Data Structures & Algorithms (CS-212)

Queues

Queue

- A queue is a container of elements that are inserted and removed according to the *first-in first-out* (*FIFO*) principle.
- Elements can be inserted in a queue at any time, but only the element that has been in the queue the longest can be removed at any time.
- Queues have two ends:
  - Elements are added at one end.
  - Elements are removed from the other end.
- We usually say that elements enter the queue at the *rear* and are removed from the *front*.

Queue Operations

enqueue(e): Insert element e at the rear of the queue.

dequeue(): Remove element at the front of the queue; an error occurs if the queue is empty.

front(): Return, but do not remove, a reference to the front element in the queue; an error occurs if the queue is empty.

The queue ADT also includes the following supporting member functions:

size(): Return the number of elements in the queue.

empty(): Return true if the queue is empty and false otherwise.

Example 5.4: *The following table shows a series of queue operations and their effects on an initially empty queue, Q, of integers.*

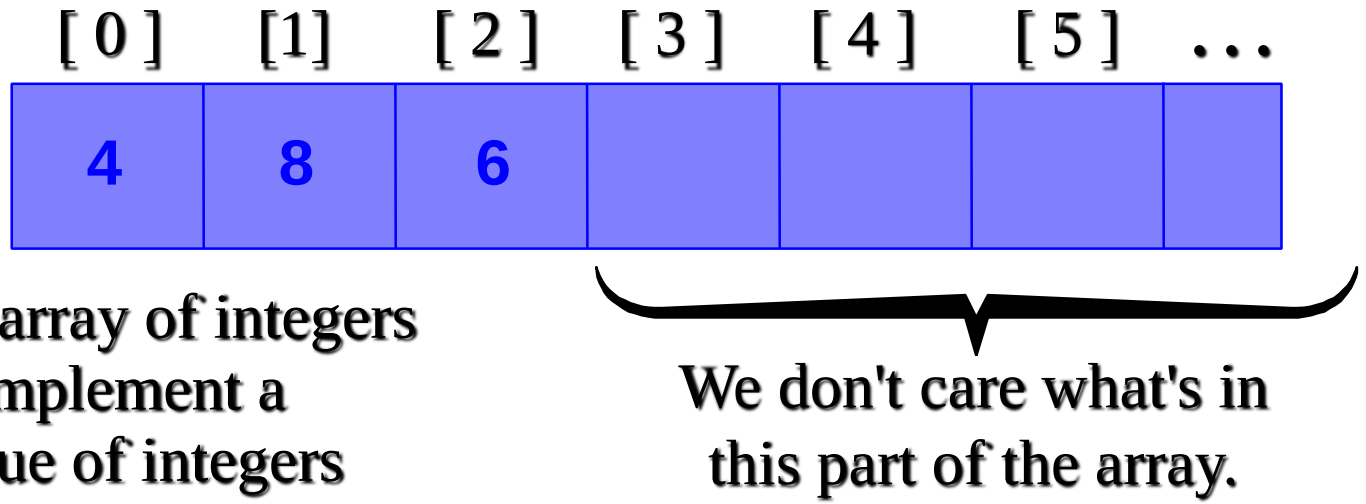
Operation	*Output*	*front* ← *Q* ← *rear*
enqueue(5)	–	(5)
enqueue(3)	–	(5,3)
front()	*5*	(5,3)
size()	*2*	(5,3)
dequeue()	–	(3)
enqueue(7)	–	(3,7)
dequeue()	–	(7)
front()	*7*	(7)
dequeue()	–	()
dequeue()	*"error"*	()
empty()	*true*	()

Implementing Queue ADT: Array Queue

- Keep track of the number of elements in the queue, `size`.
- Enqueue at the back of the array (`size`).
- Dequeue at the front of the array (index 0).

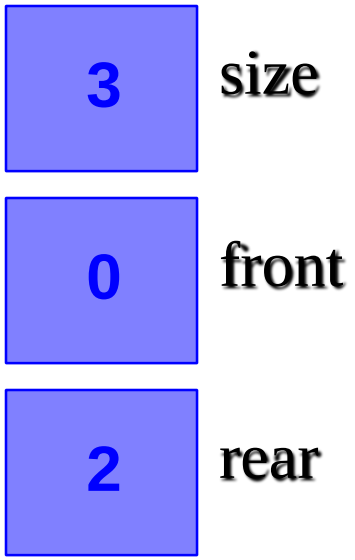
Array Implementation

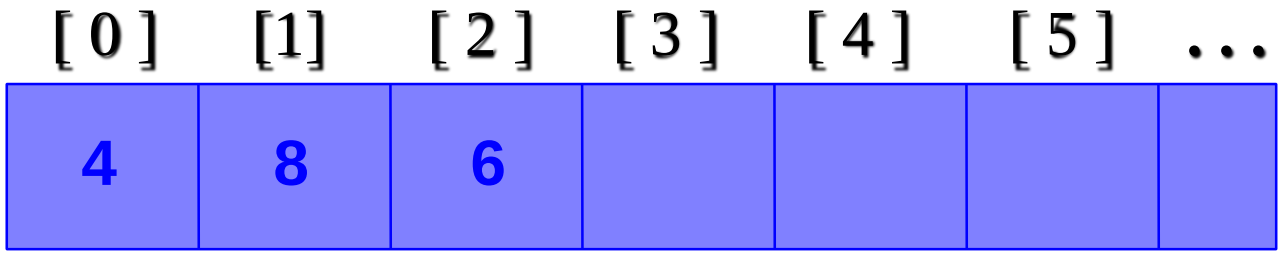
- A queue can be implemented with an array
- Queue contains the integers 4 (at the front), 8 and 6 (at the rear).



Array Implementation

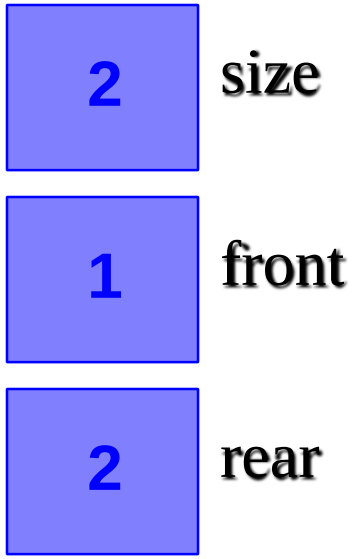
- Keeps track of the number of items in the queue
- Index of the first element i.e. front
- Index of the last element i.e. rear

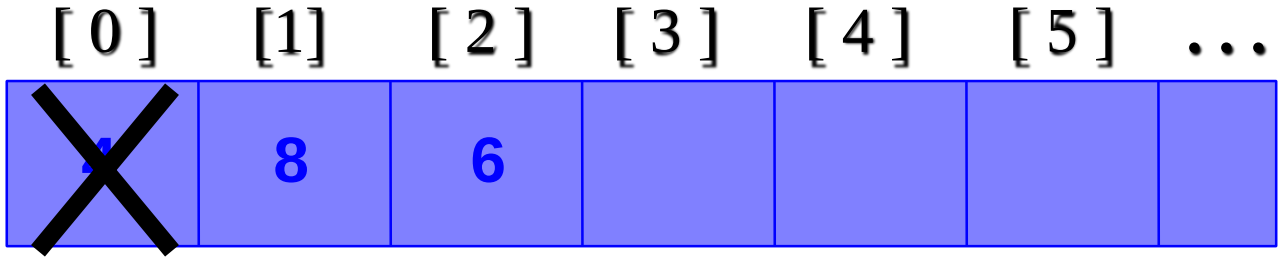




A Dequeue Operation

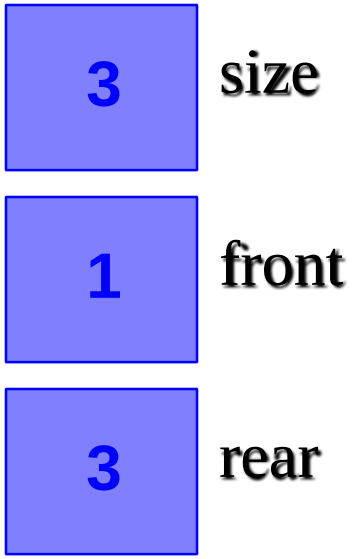
- When an element leaves the queue, size is decremented, and front changes, too.

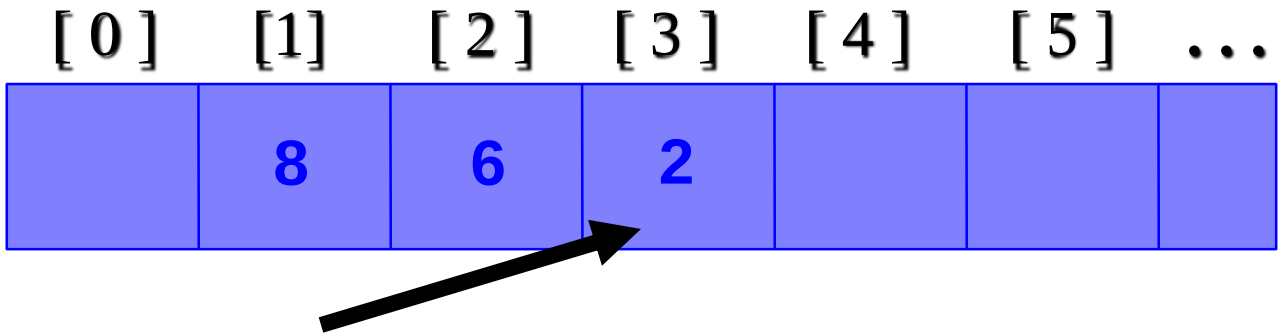




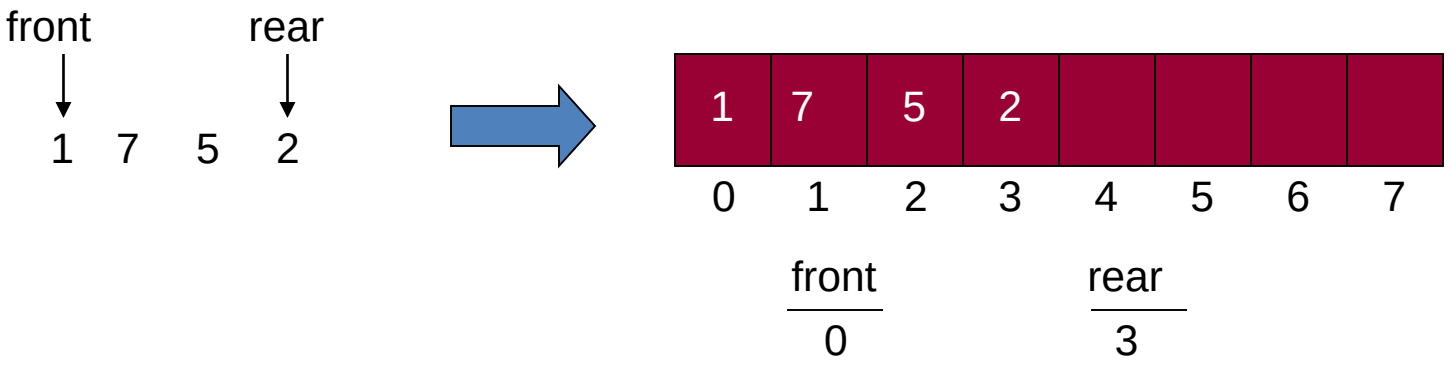
An Enqueue Operation

- When an element enters the queue, size is incremented, and rear changes, too.



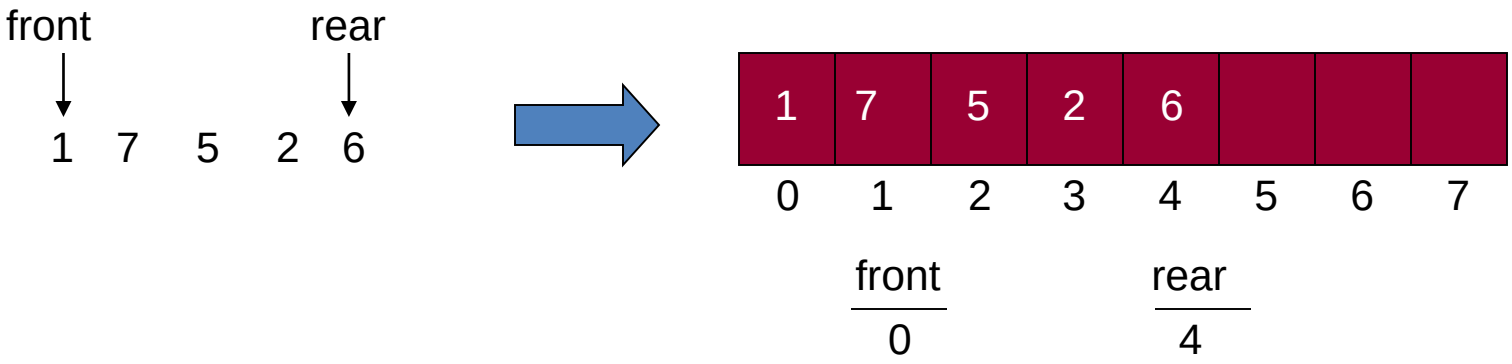


Queue using Array



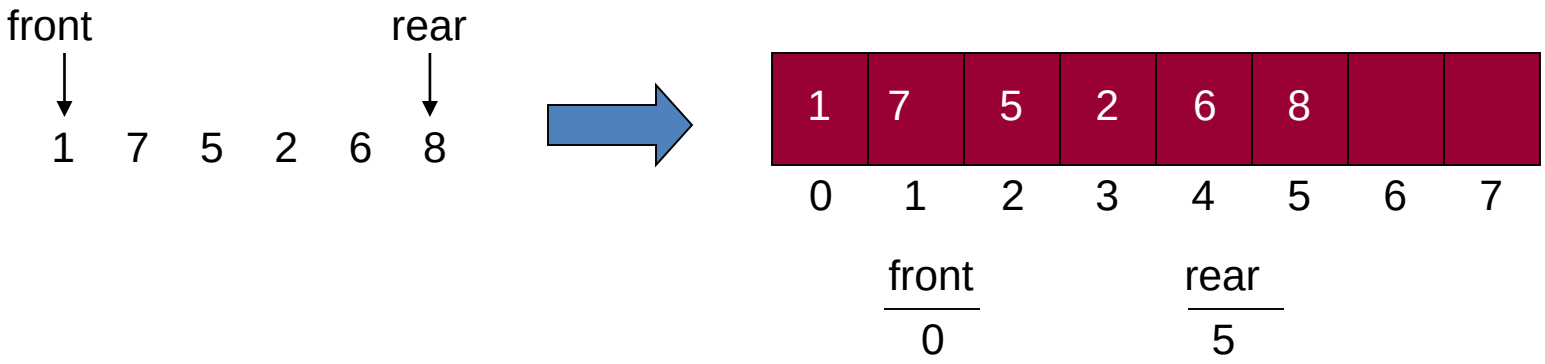
Queue using Array

enqueue(6)



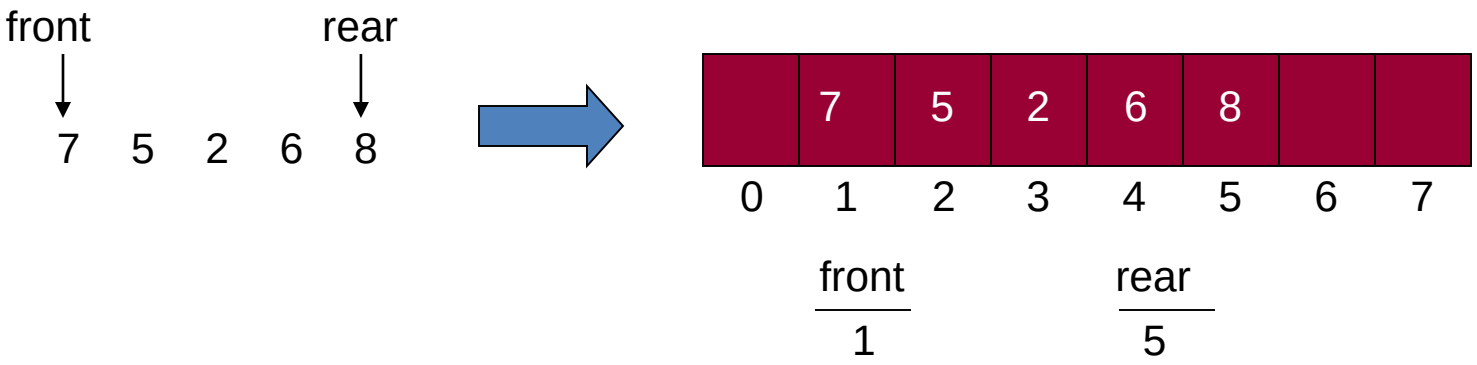
Queue using Array

enqueue(8)



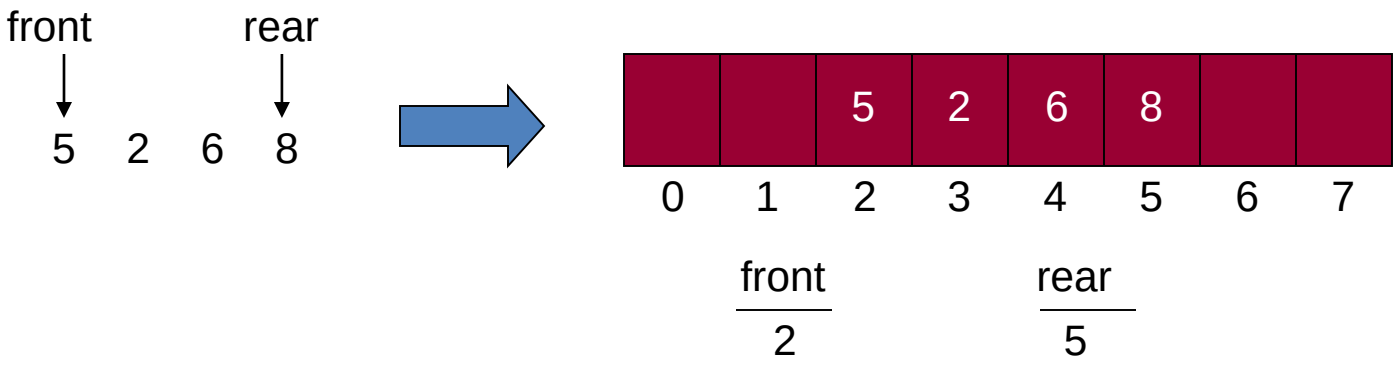
Queue using Array

dequeue()



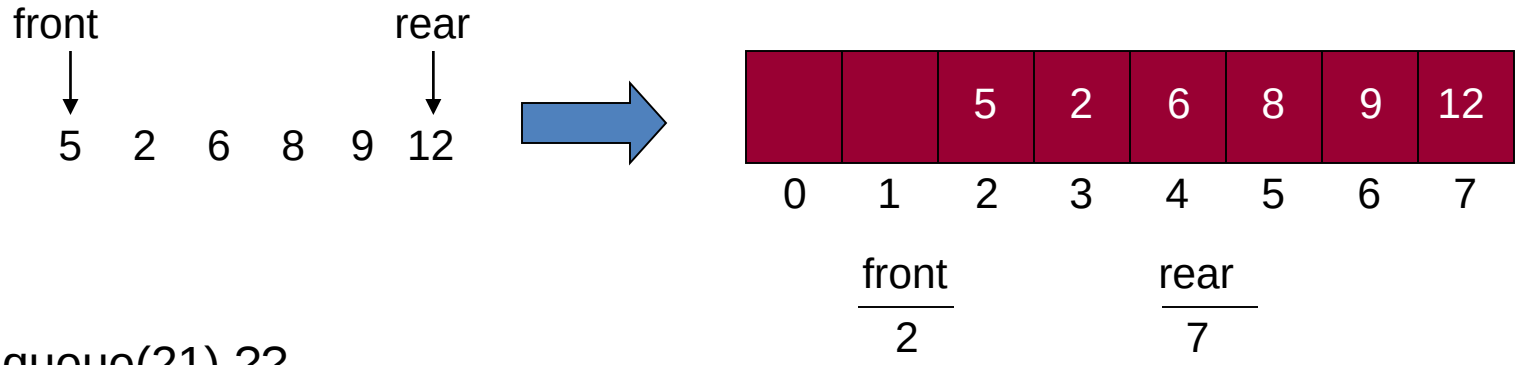
Queue using Array

dequeue()



Queue using Array

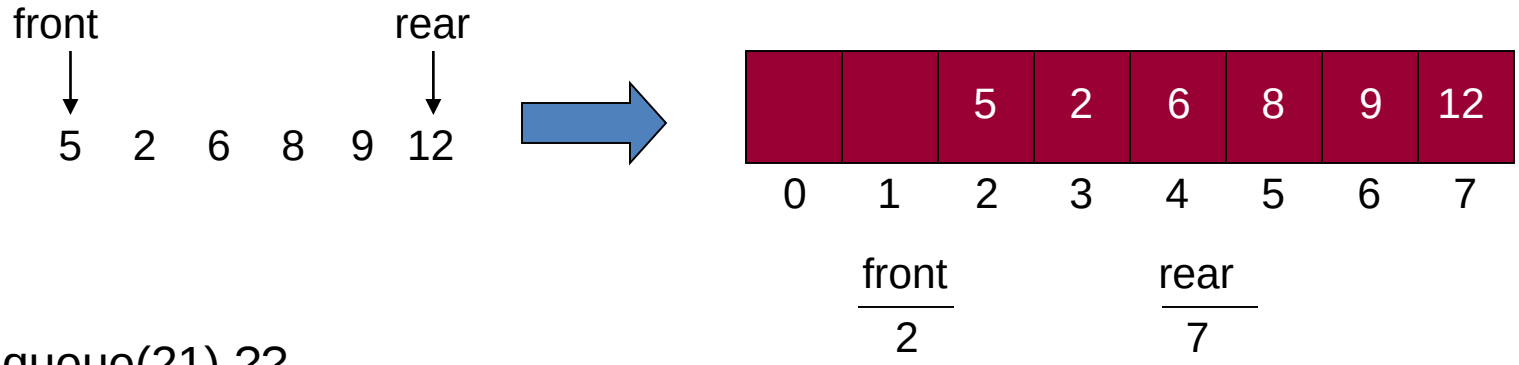
enqueue(9)
enqueue(12)



enqueue(21) ??

Queue using Array

enqueue(9)
enqueue(12)



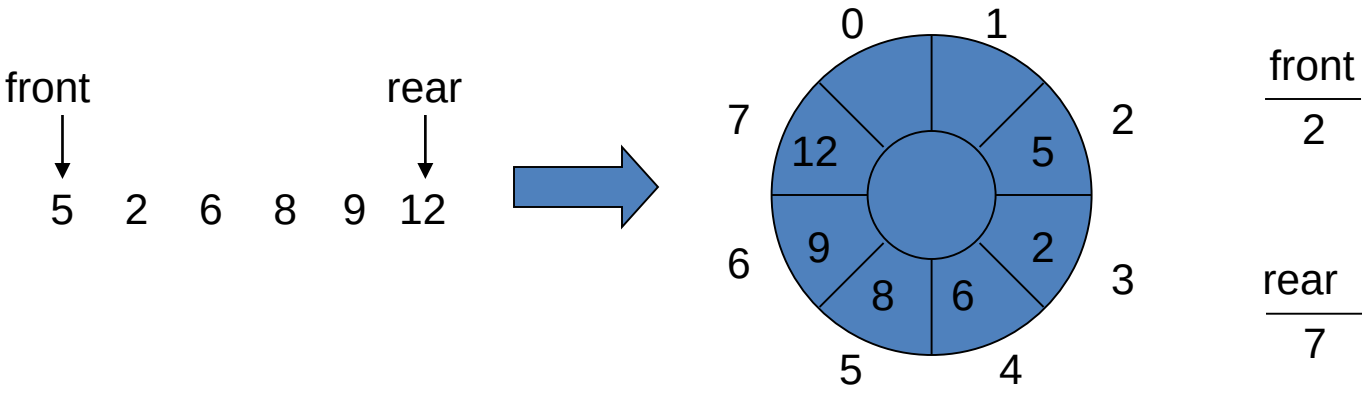
enqueue(21) ??

Queue using Array

- We have inserts and removal running in constant time but we created a new problem.
- Cannot insert new elements even though there are two places available at the start of the array.
- Solution: allow the queue to “wrap around”.

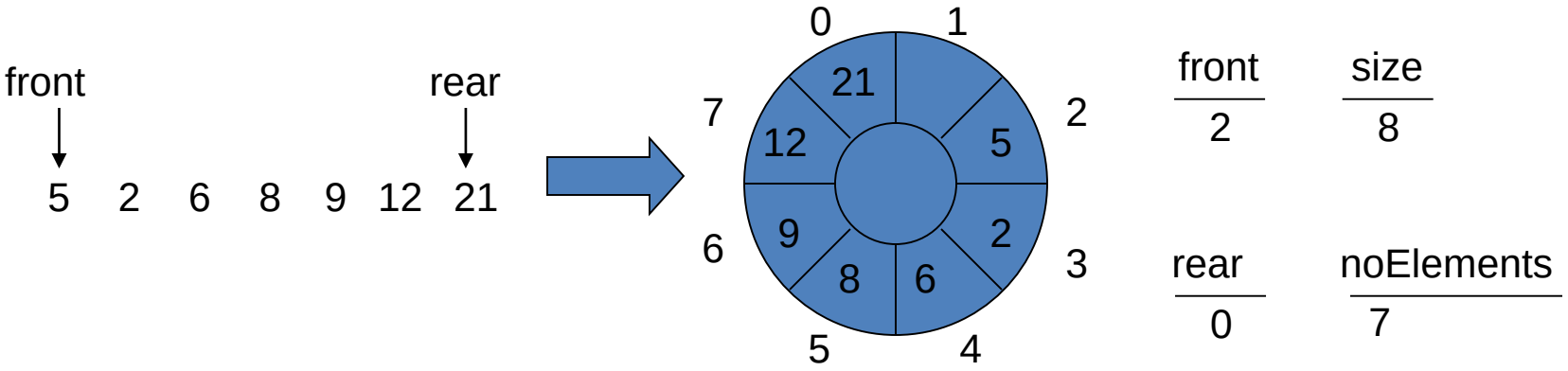
Queue using Array

- Basic idea is to picture the array as a *circular array*.



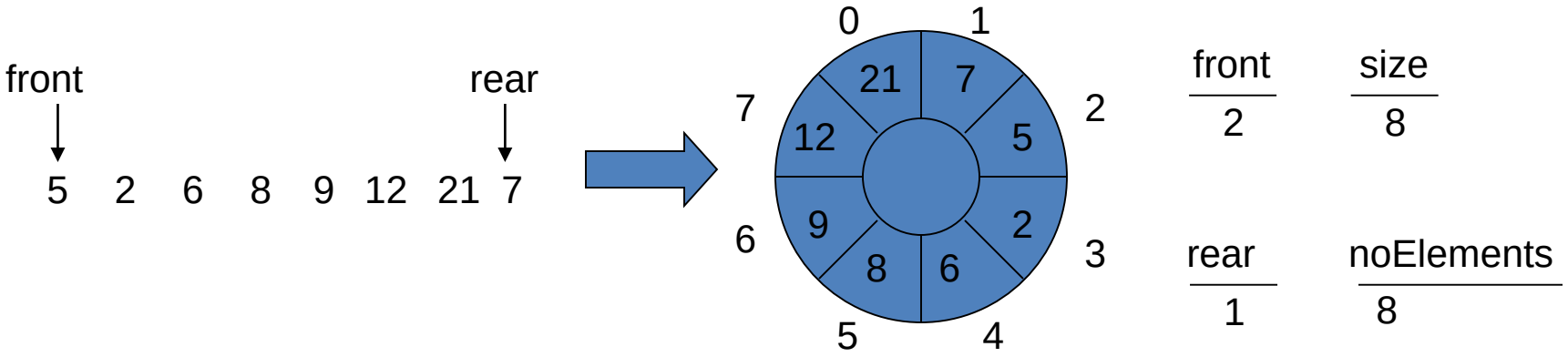
Queue using Array

enqueue(21)



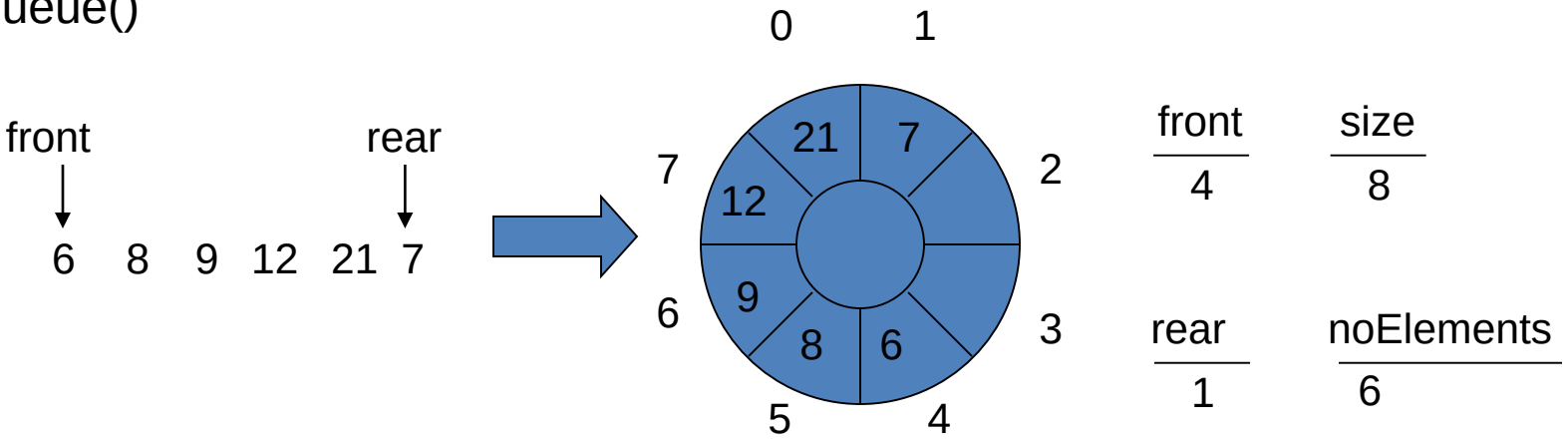
Queue using Array

enqueue(7)



Queue using Array

dequeue()
dequeue()



Using an Array in a Circular Way

- noElemenets = 0
- front = rear = 0
- Enqueue
  - Increment noElemenets
  - Increment rear
- Dequeue
  - Decrement noElemenets
  - Increment front

Using an Array in a Circular Way

- Enqueuing and dequeueing a single element N times causes an out of bounds error

Use Modulus

- Instead of incrementing f and r, use modulus operator
  - (f + 1) mod N
  - (r + 1) mod N

Modulus

- 14 mod 7 = 0
- 15 mod 7 = 1
- 6 mod 7 = 6
- 7 mod 7 = 0

Using an Array in a Circular Way

- noElemenets = 0
- front = rear = 0
- Enqueue
  - Increment noElemenets
  - Increment rear [(rear+1) mod N]
- Dequeue
  - Decrement noElemenets
  - Increment front [(front+1) mod N]

IMPLEMENTATION

Queue Implementation

```
#define MAX_SIZE 20
class Queue {
private:
        int myqueue[MAX_SIZE]
        int front;
        int rear;
        int noElements;
public:
        Queue()
        {
                front = -1;
                rear = -1;
                noElements = 0;
        }
        bool isFull();
        void enqueue(int x);
        int dequeue();
        bool isEmpty()
};
```

enqueue

```
void Queue::enqueue(int x)
{
    if (front==-1)
        front =0;
    rear = (rear+1)%MAX_SIZE;
    myqueue[rear] = x;
     noElements = noElements+1;
}
```

dequeue

```
int Queue::dequeue()
{
     int x = myqueue[front];
    front = (front+1)%MAX_SIZE;
    noElements = noElements-1;
    return x;
}
```

```
int Queue::isFull()
{
    return noElements == size;
}

int Queue::isEmpty()
{
   return noElements == 0;
}
```

Lecture content adapted from Michael T. Goodrich textbook, chapters 5.