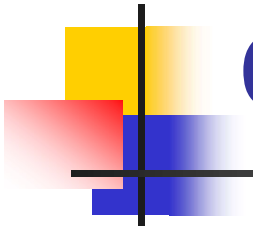
B. Tech. I CSE (Sem-2)
Data Structures
CS102

# Queue: A Data Structure

*Some slides are kept with logical or syntax error. So, if you are absent in theory class, please also refer the slides provided at the end of the presentation.

Dr. Rupa G. Mehta
Dr. Dipti P. Rana
Department of Computer Science and Engineering
SVNIT, Surat



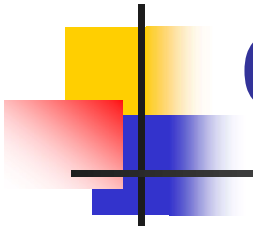
# Queue Data Structure

- Basic principles
- Operation of queue
- Queue using Array
- Applications of queue
- Queue using Linked List

# Queue Data Structure

- Data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).
- An ordered group of homogeneous items of elements.

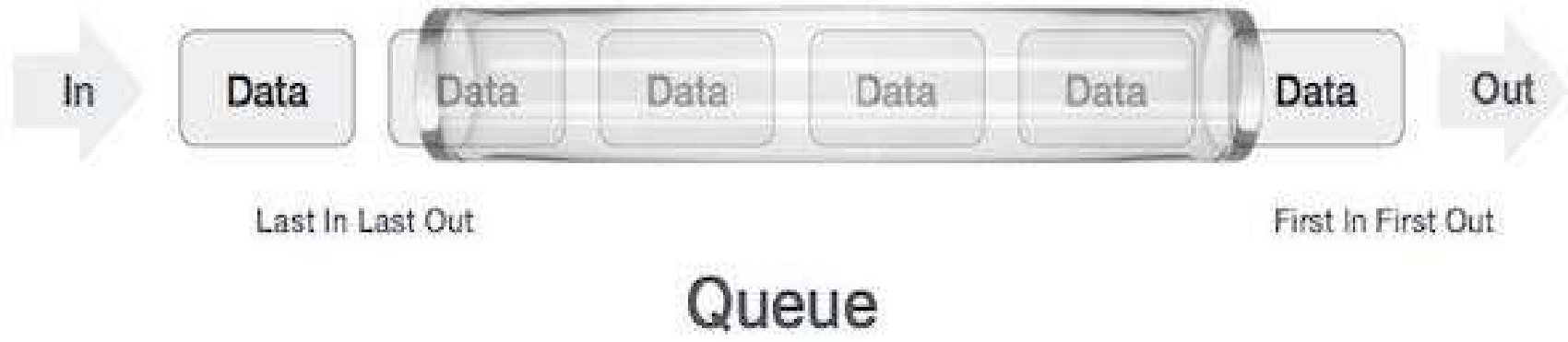

- Elements are added at one end.
- Elements are removed from the other end.
- The element added first is also removed first
- FIFO: First In, First Out



# Queue Data Structure

- Traffic Management Systems
  - In traffic control and toll booths, vehicles often form a queue as they wait for their turn
  - An example of queue real-time applications where vehicles are processed in the order they arrive
- Keyboards' keypress sequence
- Queue of network data packets to send
- Used as buffers on MP3 players and portable CD players, iPod playlist

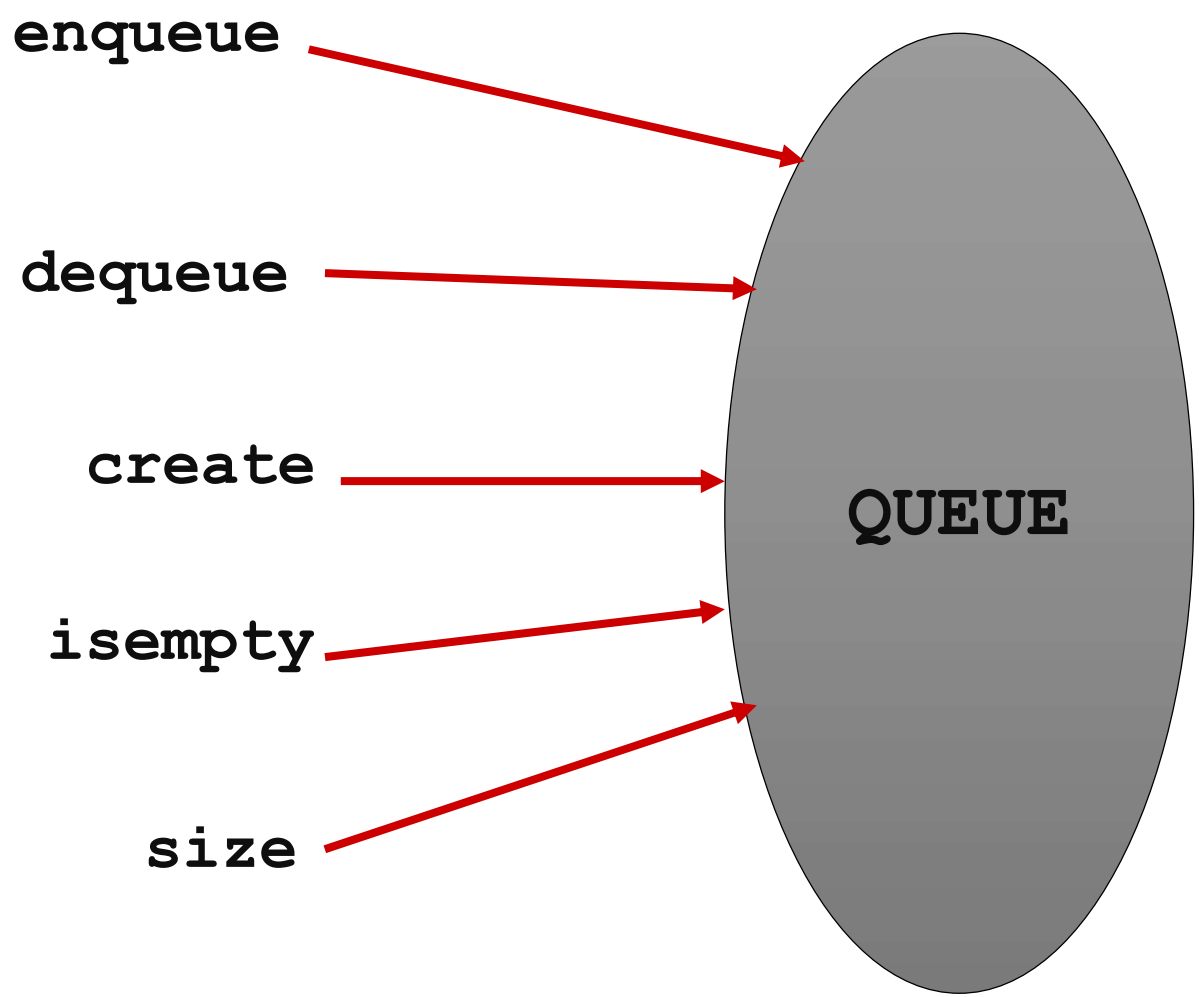
# Queue Data Structure

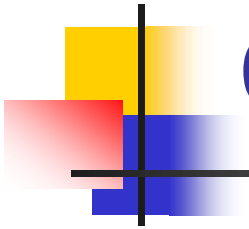



- As in stacks, a queue can also be implemented using Arrays, Linked-lists, Pointers and Structures.

# Queue Data Structure

Operations





# Queue Data Structure

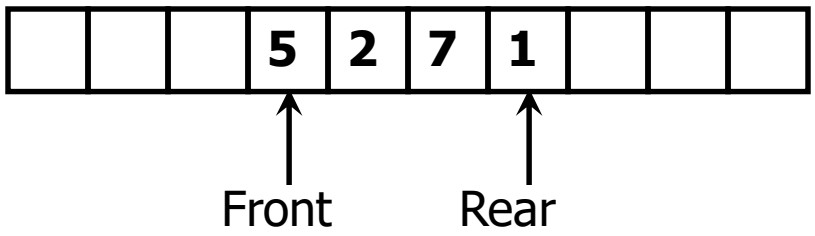
- Like stacks, *queues* are lists.
- With a queue, however, insertion is done at one end, whereas deletion is performed at the other end.
- The basic operations on a queue are *Enqueue*, which inserts an element at the end of the list (called the rear), and *Dequeue*, which deletes (and returns) the element at the start of the list (known as the front).

# Array Implementation of Queues

- For each queue data structure, we keep an array, *Queue*[], and the positions *Front* and *Rear*, which represent the ends of the queue.

- We also keep track of the number of elements that are actually in the queue, *Size*. All this information is part of one structure.

- The following figure shows a queue in some intermediate state. The cells that are blanks have undefined values in them:



# Algorithm Enqueue

```
Algorithm Enqueue (Q, F, R, Y)
    //Queue Q consisting of N elements
If( R>=N) /* check for the overflow condition */
        Printf ("Overflow"); return;
Else
{
        R = R + 1; /*Increment rear pointer */
        Q[R] = Y; /* insert element */
        If (F == 0) /* set the front pointer */
                F = 1;
        Return;
}
```

# Algorithm Dequeue

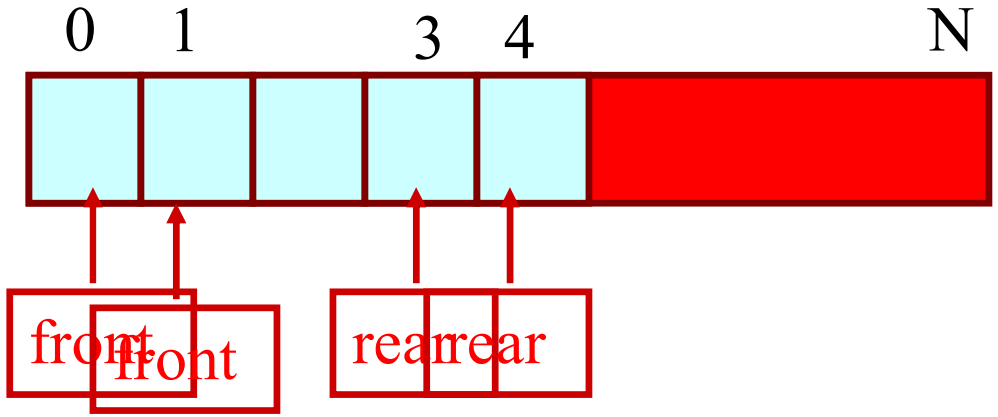
```
Algorithm DeQueue (Q, F, R)
If (F==0) /* check for the underflow condition */
        Printf (“underflow”);
        return 0; /* 0 denotes the empty queue */
Else
{
Y = Q [F]; /* delete element */
If (F == R)
F = R = 0;
Else
F = F + 1; /* increment front pointer */
Return Y;
}
```

# Array Implementation of Queues

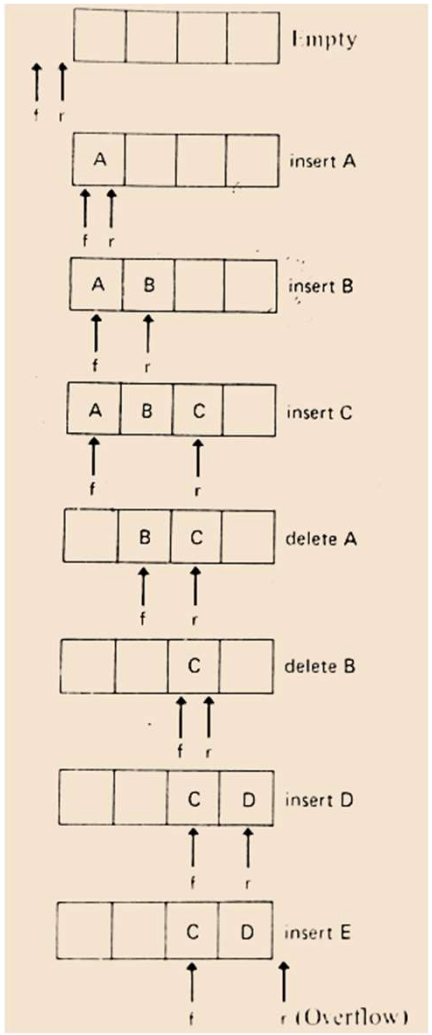
- The size of the queue depends on the number and order of enqueue and dequeue.
- It may be situation where memory is available but enqueue is not possible.

ENQUEUE DEQUEUE

Overlfow occurs on trying to insert , even though the first few locations are not being used
Effective queuing storage area of array gets reduced.



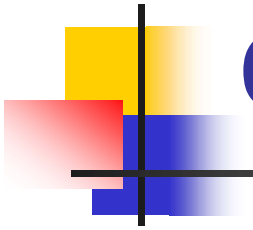
# Array Implementation of Queues




- How many elements are available?
- How many MORE elements CAN BE INSERTED?
- Show the different situation(s)
  - When only 1 element is available?
  - When 2 elements are available?
  - Etc.

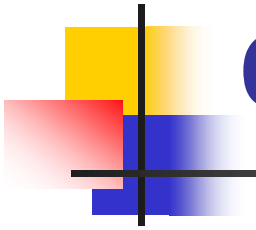
An introduction to data structures with applications
By Jean-Paul Tremblay, P. G. Sorenson · 1976

# Array Implementation of Queues



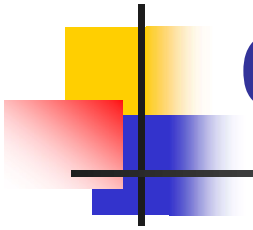
- Very wasteful of storage
- Arbitrary large amount of memory would be required to accommodate the elements
- ONLY BE USED
  - When the queue is emptied at certain intervals

# Array Implementation of Queues



- Performance
  - Let $n$ be the number of elements in the queue
  - The space used is $O(n)$
  - Each operation runs in time $O(1)$
- Limitations
  - The maximum size of the queue must be defined *a priori*, and cannot be changed
  - Trying to enqueue a new element into a full queue causes an implementation-specific exception

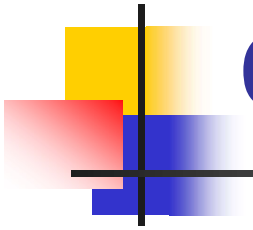
# Array Implementation of Queues



## Forwarding Elements Array-based Queue

- On element removal from array front location, remaining elements are shifted forward one position in the array
- Similar to what we did for an array-based stack
- The enqueue operation has running time
  - $O(n+k)$ with the k shifting

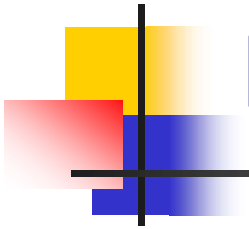
# Array Implementation of Queues



## Growable Array-based Queue

- In an enqueue operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one
- Similar to what we did for an array-based stack
- The enqueue operation has running time
  - $O(n)$ with the incremental strategy
  - $O(1)$ with the doubling strategy

# Applications of Queue

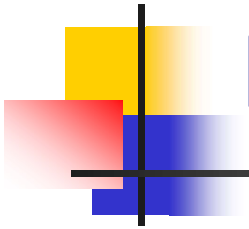


# Reading a String of Characters

- A queue can retain characters in the order in which they are typed

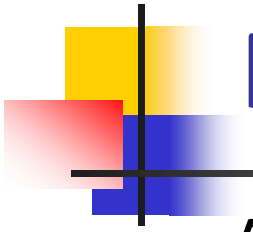
```
Queue.createQueue()
while (not end of line) {
    Read a new character ch
    Queue.enqueue(ch)
} //end while
```

- Once the characters are in a queue, the system can process them as necessary



# Recognizing Palindromes

- A palindrome
  - A string of characters that reads the same from left to right as its does from right to left
- To recognize a palindrome, a queue can be used in conjunction with a stack
  - A stack reverses the order of occurrences
  - A queue preserves the order of occurrences

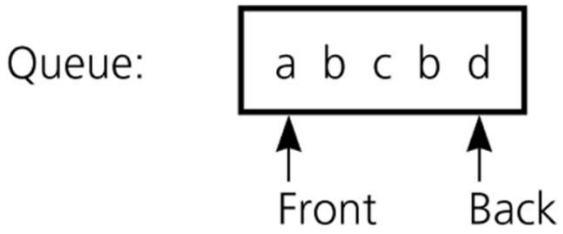


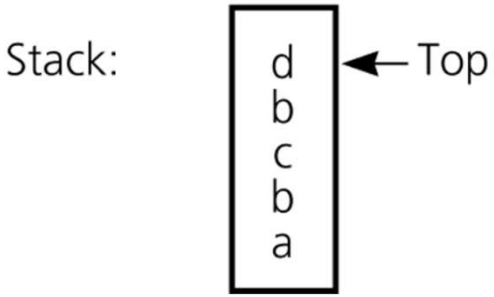
# Recognizing Palindromes

- A nonrecursive recognition algorithm for palindromes
  - As traverse the character string from left to right, insert each character into both a queue and a stack
  - Compare the characters at the front of the queue and the top of the stack

# Recognizing Palindromes

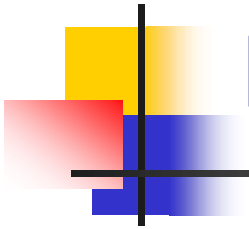
String: abcbd



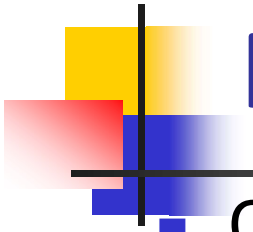

Inserting a string into both

a queue and a stack



# Radix Sort

- Radix is a synonym for base. base 10, base 2
- Multi pass sorting algorithm that only looks at individual digits during each pass
- Use queues as buckets to store elements
- Create an array of 10 queues
- Starting with the least significant digit place value in queue that matches digit
- Empty queues back into array
- Repeat, moving to next least significant digit



# Radix Sort

- Original values in array

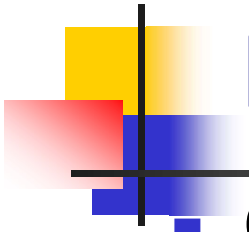
  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Look at ones place

  **9**, 11**3**, 7**0**, 8**6**, 1**2**, 9**3**, 3**7**, 4**0**, 25**2**, **7**, 7**9**, 1**2**

- Array of Queues (all empty initially):

| | |
|---|---|
| 0 | 5 |
| 1 | 6 |
| 2 | 7 |
| 3 | 8 |
| 4 | 9 |



# Radix Sort

- Original values in array

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Look at ones place

  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Queues:

| | | | |
|---|---|---|---|
| 0 | 70, 40 | 5 | |
| 1 | | 6 | 86 |
| 2 | 12, 252, 12 | 7 | 37, 7 |
| 3 | 113, 93 | 8 | |
| 4 | | 9 | 9, 79 |

# Radix Sort

- **Empty queues in order from 0 to 9 back into array**

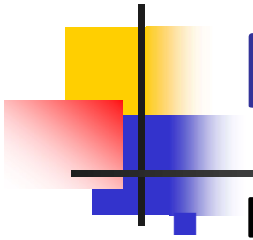
  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- **Now look at 10's place**

  70, 40, 12, 252, 12, 113, 93, 86, 37, _7, _9, 79

- Queues:

| | |
|---|---|
| 0 _7, _9 | 5 252 |
| 1 12, 12, 113 | 6 |
| 2 | 7 70, 79 |
| 3 37 | 8 86 |
| 4 40 | 9 93 |



# Radix Sort

- Empty queues in order from 0 to 9 back into array

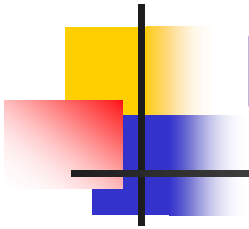
  7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93

- Now look at 100's place

  __7, __9, _12, _12, 113, _37, _40, 252, _70, _79, _86, _93

- Queues:

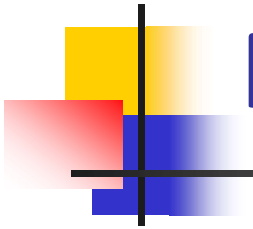
| | | |
|---|---|---|
| 0 | _7, _9, _12, _12, _37, _40, _70, _79, _86, _93 | 5 |
| 1 | 113 | 6 |
| 2 | 252 | 7 |
| 3 | | 8 |
| 4 | | 9 |



# Radix Sort

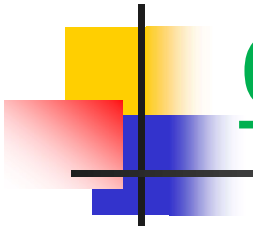
- Empty queues in order from 0 to 9 back into array

  7, 9, 12, 12, 40, 70, 79, 86, 93, 113, 252



# Point of Sale (POS)

- Maintains a list of customers waiting for service
  - Can add a new customer at the end
  - Can display name of next customer to serve
  - Can display the length of the line
  - Can determine how many people are ahead of a particular waiting customer



# Queue Variants

- Circular Queue
- Doubly Ended Queue
- Priority Queue