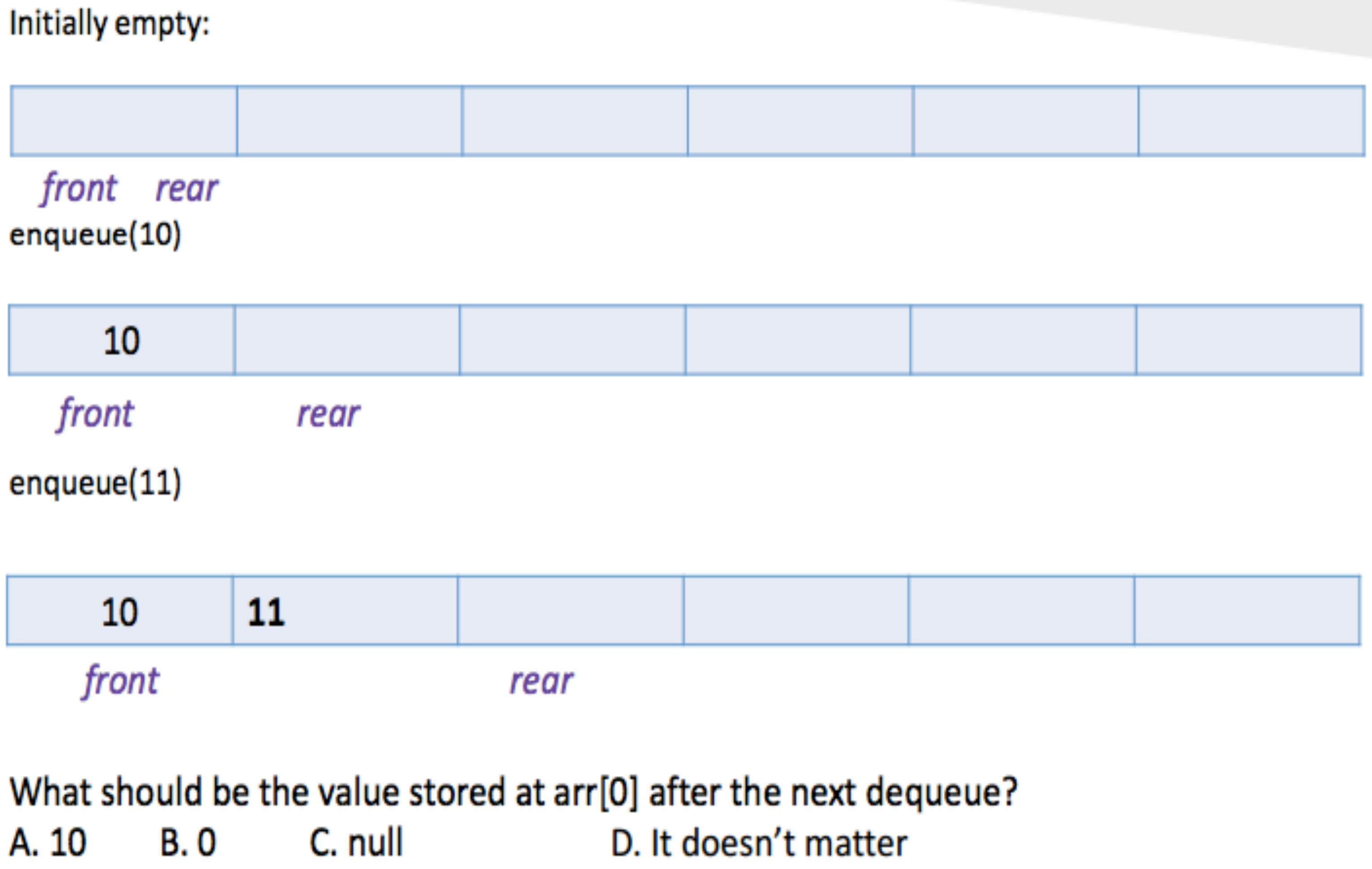
10	11				

front *rear*

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

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

Queues using circular Array



Initially empty:

front *rear*					

enqueue(10)

10					
front	*rear*				

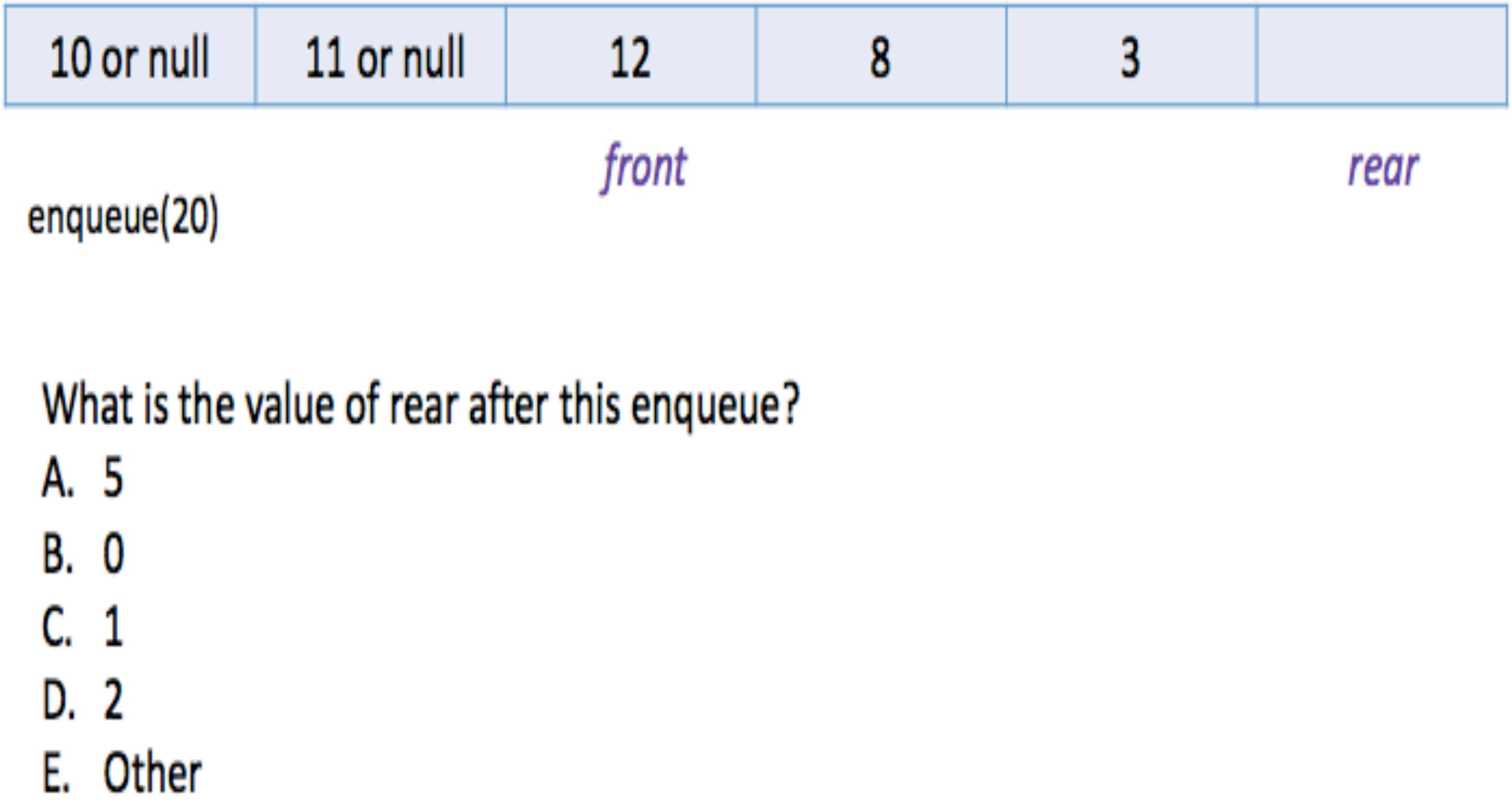
enqueue(11)

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 Array



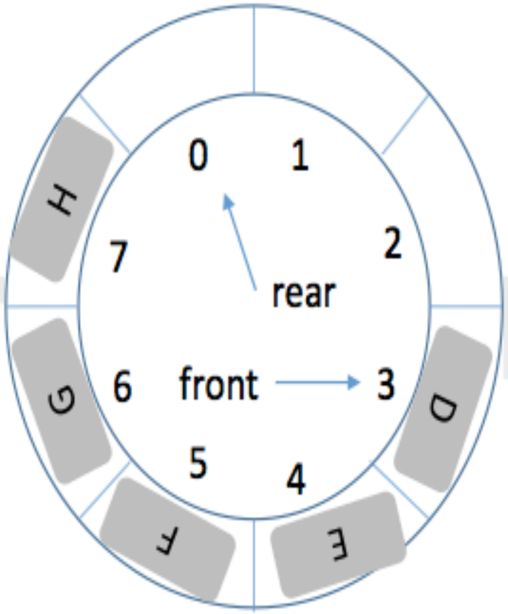
10 or null	11 or null	12	8	3	
		front			*rear*

enqueue(20)

What is the value of rear after this enqueue?

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

```
public E dequeue(){
  // potential issue if empty,
  // for now, assume not empty
  size--;
  E e = array[front];
  <YOUR CODE HERE>
  return e;
}
```



Select the correct code to insert from below:

A

```
front++;
if(front == array.length)
  front = 0;
```

B

```
rear = rear-1;
if(rear <0)
  rear = array.length-1;
```

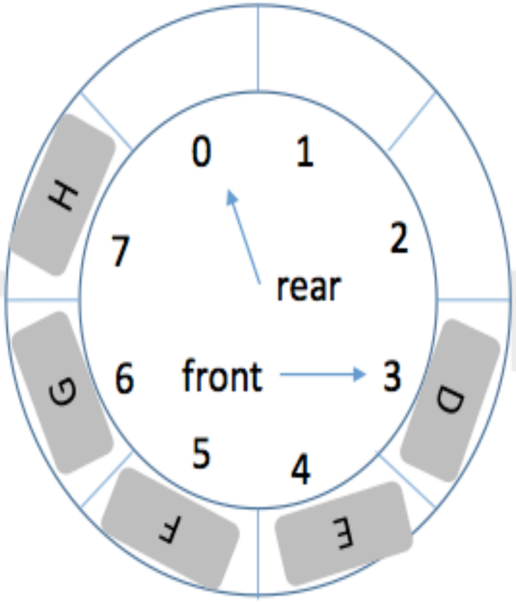
C

```
for(int i= 0; i<rear; i++) {
   array[i] = array[i+1];
}
rear = rear -1;
if(rear < 0)
   rear = array.length-1;
```

D

None of these are correct

```
public void enqueue(E e){
  // potential issue if full,
  // for now, assume not full
    <YOUR CODE HERE>
  size++;
}
```



Select the correct code to insert from below:

A

```
rear++;
if(rear == array.length)
  rear = 0;
array[rear] = e;
```

B

```
array[rear] = e;
rear++;
```

C

```
for(int i= front; i<rear; i++) {
   array[i] = array[i+1];
}
array[rear] = e;
front--;
```

D

None of these are correct

Double-ended queue

- **a double-ended queue** is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front or back.
  - abbreviated to deque, pronounced “deck”.

- What is a good data structure to implement it?

- An input-restricted deque: deletion can be made from both ends, but insertion can be made at one end only.
- An output-restricted deque: can be made at both ends, but deletion can be made from one end only.