QUEUES

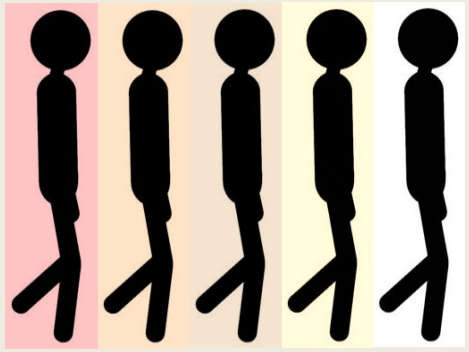
Mahsa Ziraksima
ACIBADEM UNIVERSITY
Fall 2025

SUMMARY

- Performance
- Creating a Queue

QUEUES

- In computer science, a queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.
- By convention, the end of the sequence at which elements are added is called the **back**, **tail**, or **rear** of the queue,
- and the end at which elements are removed is called the **head** or **front** of the queue.

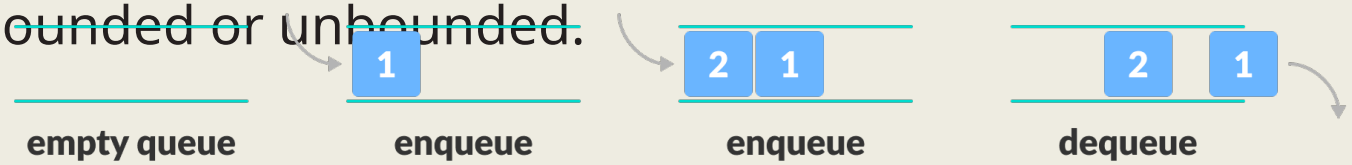


QUEUES

- It is an abstract data type and a linear data structure where you can add items only to the rear and remove them from the front.

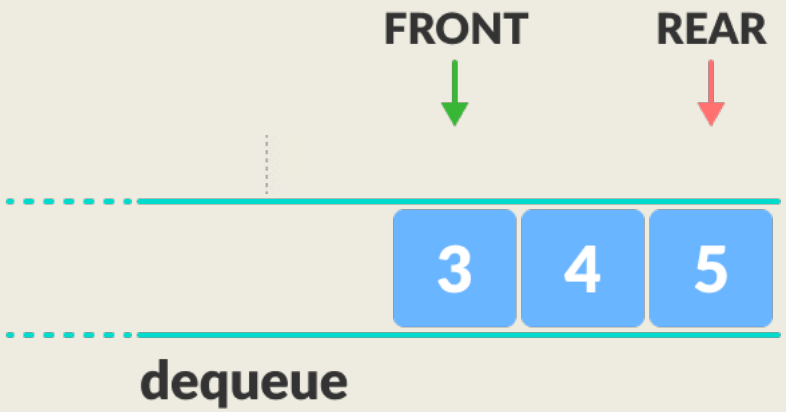
  example of a first- in, first- out (FIFO) data structure

- in which the first item to enter the data structure is the first to come out of it.
- They are limited access data structures.
- They can be bounded or unbounded.



- Its two primary operations:
  - *Enqueueing means adding an item to the rear of a queue,*

QUEUES



- ✓ If front = rear, queue is empty.
- ✓ If rear = n, queue is full.

```python
class Node:
    def __init__(self, data, next=None):
        self.data = data
        self.next = next


class Queue:
    def __init__(self):
        self.front = None
        self.rear = None
        self._size = 0

    def enqueue(self, item):
        self._size += 1
        node = Node(item)
        if self.rear is None:
            self.front = node
            self.rear = node
        else:
            self.rear.next = node
            self.rear = node

    def dequeue(self):
        if self.front is None:
            raise IndexError('pop from empty queue')
        self._size -= 1
        temp = self.front
        self.front = self.front.next
        if self.front is None:
            self.rear = None
        return temp.data

    def size(self):
        return self._size
```

```python
queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print(queue.size())
for i in range(3):
    print(queue.dequeue())
```

```
3
1
2
3
```

QUEUES

```
from queue import Queue

q = Queue()
q.put('a')
q.put('b')
q.put('c')
print(q.qsize())
for i in range(3):
    print(q.get())
```

```
3
a
b
c
```

QUEUES

- Enqueueing and dequeueing are both O(1).
- Accessing an item and searching a queue are both O(n)

Data Structure	Time Complexity							
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)

IMPLEMENTING QUEUE WITH

```
class Queue:
    def __init__(self):
        self.s1 = []
        self.s2 = []

    def enqueue(self, item):
        while len(self.s1) != 0:
            self.s2.append(self.s1.pop())
        self.s1.append(item)
        while len(self.s2) != 0:
            self.s1.append(self.s2.pop())

    def dequeue(self):
        if len(self.s1) == 0:
            raise Exception("Cannot pop from empty queue")
        return self.s1.pop()

queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
for i in range(3):
    print(queue.dequeue())
```

```
1
2
3
```

QUEUES

- Challenge: what would be the results of A, B, and C?

A = 5 B = 10 C = 2 n = 4

Addqueue (A + B)

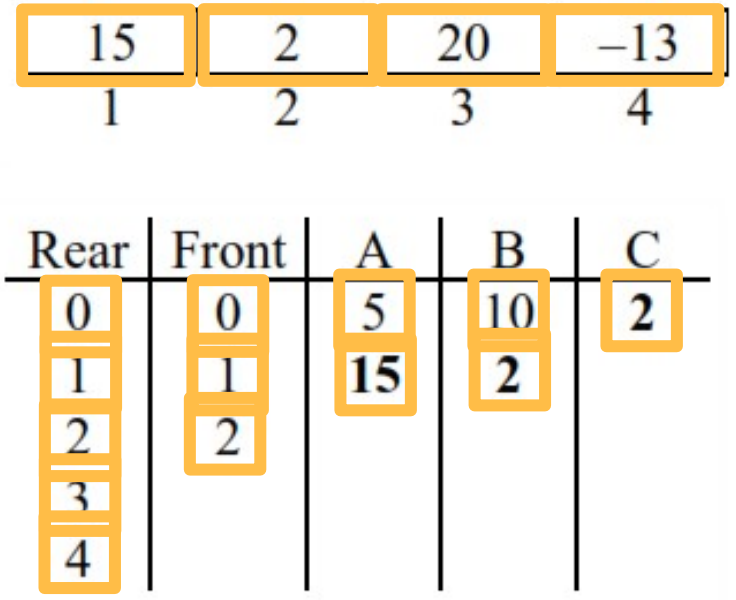
Addqueue (C)

Addqueue (B × C)

A = delqueue ( )

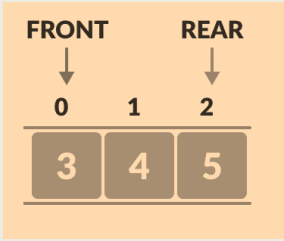
B = delqueue ( )

Addqueue (B – A)

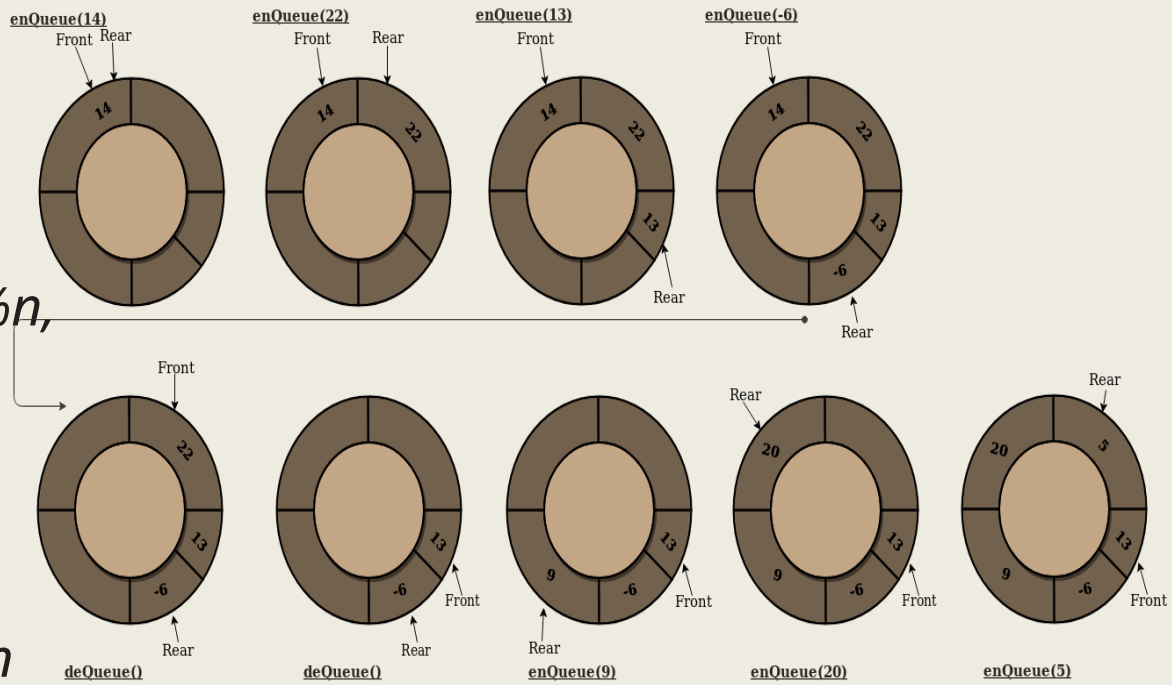


QUEUES

- Linear queue:

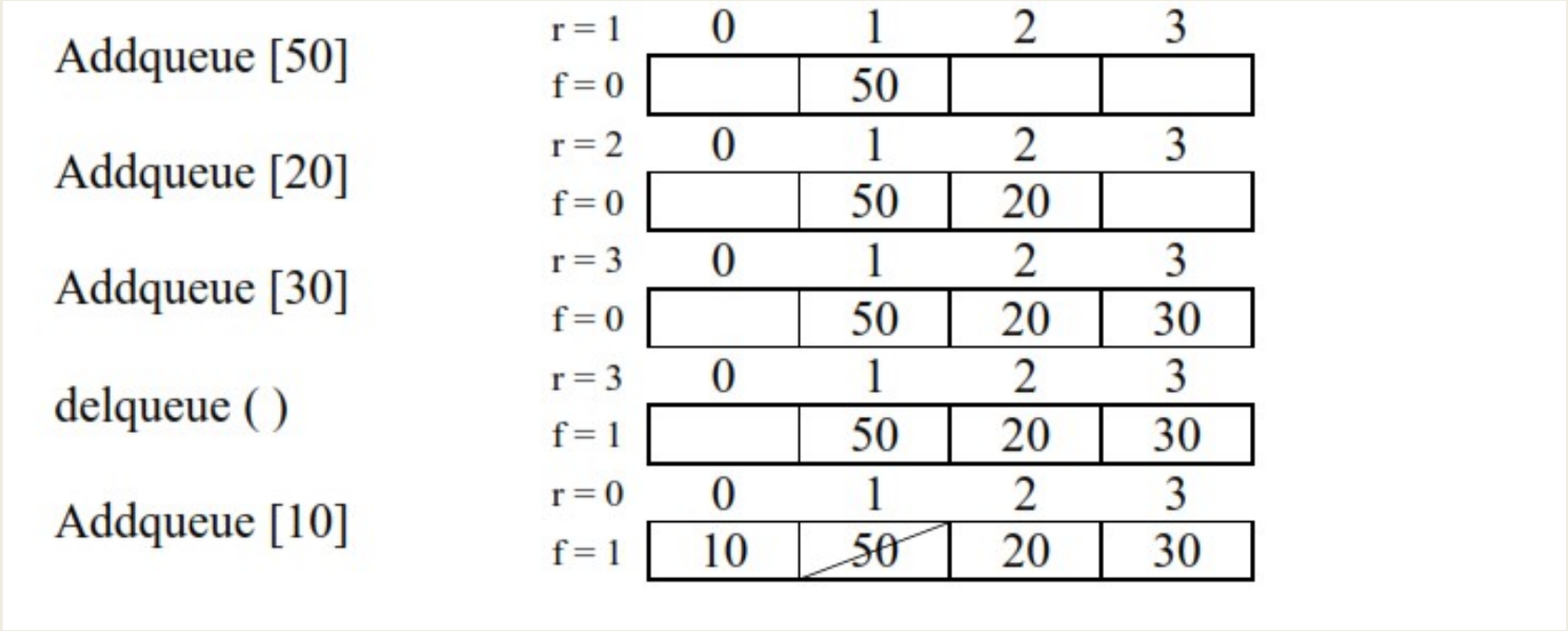


- Circular queue:
  - *If front = rear,*
    - queue is empty
  - *If front = (rear+1)%n,*
    - queue is full.
  - *rear = (rear+1)%n*
  - *front = (front+1)%n*



QUEUES

- An example of circular queue:



QUEUES

- Double-ended queues:

```
from collections import deque

dq = deque(maxlen=5)   # Double-ended queue with max size 5
dq.append(10)          # Add to the right
dq.append(30)
dq.appendleft(20)      # Add to the left
print(dq)              # deque([20, 10, 30])

dq.pop()               # Remove from the right
print(dq)              # deque([20, 10])

dq.popleft()           # Remove from the left
print(dq)              # deque([10])

from queue import Queue

q = Queue(maxsize=5)   # FIFO queue with max size 5
q.put(10)              # Add to the queue
q.put(20)
print(q.get())         # Remove and return the first element (10)
print(q.get())         # Remove and return the next element (20)
```

```
deque([20, 10, 30], maxlen=5)
deque([20, 10], maxlen=5)
deque([10], maxlen=5)
10
20
```