ASSIGNMENT INSTRUCTIONS

1. Assignment 03: **50 points w/ 0 E.C. points**
2. Due Date & Time: **07-31-2019 at 11:55 PM**

WHAT TO SUBMIT

1. Assignment Report
2. Code

HOW TO SUBMIT

- Please refer to the "Guidelines for All Assignments" and the "Assignment Report Template" which we discussed in detail in assignment 00.

PERFORMANCE TRACKER		
ASMT	GRADE	YOUR GRADE
00	15	
01	50	
02	50	
MID-REDO	25	
03	50	
TOTAL	190	

A: 90-100% **B**: 80-89% **C**: 70-79% **D**: 60-69% **F**: 0-60%
The course grader provides feedback to your assignments on iLearn.

This assignment contains several small problems.

- You will need the code provided here: http://csc220.ducta.net/Assignments/Assignment-03-Code.zip
- Please include screenshots of your output in your report and remember to submit your source code.

PART 1 – Introduction to Sorting, **26 points**

- Use this array of integer for the problems 1A, 1B, and 1C: **`9 5 7 8 3 2 4 7 6 1`**
- Each of these problems is worth **3 points**.

A. Show the contents of the array each time a **selection sort** changes it while sorting the array into **ascending order**.
B. Show the contents of the array each time an **insertion sort** changes it while sorting the array into **ascending order**.
C. Show the contents of the array each time a **Shell sort** changes it while sorting the array into **ascending order**.

D. --- **2 points** --- Suppose you want to find the largest entry in an unsorted array of n entries. Algorithm A searches the entire array sequentially and records the largest entry seen so far. Algorithm B sorts the array into descending order and then reports the first entry as the largest. Compare the time efficiency of the two approaches.

E. --- **15 points** --- Consider an **`n by n`** array of integer values. Write an algorithm to sort the rows of the array by their first value.

Implement your algorithm.

The code for this problem is provided in the **Assignment-03-Code.zip** archive. Your output must be identical to the output to the right.

```
The array is initially
    1  2  3  4  5
    3  4  5  1  2
    5  2  3  4  1
    2  3  1  4  5
    4  2  3  1  5

The array after sorting is
    1  2  3  4  5
    2  3  1  4  5
    3  4  5  1  2
    4  2  3  1  5
    5  2  3  4  1
```

PART 2 – Queues, Deques, and Priority Queues, **9 points**

A. --- **3 points** --- After the following statements execute, what are the contents of the queue?

```
QueueInterface<String> myQueue = new LinkedQueue<>();
myQueue.enqueue("Jane");
myQueue.enqueue("Jess");
myQueue.enqueue("Jon");
myQueue.enqueue(myQueue.dequeue());
myQueue.enqueue(myQueue.getFront());
myQueue.enqueue("Jim");
String name = myQueue.dequeue();
myQueue.enqueue(myQueue.getFront());
```

B. --- **3 points** --- After the following statements execute, what are the contents of the queue?

```
DequeInterface<String> myDeque = new LinkedDeque<>();
myDeque.addToFront("Jim");
myDeque.addToFront("Jess");
myDeque.addToBack("Jen");
myDeque.addToBack("Josh");
String name = myDeque.removeFront();
myDeque.addToBack(name);
myDeque.addToBack(myDeque.getFront());
myDeque.addToFront(myDeque.removeBack());
myDeque.addToFront(myDeque.getBack());
```

C. --- **3 points** --- After the following statements execute, what are the contents of the priority queue? Assume that the alphabetically earliest string has the highest priority.

```
PriorityQueueInterface<String> myPriorityQueue = new
LinkedPriorityQueue<>();
myPriorityQueue.add("Jim");
myPriorityQueue.add("Josh");
myPriorityQueue.add("Jon");
myPriorityQueue.add("Jane");
String name = myPriorityQueue.remove();
myPriorityQueue.add(name);
myPriorityQueue.add(myPriorityQueue.peek());
myPriorityQueue.add("Jose");
myPriorityQueue.remove();
```

D. --- **15 points** --- Use a circular doubly linked chain to **implement** the ADT deque.

In a **doubly linked chain**, the first and last nodes each contain one null reference, since the first node has no previous node and the last node has no node after it. In a circular doubly linked chain, the first node references the last node, and the last node references the first. Only one external reference is necessary—a reference to the first node—since you can quickly get to the last node from the first node.

The code for this problem is provided in the **Assignment-03-Code.zip** archive. Your output must be identical to the output to the right.

```
Create a deque:


isEmpty() returns true

Add to front and back of deque to get
Joe Jess Jim Jill Jane Jerry


isEmpty() returns false



Testing getFront, getBack, removeFront, removeBack:

Joe is at the front of the deque.
Jerry is at the back of the deque.
Joe is removed from the front of the deque.
Jerry is removed from the back of the deque.
Jess is at the front of the deque.
Jane is at the back of the deque.


Testing clear:


isEmpty() returns true


removeFront correctly finds deque empty
removeBack correctly finds deque empty

Done.
```