ZAMAN UNIVERSITY

Data Structures and Algorithms

Chapter 2

Abstract Data Types

Outline

- Stacks
- Queues and Priority Queues
- Linked Lists
- Abstract Data Types
- Specialized Lists

Outline

- Stacks
- Queues and Priority Queues
- Linked Lists
- Abstract Data Types
- Specialized Lists

Queues

- The word *queue* is British for *line* (the kind you wait in)
- In computer science a queue is a data structure that is similar to a stack, but in a queue the First In, the First Out (FIFO)

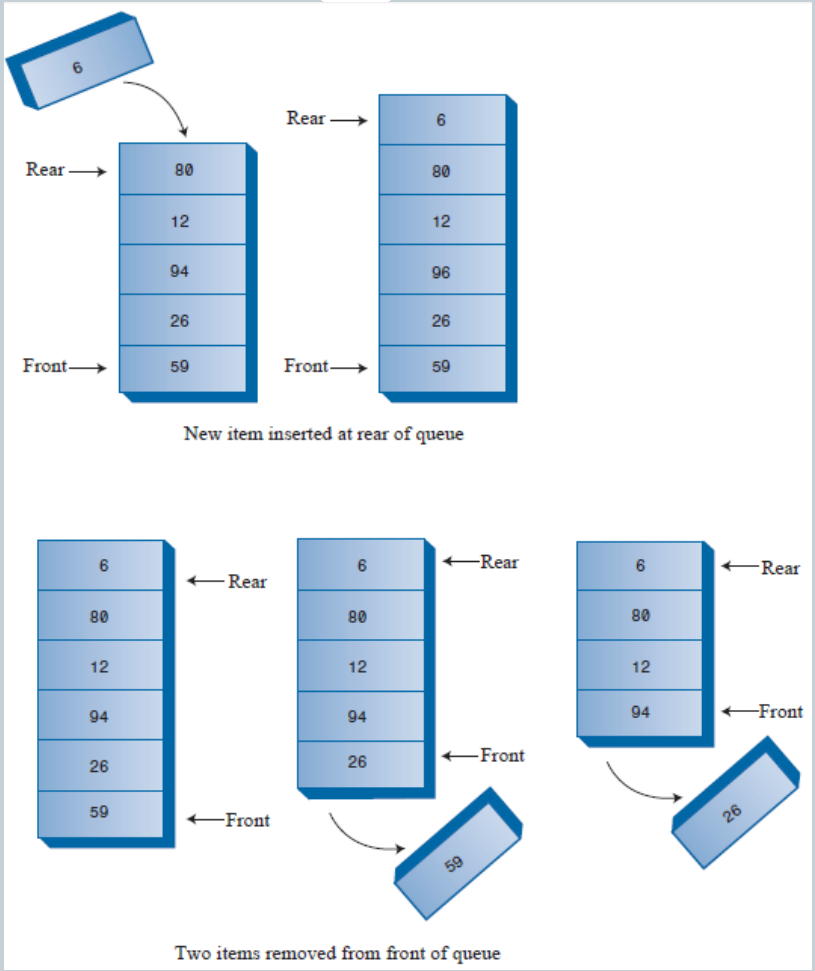
Queues: How Does It Work?

- A queue works like the line at the movies
- The first person to join the rear of the line is the first person to reach the front of the line and buy a ticket
- The last person to line up is the last person to buy a ticket

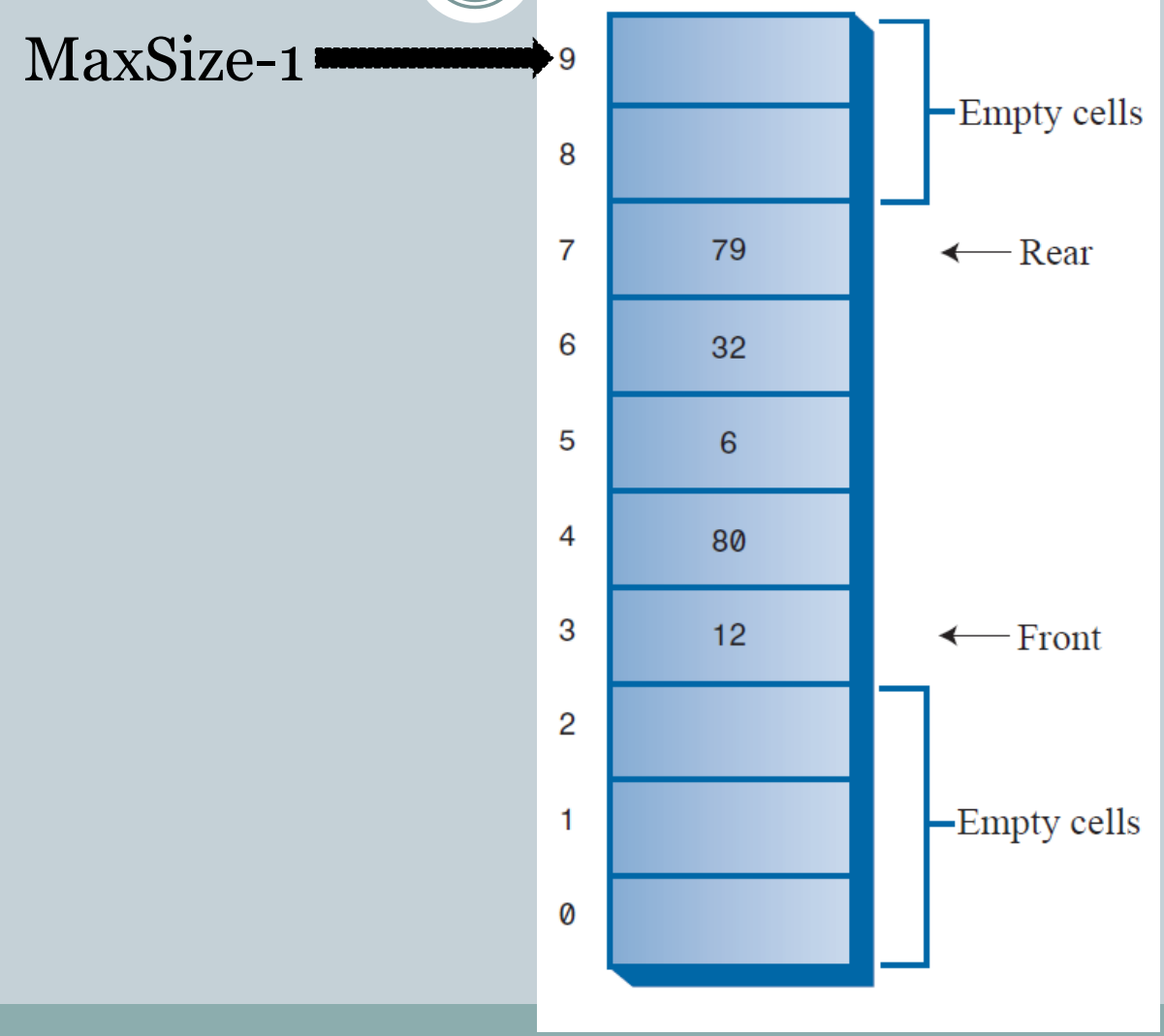
People join the queue at the rear

People leave the queue at the front

Queue: Insert and Remove Operations



Queue: Some Items Removed



Queue: The Empty and Full Error

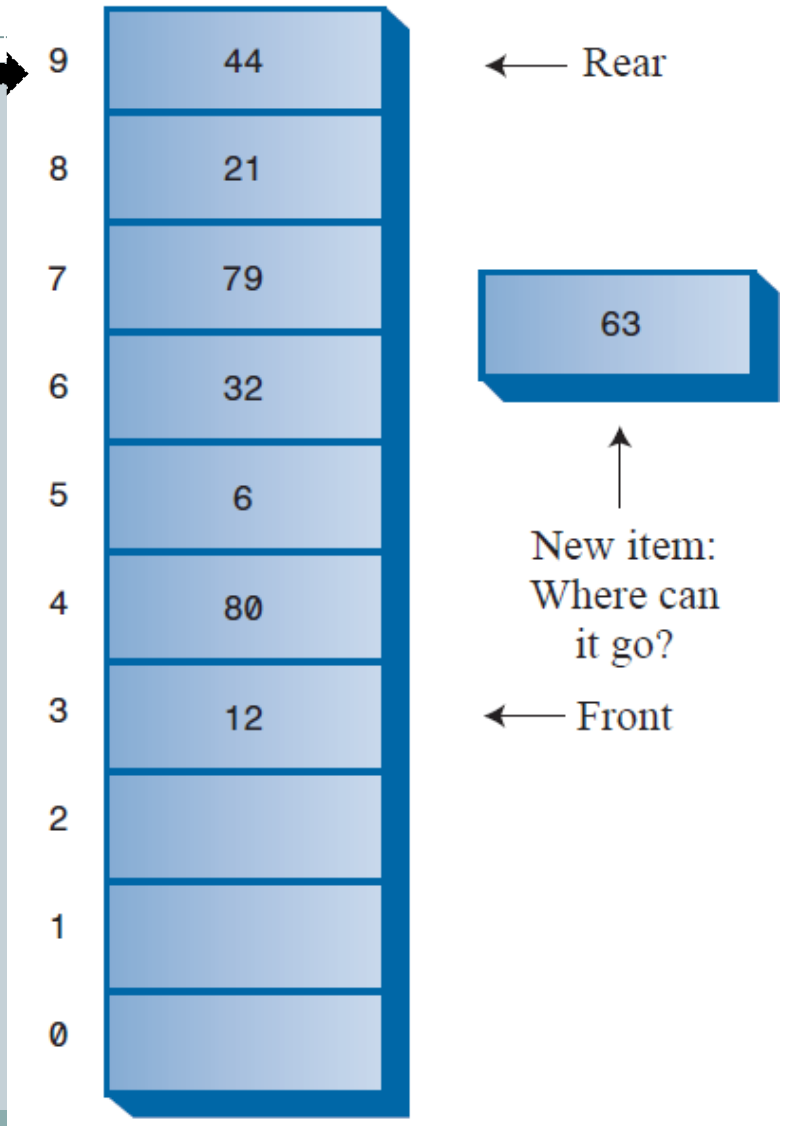
- In case, there are no more items in the queue, you cannot remove an item from queue
- In case, all the cells are already occupied, you cannot insert a new item to queue

Circular Queue[1]

- To Avoid the problem of not being able to insert more items into the queue even when it's not full

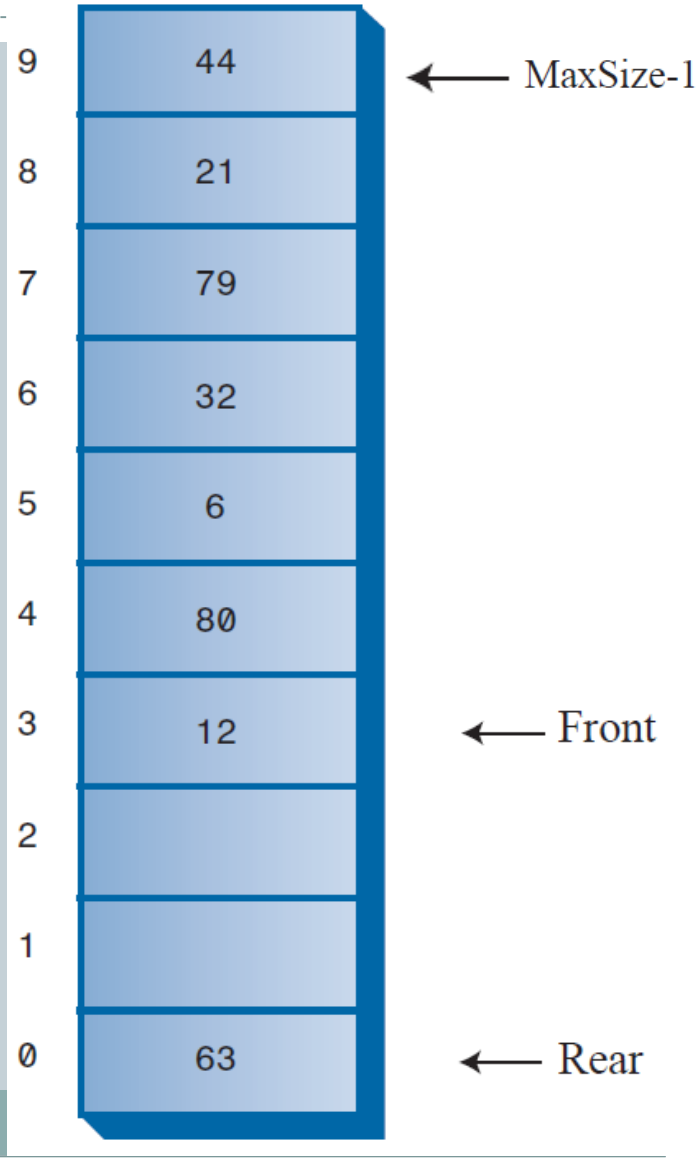
- the Front and Rear arrows *wrap around* to the beginning of the array.

- The result is a *circular queue* (sometimes called a *ring buffer*).

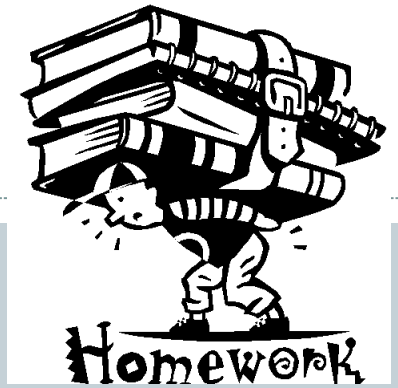


Circular Queue[2]

- **Insert a few more items**
  - The Rear arrow moves upward as you would expect.
  - After Rear has wrapped around, it's now below Front, the reverse of the original arrangement.
  - This ***broken sequence***: the items in the queue are in two different sequences in the array.
- Delete enough items
  - The Front arrow also wraps around
  - Back to the original arrangement, with Front below Rear
  - The items are in a single ***contiguous sequence.***



```
#define DEF_SIZE 10                                          //Define Maximum Size of Array (stack)
class CMyQueue{                                              //class CMyQueue
  private:
      int QueueMaxSize;                                      //Size of Queue Array
      int Array[ DEF_SIZE ];                                 //Queue Array
      int Rear;                                              //Rear of Queue
      int Front;                                             //Front of Queue
      int ItemNumber;                                        //Number Items of Queue
public:
      void InitMyQueue();                    //Function initiation StackMaxSize and Top
      bool IsFull();                                         //Function to check, is the Queue full? true - full
      void Insert( int NewItem );                            //Function to Insert a new item to the Rear of Queue
      bool IsEmpty();                                        //Function to check, is the Queue empty? true - empty
      void Remove();                                         //Function to remove item from the front of Queue
      int Size();                                            //Function to return the number of elements in the Queue
      int PeekFront();                              //Function to read item value from the front of the Queue
};
void CMyQueue::InitMyStack(){
    StackMaxSize = DEF_SIZE;
    Rear = -1;                             //Initiation value of Rear is -1
    Front = 0;
    ItemNumber = 0;
};
bool CMyQueue::IsFull(){
    if( ItemNumber == StackMaxSize-1 ) return true;        //return true if the Queue if true
    else return false;                                     //otherwise, return false
};
void CMyQueue::Insert( int NewItem ){
    if( !IsFull() ){                                       //check, if the Queue is NOT full
        if( Rear == QueueMaxSize-1 )                       //Deal with wrap around
              Rear = -1;
        Rear++;                                             //Move Top up (increase)
        Array[ Rear ] = NewItem;                            //Add new item to the top of Queues
        ItemNumber++;                                       //Increase Number Item in Queue
        cout << "Insert New Item Successfully";
    }
    else cout << "The Queue is full";                       //if the stack is full
};
```



Create a Queue with the full operations: Insert, Remove, PeekFront, size, isEmpty, isFull

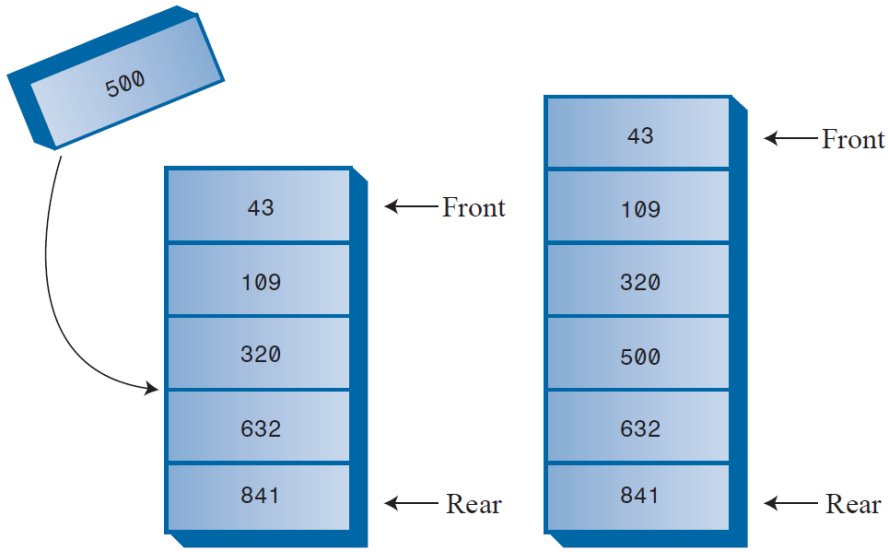
Priority Queues

- A priority queue is a more specialized data structure than a stack or a queue
- It is useful tool in a surprising number of situations
- Like an ordinary queue, a priority queue has a front and a rear, and items are inserted in the rear and removed from the front
- In a priority queue, items are ordered by key value, so that the item with the lowest key (or highest key) is always at the front
- Items are inserted in the proper position to maintain the order

Priority Queues: Inserting a New Item

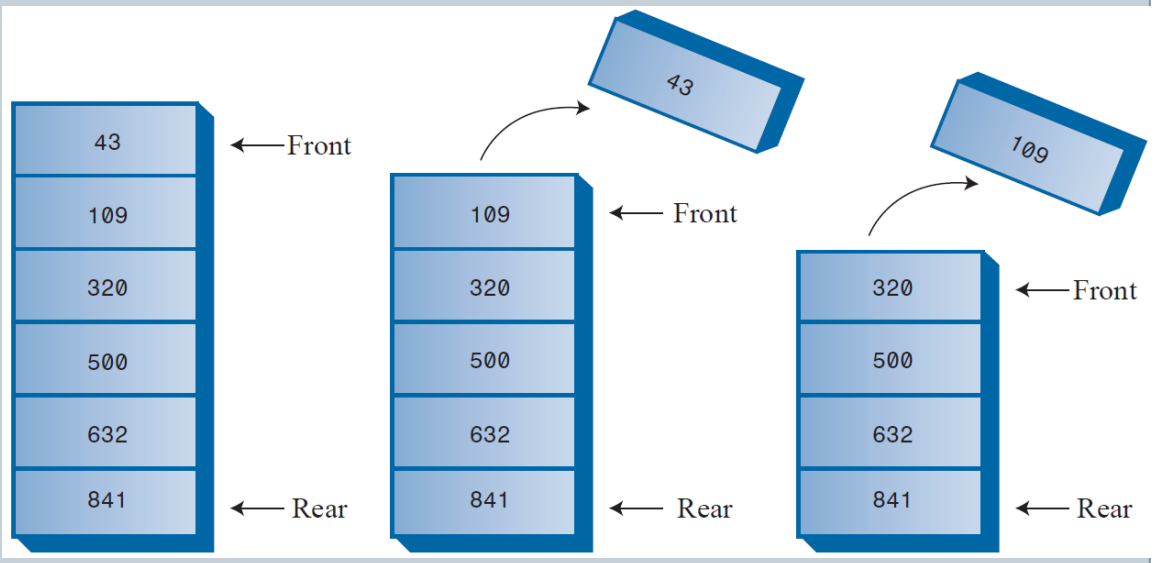
- Firstly, find the appropriate position by his value
- Insert a new item to the found position

- Notice that there is no wraparound in implementation of the priority queue
- Insertion is slow because the proper in-order position must be found, but deletion is fast



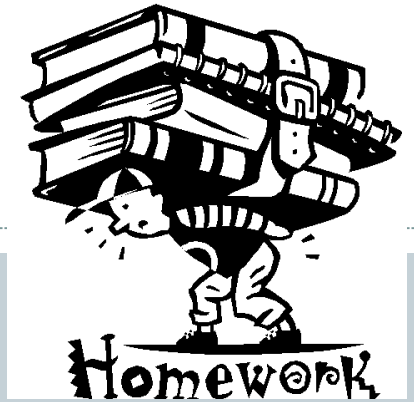
Priority Queues: Deleting an Item

- The item to be removed is always the front item (in both ascending and descending), thus removal is quick and easy
- The item is removed and the Front moves to the next item of the array
- No comparisons or shifting are necessary



Priority Queues: Priority Queues in C++

```
class CMyPriorQueue{                          //Array is sorted order, from max at 0 to min at ItemNumber-1
    int Array[ DEF_SIZE ];                    //PQ Array
    int QueueMaxSize;                         //Size of PQ
    int ItemNumber;                           //Item Number of PQ
  public:
      void InitPriorQueue();                  //Function for initiation QueueMaxSize and ItemNumber
      bool IsFull();                          //Function to check, is the PQ full? true - full
      void InsertPQ( int NewItem );           //Function to Insert NewItem to PQ
      void RemovePQ();                        //Remove the first item from PQ
      bool IsEmptyPQ();                       //Check, is the PQueue empty?
      int SizePQ();                           //Return ItemNumber
      int PeekFrontPQ();                      //Read, value of the front item (Min or Max)
};
void CMyPriorQueue::InitPriorQueue(){
    QueueMaxSize = DEF_SIZE;                  //Init value of QueueMaxSize and ItemNumber
    ItemNumber = 0;
};
void CMyPriorQueue::InsertPQ(int NewItem){
    if( ItemNumber == 0 ){                    //If the PQ is Empty
        Array[ ItemNumber ] = NewItem;        //Insert NewItem and increase ItemNumber
        ItemNumber++;
    }else{                                    //if the PQ is not Empty
        if( ItemNumber < QueueMaxSize ){      //if the PQ is not full
            int i;
            for( i=ItemNumber-1; i >= 0; --i  ){
                if( NewItem > Array[ i ] )    //Define the appropriate position by value of NewItem
                    Array[ i+1 ] = Array[ i ];
                else break;                   //Found the appropriate position for the NewItem
            }
            Array[ i+1 ] = NewItem;           //Insert NewItem
            ItemNumber++;                     //Increase ItemNumber
        }else cout << "The PQ is full!";      //In case, the PQ is full
    }
};
```



Create Priority Queue (ascending and descending) with full operations: IsFull(), InsertPQ(int NewItem), RemovePQ(), IsEmptyPQ(), SizePQ(), and PeekFrontPQ().

Read book of **Robert Lafore**, page: 146– 165 for next lecture

To be continued…