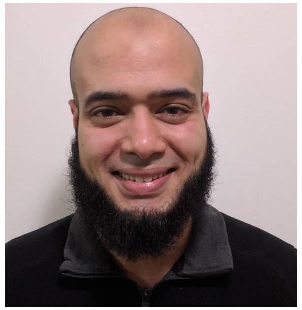
Data Structures
Circular Queue

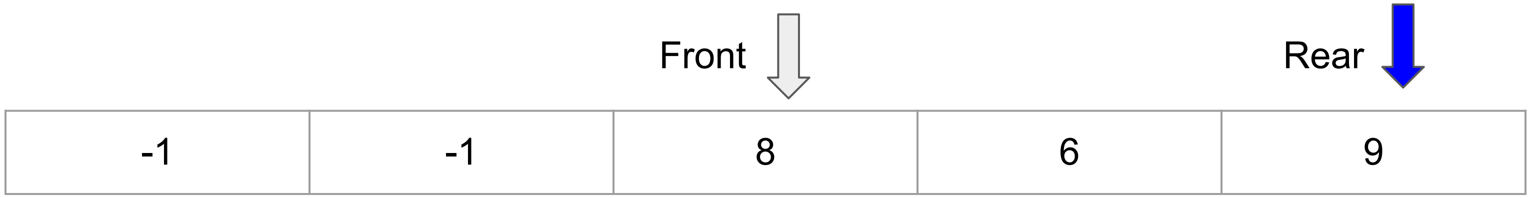
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



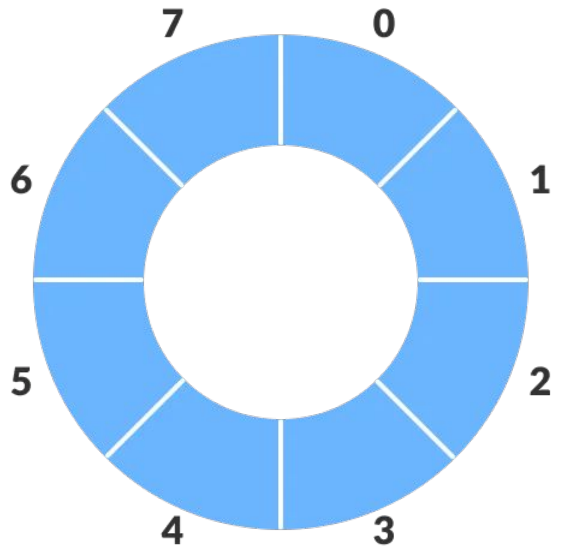
Array-based: Front-Rear approach

- We'll use two **indices**; front and rear; which represent the start and end of the array respectively
  - When we **enqueue** an element, we add it to the rear ⇒ **O(1)**
  - When dequeue an element, we **shift the front** index to the right ⇒ **O(1)**
- Enqueue 3: ERROR Queue is full!
- However, there are empty slots at the beginning!
  - This is a critical **drawback** of this approach



Circular Queue

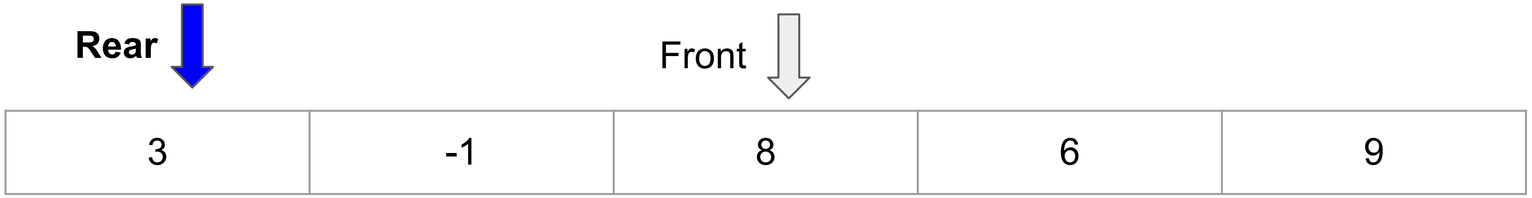
- There is a simple way to solve the previous space issue
- Simply, think of the array as a circle
    - On the right, an array of 8 elements is arranged as a circle
- This helps to visualize the circular queue. In the diagram, after the final element in the array, there is another element, which is at position 0
- Now, the queue is full IFF all elements are in use



Img src

Array-based Circular queue

- We will have 2 indices: front and rear representing start to end in array
  - When we enqueue element we add it in rear
  - When we dequeue element we shift font to the right ⇒ O(1)
- Enqueue 3
  - Now, move from the last index to index 0 - and add the new element
  - Observe how 'rear' is now **BEFORE** front

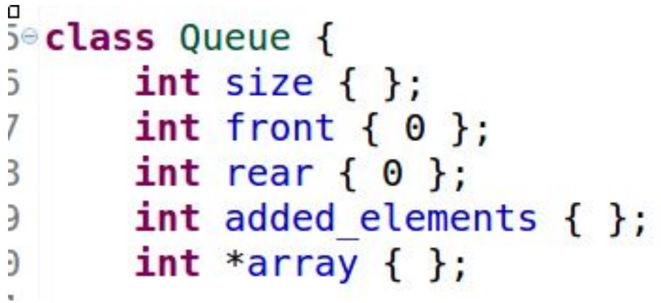


Initial values for rear & front

- There are several approaches for that
  - For all approaches, you must be **consistent** throughout the entire implementation
  - **Careful** conditions for IsEmpty and IsFull
- Possible initializations
  - rear = front = -1 [initially equal]
  - rear = front = 0 [initially equal]
  - rear = -1 and front = 0 [initially !equal]
  - rear = size - 1 and front = 0 [initially !equal]
- **int added_elements = 0;**
  - To both simplify coding and avoid tricky conditions, maintain a counter for the number of elements added

Circular Queue: Data Structure

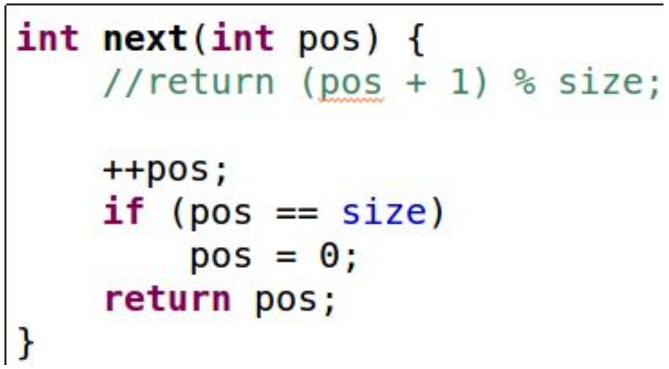
- We initially use **front = rear = 0**
    - To add a new element, **add** at **rear** and **move rear**
    - To dequeue an element, get the **front**, and then move **front**



```
class Queue {
        int size { };
        int front { 0 };
        int rear { 0 };
        int added_elements { };
        int *array { };
```

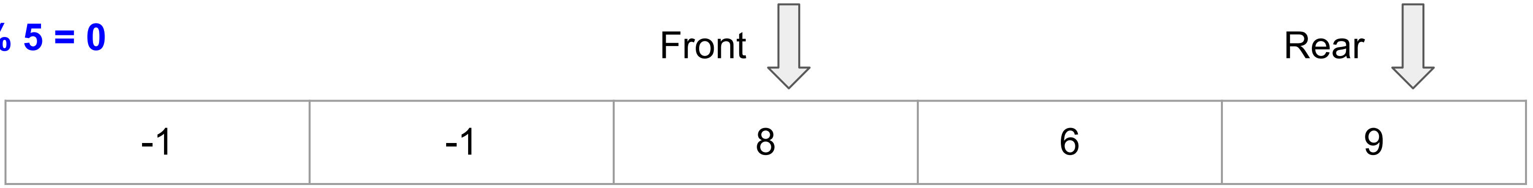
Circular Queue: Move index

- To move the index forward, consider:
  - If this is the **last element** in the array, make the next position = 0
  - We can do this with an if condition (efficient)
  - Or using the modulus operator (%)
- Assume size = 5. Let’s try positions from 0 to 5
  - 0 % 5 = 0
  - 1 % 5 = 1
  - 2 % 5 = 2
  - 3 % 5 = 3
  - 4 % 5 = 4
  - **5 % 5 = 0**



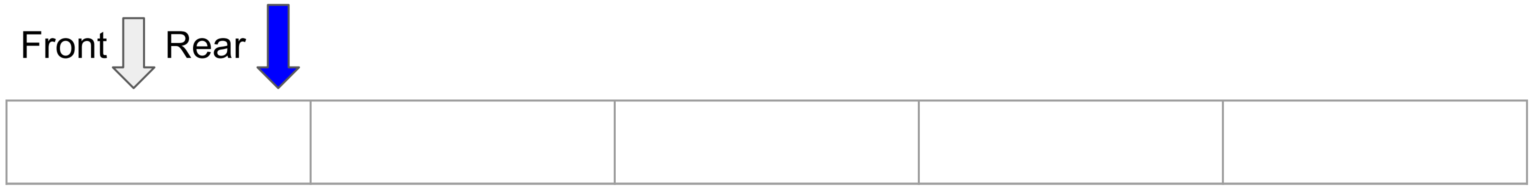
```
int next(int pos) {
    //return (pos + 1) % size;

    ++pos;
    if (pos == size)
        pos = 0;
    return pos;
}
```



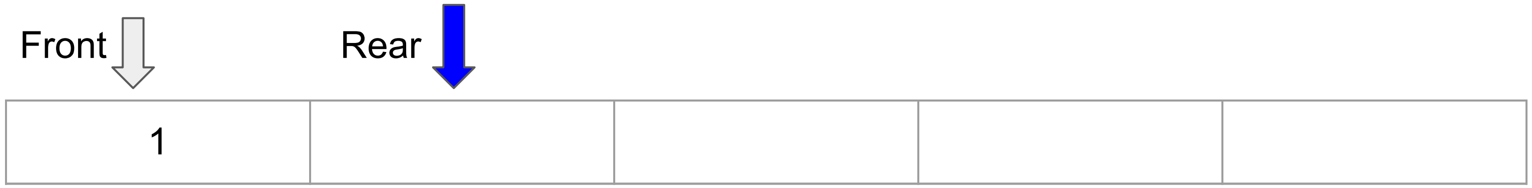
Let’s Simulate: Queue of size 5

- Initially an empty queue. Initially, rear = front = 0
- Observe: The EMPTY queue has rear==front



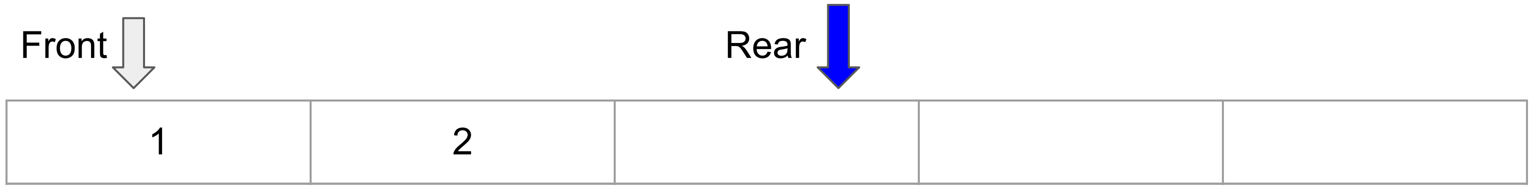
Let’s Simulate: Queue of size 5

- Enqueue (1) ⇒ Add to the rear position and move it



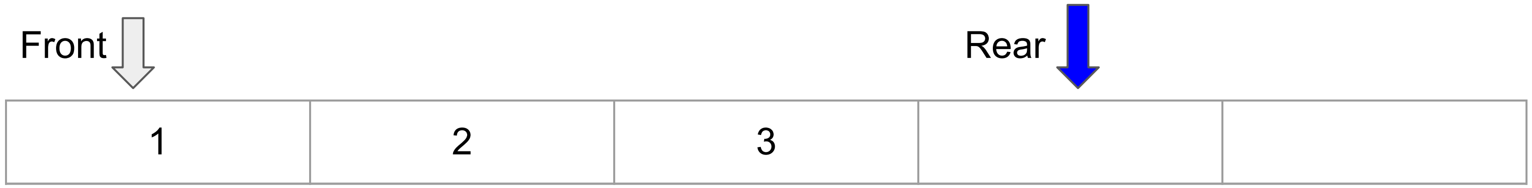
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2)



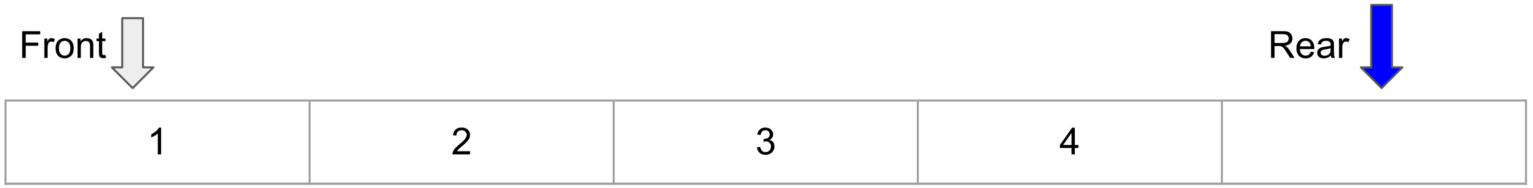
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3)



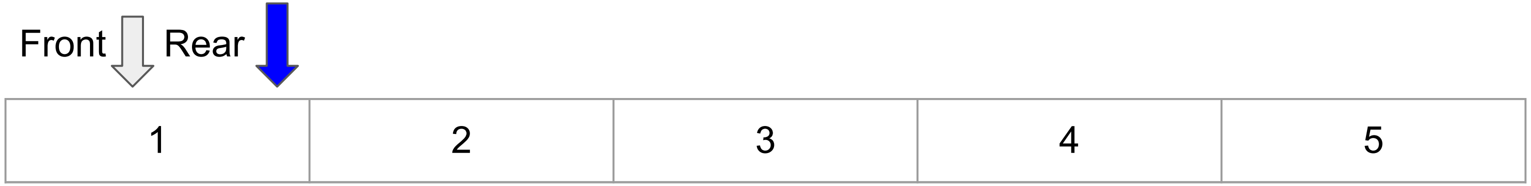
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4)
- Observe: rear is now in the last array position
    - One more enqueue and it moves the index to 0



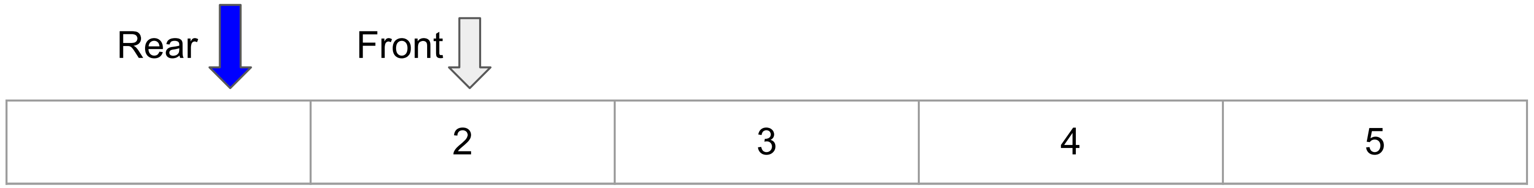
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Observe: the queue is full, but rear==front
  - How can we know whether the array is empty or full?
  - We **can’t!**
  - Use the added_elements variable
    - 0 = empty
    - 5 = full



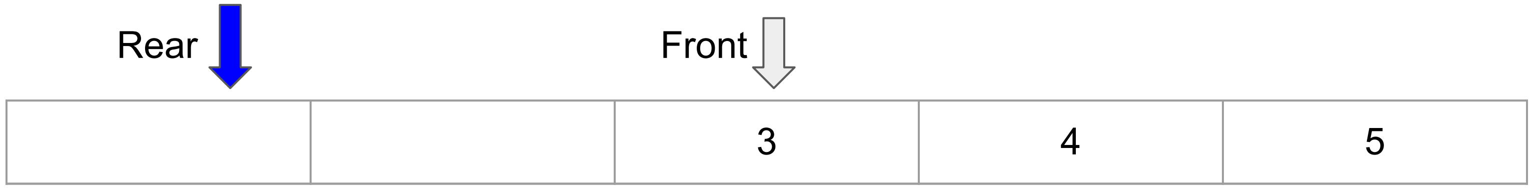
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Observe: Front after Rear



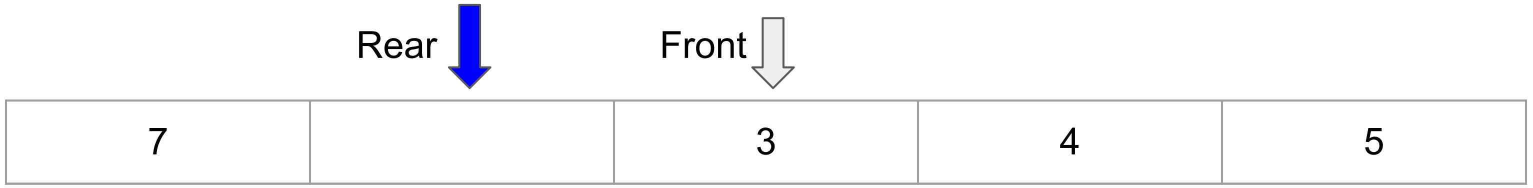
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2



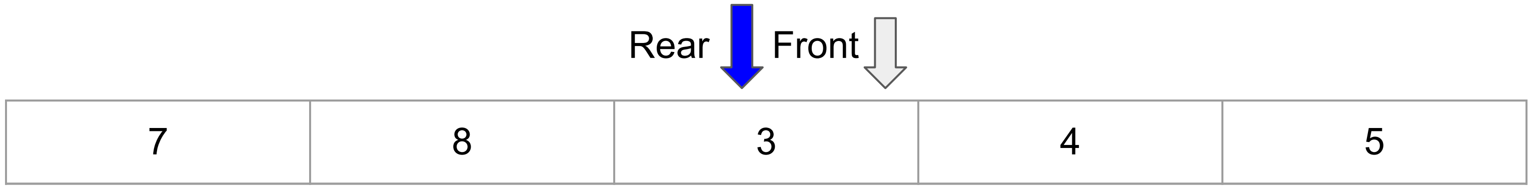
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7)



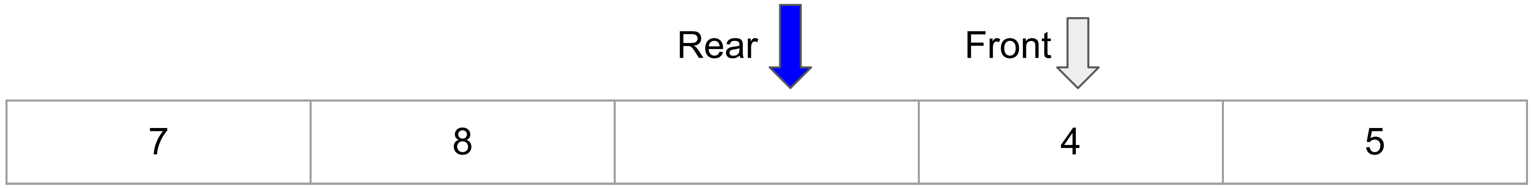
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Again full but rear = front = index 2



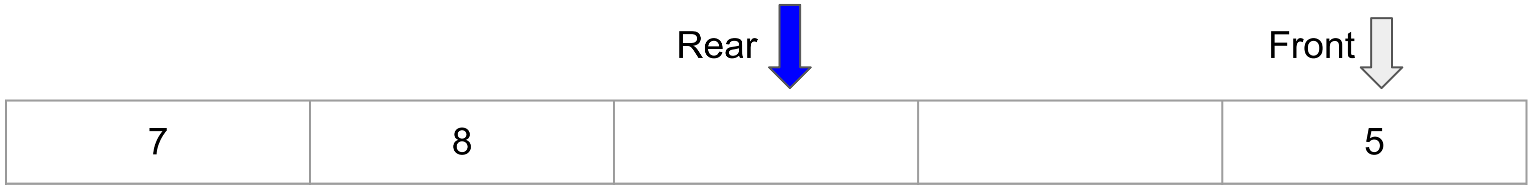
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3



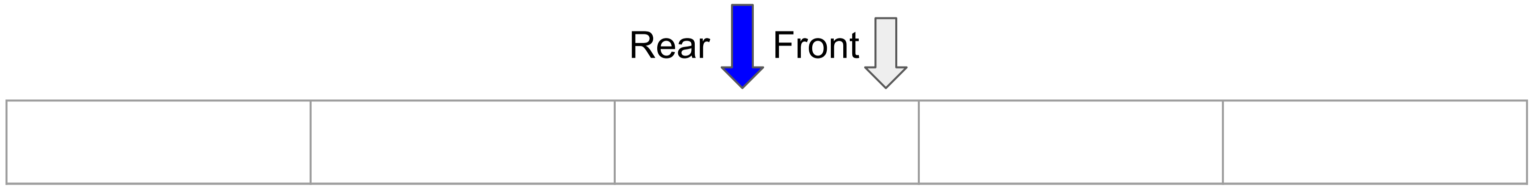
Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3
- Dequeue ⇒ 4



Let’s Simulate: Queue of size 5

- Enqueue (1), Enqueue (2), Enqueue (3), Enqueue (4), Enqueue (5)
- Dequeue ⇒ 1
- Dequeue ⇒ 2
- Enqueue (7), Enqueue (8)
- Dequeue ⇒ 3
- Dequeue ⇒ 4
- Dequeue, Dequeue, Dequeue ⇒ 5, 7, 8
    - Observe: empty with front = rear = 2



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."