Data Structures & Algorithms (COMP2113)
Lecture # 21
Basic Data Structures | Part 05
Queue

Course Instructor

Dr. Aftab Akram

PhD CS

Assistant Professor

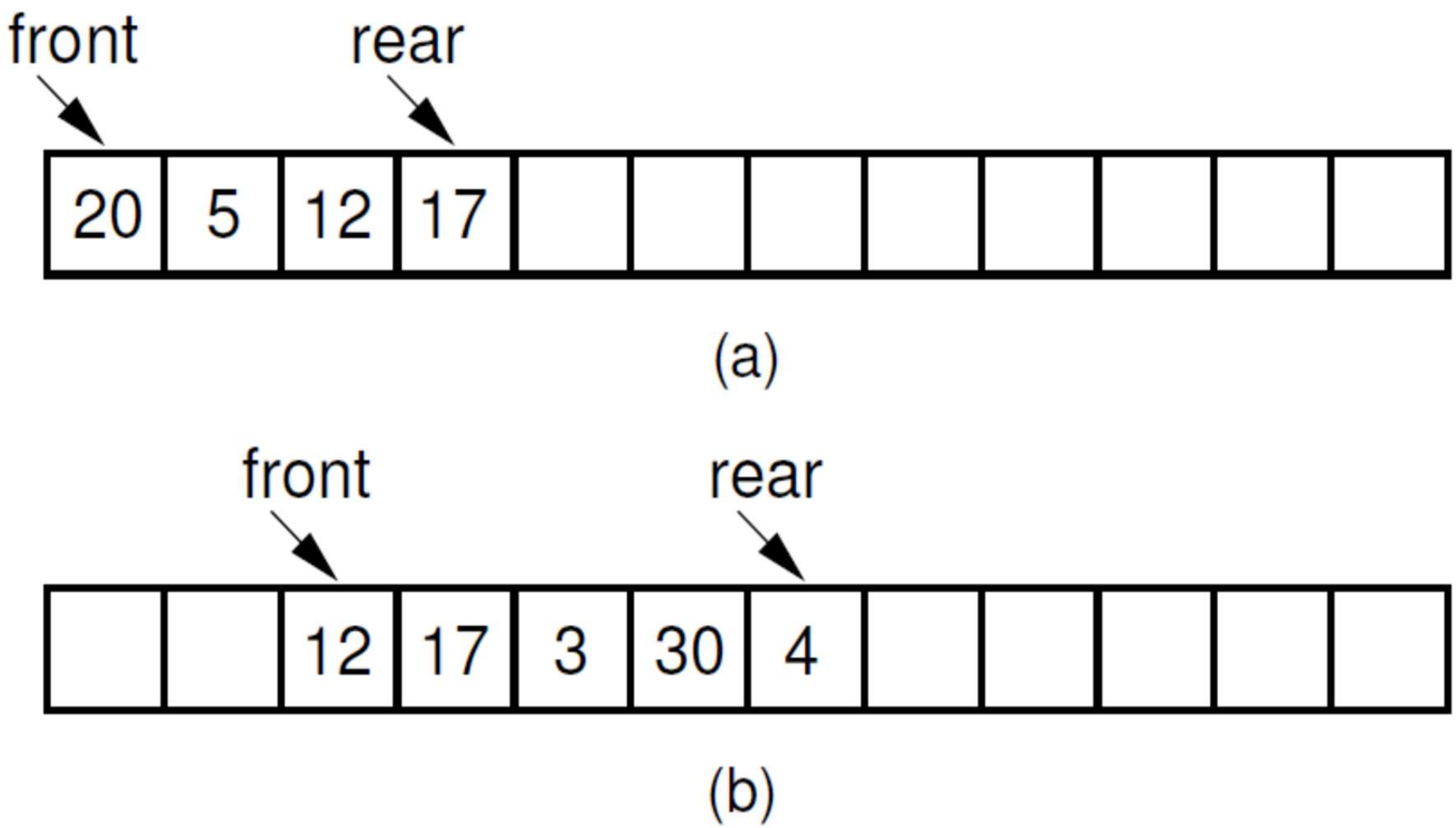
Department of Information Sciences, Division of Science & Technology

University of Education, Lahore

Queue

- The queue is a list-like structure that provides restricted access to its elements.
- Queue elements may only be inserted at the back (called an **`enqueue`** operation)
- And removed from the front (called a **`dequeue`** operation).
- Queues operate like standing in line at a movie theater ticket counter. If nobody cheats, then newcomers go to the back of the line. The person at the front of the line is the next to be served.
- Queues release their elements in order of arrival.
- A queue is called a **"FIFO"** list, which stands for **"First-In, First-Out."**

Queue



Queue-ADT

- **`clear-`**Reinitialize the queue.
- **`enqueue-`**Place an element at the rear of the queue.
- **`dequeue-`** Remove and return element at the front of the queue.
- **`frontValue-`** returns a copy of the front element.
- **`length-`** Returns the number of elements in the queue.

Array Based Queue

- The array-based queue is somewhat tricky to implement effectively.
- A simple conversion of the array-based list implementation is not efficient.
- Assume that there are n elements in the queue.
- By analogy to the array-based list implementation, we could require that all elements of the queue be stored in the first n positions of the array.
- If we choose the rear element of the queue to be in position 0, then **`dequeue`** operations require only $\Theta(1)$ time because the front element of the queue (the one being removed) is the last element in the array.
- However, **`enqueue`** operations will require $\Theta(n)$ time, because the n elements currently in the queue must each be shifted one position in the array.

Array Based Queue

- If instead we chose the rear element of the queue to be in position$n-1$, then an **`enqueue`** operation is equivalent to an **`append`** operation on a list.
- This requires only $\Theta(1)$ time.
- But now, a **`dequeue`** operation requires $\Theta(n)$ time, because all of the elements must be shifted down by one position to retain the property that the remaining $n-1$ queue elements reside in the first $n-1$ positions of the array.

Array Based Queue

- A far more efficient implementation can be obtained by relaxing the requirement that all elements of the queue must be in the first n positions of the array.
- We will still require that the queue be stored be in contiguous array positions, but the contents of the queue will be permitted to drift within the array.
- Now, both the **`enqueue`** and the **`dequeue`** operations can be performed in $\Theta(1)$ time because no other elements in the queue need be moved.

Array Based Queue

- This implementation raises a new problem.
- Assume that the front element of the queue is initially at position 0, and that elements are added to successively higher-numbered positions in the array.
- When elements are removed from the queue, the front index increases.
- Over time, the entire queue will drift toward the higher-numbered positions in the array.
- Once an element is inserted into the highest-numbered position in the array, the queue has run out of space.
- This happens despite the fact that there might be free positions at the low end of the array where elements have previously been removed from the queue.

Next Lecture

- In next lecture, we will discuss circular and linked queues.