CS 112 – Introduction to Computing II

Wayne Snyder
Computer Science Department
Boston University

Today

- Queues
- Implementing a queue using a Circular (or Ring) Buffer;
- Deques
- Priority Queues
- Reading: Wikipedia article on "Circular Buffers"

Next:

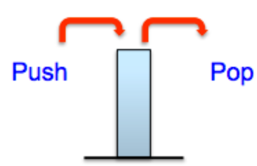
- Analysis of Algorithms: How to measure the running time of algorithms
- Iterative sorting: Selection Sort and Insertion Sort

Computer Science

Queue ADT

Computer Science

The **Queue ADT** is a simple variant of a stack which makes a simple change which in fact changes everything: instead of moving items in and out of the same "end" of the list, as in a stack:



Instead you use different ends of the list:

2

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office!), I'll only give a brief example:

3

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

enqueue(5);

4

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

enqueue(5);
enqueue(7);

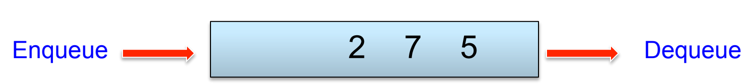
5

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

enqueue(5);
enqueue(7);
enqueue(2);



6

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

```
enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();
```

Enqueue → | 2 7 | → Dequeue

k = 5

7

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

```
enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();
enqueue(8);
```

Enqueue → | 8 2 7 | → Dequeue

k = 5

8

Queue ADT

Computer Science

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I'll only give a brief example:

```
enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();
enqueue(8);
enqueue( dequeue() )
```

Enqueue → | 7 8 2 | → Dequeue

k = 5

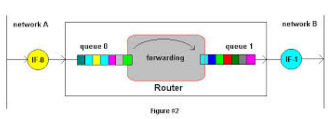
9

Queue ADT

Computer Science

Queues occur all the time, in real life:

And in computer systems (CPUs and Networks):



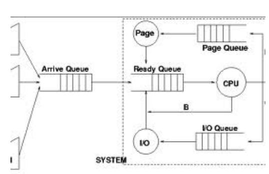


Figure 1: Simple Operating System

In fact, anywhere where one service is desired by many, and must be fairly distributed... there is a whole branch of math called "queueing theory" which you will learn about in CS 237 and CS 350.....

10

Queue ADT

Computer Science

The **informal** interface for a Queue is similar to that for a stack:

`public void enqueue(int n)` -- Insert n at the read of the queue

`public int dequeue()` -- Remove the integer at the front of the queue and return it

`public int peek()` -- Return the number at the front of the queue without removing it

`public int size()` -- Return the number of integers in the queue

`public boolean isEmpty()` – Return true if the queue is empty and false otherwise

Array-based Implementation of Queues

Computer Science

The Java Interface (subject of today's lab) for such an ADT is as follows:

```
// Queueable Interface

public interface Queueable {
      void enqueue(int n);      // insert at the rear of the queue
      int dequeue();            // Remove and return head of queue
      int peek();               // Return head of queue without removing
      boolean isEmpty();
      int size();               // returns number of integers in queue
}
```

How to implement this with arrays?

12

Array-based Implementation of Integer Queues

Computer Science

To implement an array-based queue for ints, here is the **first thing** you might think of.....

	9	8	7	6	5	4	3	2	1	0
A:								2	9	5
							↑ next			↑ front

```
void enqueue(int k) {
   A[next] = k;
   ++next;
}

int size() {
   return (next – front);
}
```

```
int dequeue() {
   int temp = A[front];
   ++front;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

13

Array-based Implementation of Integer Queues

Computer Science

But there is an obvious problem, and not so trivial..... **running off the end** of the array!

	9	8	7	6	5	4	3	2	1	0
A:		0	1	-3	7	91	25	2	9	5
	↑ next			↑ front						

```
void enqueue(int k) {
    A[next] = k;
    ++next;
}
int size() {
   return (next – front);
}
```

```
int dequeue() {
   int temp = A[front];
   ++front;
   return temp;
}

Boolean isEmpty() {
   return (size() == 0);
}
```

14

Array-based Implementation of Integer Queues

What solutions could we come up with for this problem?

Well, there are several:

Bad: Resize the array so you don't run off the end. But then your array grows and grows and grows!

Good: Each time you dequeue, shift all the data over (similarly with how a queue is managed in Starbucks: when the person at the head of the line leaves, everyone moves up!). A natural solution, but if the queue is very large, each dequeue takes a long time, since you have to touch every data item and move it.

15

Array-based Implementation of Integer Queues

So, if you have:

	9	8	7	6	5	4	3	2	1	0
A:	0	0	2	-3	7	91	25	5	1	-3
							next			front

And you dequeue the -3, , you need to shift the queue members to the right (towards 0) one slot:

	9	8	7	6	5	4	3	2	1	0
A:	0	0	2	-3	7	91	25	5	5	1
								next		front

16

Array-based Implementation of Integer Queues

Computer Science

Good: Each time you dequeue, shift all the data over (similarly with how a queue is managed in Starbucks: when the person at the head of the line leaves, everyone moves up!). A natural solution, but if the queue is very large, each dequeue takes a long time, since you have to touch every data item and move it.

Problem: For EVERY dequeue, you have to move EVERY number; we would like to avoid constantly moving the items….. so:

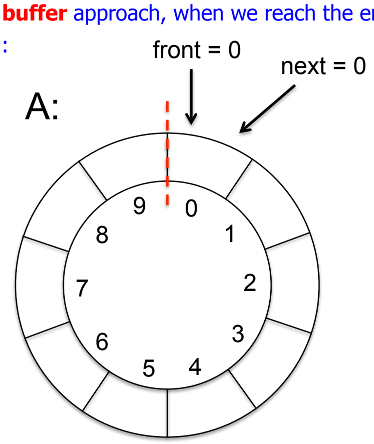
Best: Consider the array to be in a circle, with each end "glued" together, so that you never run off the array…..

17

Array-based Implementation of Queues

Computer Science

In the **ring or circular buffer** approach, when we reach the end of the array we wrap around to the beginning:



In the **fill count** version of circular buffer, we keep track of the number of elements:

size = 0

```
int size = 0;
int front = 0;
int next = 0;
```

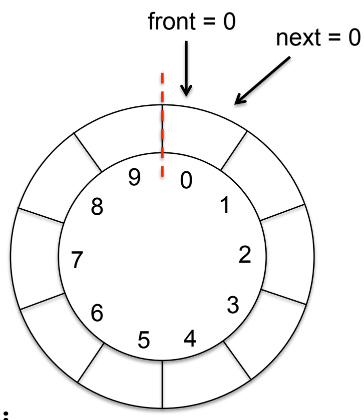
How do we move the pointers front and next around the ring?

18

Array-based Implementation of Queues

Computer Science

The **standard solution** is to wrap around to the beginning of the array, creating a **circular buffer**:




```
int size = 0;
int front = 0;
int next = 0;

// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

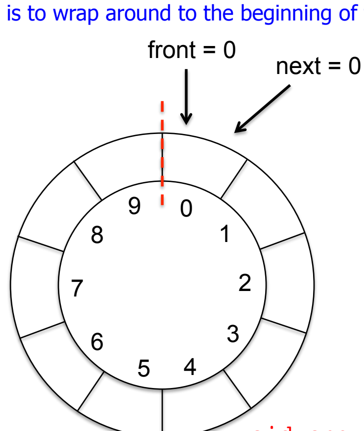
```
next = nextSlot(next);
```

19

Array-based Implementation of Queues

Computer Science

The **standard solution** is to wrap around to the beginning of the array, creating a **circular buffer**:




```
int size = 0;
int front = 0;
int next = 0;

// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

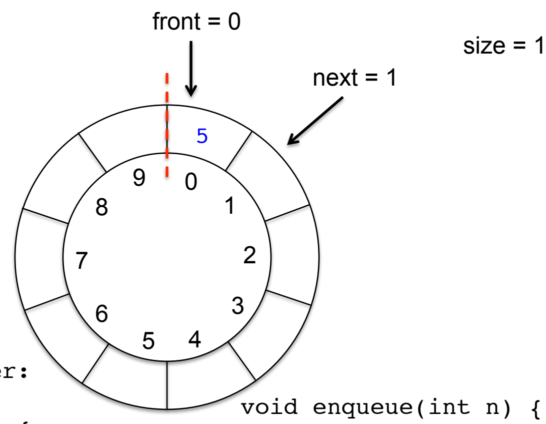
```
void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}
```

20

Array-based Implementation of Queues

Computer Science

```
enqueue(5);
```




```
// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

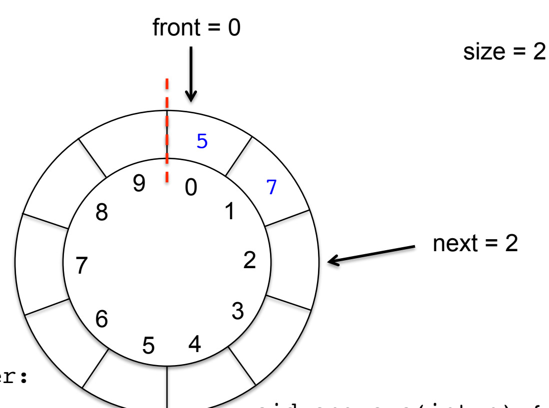
```
void enqueue(int n) {
     A[next] = n;
     next = nextSlot(next);
     ++size;
}
```

21

Array-based Implementation of Queues

Computer Science

```
enqueue(5);
enqueue(7);
```




```
// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

```
void enqueue(int n) {
     A[next] = n;
     next = nextSlot(next);
     ++size;
}
```

22

Array-based Implementation of Queues

Computer Science

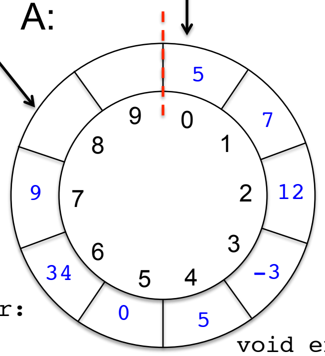
next = 8

front = 0

size = 8

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);
```

A:




```
// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

```
void enqueue(int n) {
     A[next] = n;
     next = nextSlot(next);
     ++size;
}
```

23

Array-based Implementation of Queues

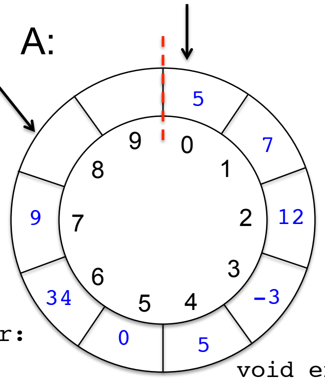
Computer Science

next = 8

front = 0

size = 8

A:




```
int dequeue() {
  int temp = A[front];
  front = nextSlot(front);
  --size;
  return temp;
}
```

```
// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

```
void enqueue(int n) {
     A[next] = n;
     next = nextSlot(next);
     ++size;
}
```

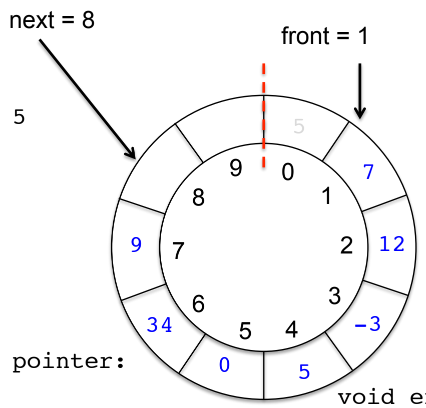
24

Array-based Implementation of Queues

Computer Science

size = 7

```
dequeue() => 5
```




```
int dequeue() {
  int temp = A[front];
  front = nextSlot(front);
  --size;
  return temp;
}
```

```
// To move a pointer:

int nextSlot(int k) {
    return ((k + 1) % A.length);
}
```

```
void enqueue(int n) {
     A[next] = n;
     next = nextSlot(next);
     ++size;
}
```

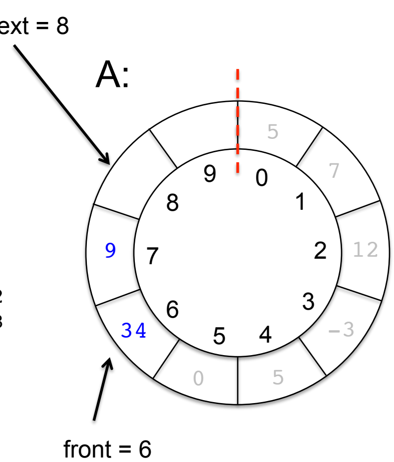
25

Array-based Implementation of Queues

Computer Science

size = 2

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);

dequeue(); => 5
dequeue(); => 7
dequeue(); => 12
dequeue(); => -3
dequeue(); => 5
dequeue(); => 0
```




```
int [] A = new int[10];
int size = 0;
int front = 0; int next = 0;

int nextSlot(int k) {
    return ((k + 1) % A.length);
}

void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}

int size() {
   return size;
}

// can still underflow!
int dequeue() {
   int temp = A[front];
   front = nextSlot(front);
   --size;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

26

Array-based Implementation of Queues

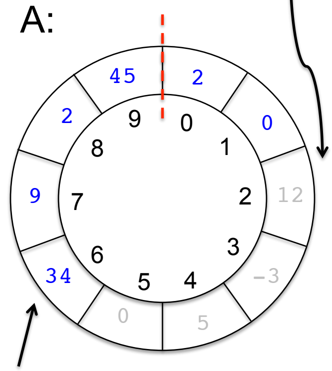
Computer Science

size = 6

next = 2

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);
dequeue(); => 5
dequeue(); => 7
dequeue(); => 12
dequeue(); => -3
dequeue(); => 5
dequeue(); => 0

enqueue(2);
enqueue(45);
enqueue(2);
enqueue(0);

Etc....
```

A:



front = 6

```
int [] A = new int[10];
int size = 0;
int front = 0; int next = 0;

int nextSlot(int k) {
    return ((k + 1) % A.length);
}

void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}

int size() {
   return size;
}

// can still underflow!
int dequeue() {
   int temp = A[front];
   front = nextSlot(front);
   --size;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

27

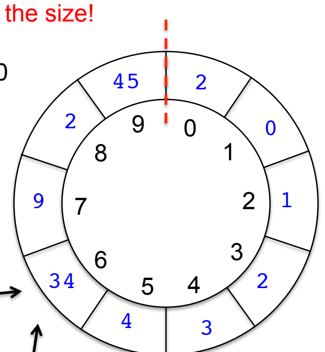
Array-based Implementation of Queues

Computer Science

Note: Can't distinguish full or empty from the pointers alone, that is why we keep track of the size!

size = 10

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);
dequeue(); => 5
dequeue(); => 7
dequeue(); => 12
dequeue(); => -3
dequeue(); => 5
dequeue(); => 0

enqueue(2);
enqueue(45);
enqueue(2);
enqueue(0);
enqueue(1);
enqueue(2);
enqueue(3);
enqueue(4);
```



front = 6

next = 6

```
int [] A = new int[10];
int size = 0;
int front = 0; int next = 0;

int nextSlot(int k) {
    return ((k + 1) % A.length);
}

void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}

int size() {
   return size;
}

// can still underflow!
int dequeue() {
   int temp = A[front];
   front = nextSlot(front);
   --size;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

28

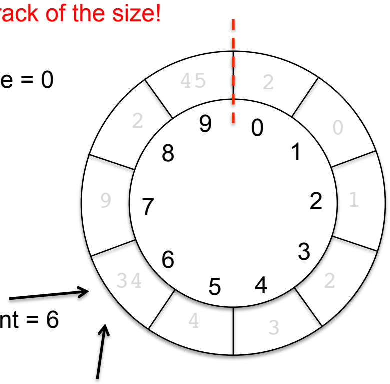
Array-based Implementation of Queues

Boston University Computer Science

Note: Can't distinguish full or empty from the pointers alone, that is why we keep track of the size!

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);
dequeue(); => 5
dequeue(); => 7
dequeue(); => 12
dequeue(); => -3
dequeue(); => 5
dequeue(); => 0

enqueue(2);
enqueue(45);
enqueue(2);
enqueue(0);
enqueue(1);
enqueue(2);
enqueue(3);
enqueue(4);
```

size = 0



front = 6

next = 6

```
int [] A = new int[10];
int size = 0;
int front = 0; int next = 0;

int nextSlot(int k) {
    return ((k + 1) % A.length);
}

void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}

int size() {
   return size;
}

// can still underflow!
int dequeue() {
   int temp = A[front];
   front = nextSlot(front);
   --size;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

29

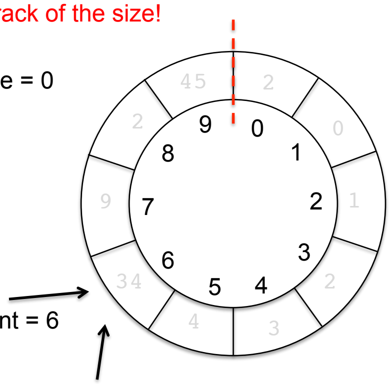
Array-based Implementation of Queues

Boston University Computer Science

Note: Can't distinguish full or empty from the pointers alone, that is why we keep track of the size!

```
enqueue(5);
enqueue(7);
enqueue(12);
enqueue(-3);
enqueue(5);
enqueue(0);
enqueue(34);
enqueue(9);
dequeue(); => 5
dequeue(); => 7
dequeue(); => 12
dequeue(); => -3
dequeue(); => 5
dequeue(); => 0

enqueue(2);
enqueue(45);
enqueue(2);
enqueue(0);
enqueue(1);
enqueue(2);
enqueue(3);
enqueue(4);
```

size = 0



front = 6

next = 6

Can solve overflow by resizing but it can still underflow!

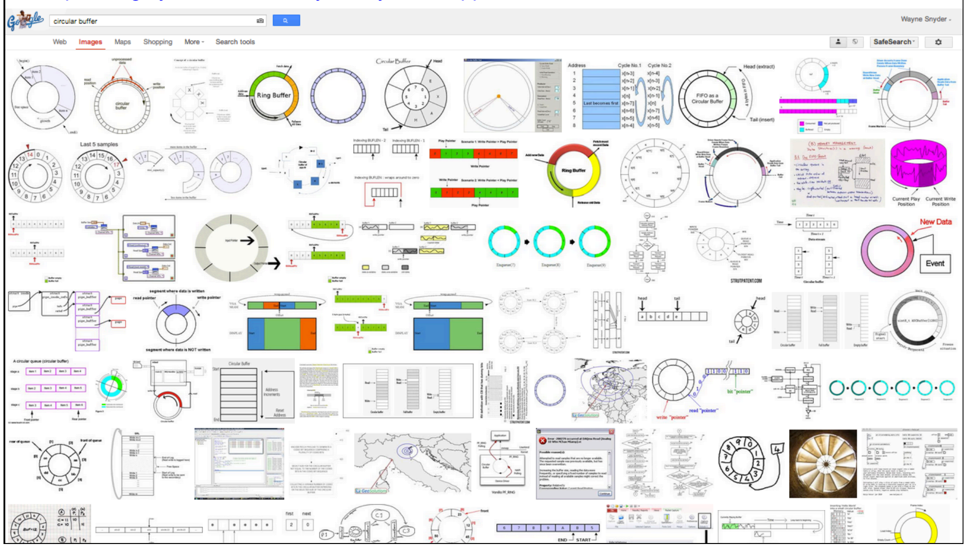
```
int [] A = new int[10];
int size = 0;
int front = 0; int next = 0;

int nextSlot(int k) {
    return ((k + 1) % A.length);
}

void enqueue(int n) {
    A[next] = n;
    next = nextSlot(next);
    ++size;
}

int size() {
   return size;
}

// can still underflow!
int dequeue() {
   int temp = A[front];
   front = nextSlot(front);
   --size;
   return temp;
}

boolean isEmpty() {
   return (size() == 0);
}
```

30

Array-based Implementation of Queues

Computer Science

Circular or ring buffers are the standard technique for implementing queues and buffers in operating systems and many, many other applications!



Queue ADT: Two Important Variations

Computer Science

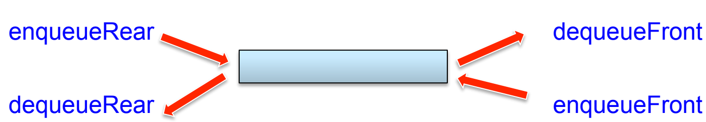
The **Deque ("deck") ADT** is a "double-ended queue" in which you can insert or remove from either end; it is either a queue going in both directions, or two stacks stuck together:

enqueueRear(k): Insert the key k in the rear

dequeueRear(): Remove and return the item from the rear of the list

enqueueFront(k): Insert the key k in the front

dequeueFront(): Remove and return the item from the front of the list



32

Queue ADT: Two Important Variations

The **Priority Queue ADT** is a queue in which the list is always kept ordered; this is useful when elements in the queue have a different need or right for service; the only change is in the enqueue method; they are typically called by different names:

insert(k): Insert the key k in order in the list

getMax(): Remove and return the item in the front of the list

33

Queue ADT: Two Important Variations

The **Priority Queue ADT** is a queue in which the list is always kept ordered; this is useful when elements in the queue have a different need or right for service; the interface is usually defined with somewhat different names for the two basic operations, depending on whether it is a "maxQueue" (ordered so that biggest go to the front) or "minQueue" (smallest go to front).

insert(k): Insert the key k in order in the list (cf. enqueue(k))

getMax() or getMin(): Remove and return the item in the front of the list (cf. dequeue())

34

Priority Queue ADT

Computer Science

insert(5);

35

Priority Queue ADT

Computer Science

insert(5);
insert(7);

36

Priority Queue ADT

Computer Science

insert(5);
insert(7);
insert(2);

insert → 2 5 7 → getMax

37

Priority Queue ADT

Computer Science

insert(5);
insert(7);
insert(2);
int k = getMax();

insert → 2 5 → getMax

k = 7

38

Priority Queue ADT

Computer Science

insert(5);
insert(7);
insert(2);
int k = getMax();
enqueue(8);

insert → | 2 5 8 | → getMax

k = 7

39

Priority Queue ADT

Computer Science

insert(5);
insert(7);
insert(2);
int k = getMax();
insert(8);
insert(getMax())

insert → | 2 5 8 | → getMax

k = 7

40