King Saud University
College of Computer and Information Sciences
Computer Science Department

CSC 212 **First Semester 1439-1440**

Tutorial #5

Problem 1

Write an efficient static method *split* that splits a queue of n elements into two queues. The elements with odd orders (i.e. 1st, 3rd, 5th …) should be put in the first queue and elements with even orders (i.e. 2nd, 4th, 6th …) should be put in the second queue. The original queue should remain unchanged at the end of the method.

Example: originalQ (18, 28, 53, 67, 32, 15, 73, 23, 1) → **oddQ** (18, 53, 32, 73, 1), **evenQ** (28, 67, 15, 23)

Method: *public static <T> void split(Queue<T> originalQ, Queue<T> oddQ, Queue<T> evenQ)*

```
public static <T> void split(Queue<T> originalQ, Queue<T> oddQ,
Queue<T> evenQ) {
      int count = originalQ.length();
      for (int i = 0; i < count; i++) {
            T element = originalQ.serve();
            originalQ.enqueue(element);
            if ((i+1) % 2 == 1) {
                  if (!oddQ.full())
                        oddQ.enqueue(element);
            }
            else {
                  if (!evenQ.full())
                        evenQ.enqueue(element);
            }
      }
}
```

Problem 2

Write a static method ***merge*** that merges two priority queues *priorityQ1* and *priorityQ2* in one priority queue *mergedQ*. Assume that *mergedQ* is initially empty.

Method: *public static <T> void merge(PriorityQueue<T> mergedQ, PriorityQueue<T> priorityQ1, Priority Queue<T> priorityQ2)*

```
public static <T> void merge(PriorityQueue<T> mergedQ,
PriorityQueue<T> priorityQ1, PriorityQueue<T> priorityQ2) {
      PQElement<T> element = null;
      int count = priorityQ1.length();
      for (int i = 0; i < count; i++) {
            element = priorityQ1.serve();
            if (!mergedQ.full())
                  mergedQ.enqueue(element.data, element.priority);
      }
      count = priorityQ2.length();
      for (int i = 0; i < count; i++) {
            element = priorityQ2.serve();
            if (!mergedQ.full())
                  mergedQ.enqueue(element.data, element.priority);
      }
}
```

Problem 3:

Write a static method ***remove*** that removes every element in the priority queue *priorityQ* having priority less than *priority*.

Method: *public static <T> void remove (PriorityQueue<T> priorityQ, int priority)*

```
public static <T> void remove(PriorityQueue<T> priorityQ, int
priority) {
      PQElement<T> element = null;
      PriorityQueue<T> tempPQ = new LinkedPQ<T>();
      int count = priorityQ.length();
      for (int i = 0; i < count; i++) {
            element = priorityQ.serve();
            if (element.priority >= priority)
                  tempPQ.enqueue(element.data, element.priority);
      }
      count = tempPQ.length();
      for (int i = 0; i < count; i++) {
            element = tempPQ.serve();
            priorityQ.enqueue(element.data, element.priority);
      }
}
```

Extra Problem

Show the array queue after each step noting the *size, head* and *tail*.

```
Queue<Integer> arrayQ = new ArrayQueue<Integer>(5);
arrayQ.enqueue(7);
arrayQ.enqueue(10);
arrayQ.enqueue(9);
Integer element = arrayQ.serve();
arrayQ.enqueue(8);
arrayQ.enqueue(5);
arrayQ.enqueue(2);
for (int i = 0; i < 3; i++)
     arrayQ.serve();
arrayQ.enqueue(element);
```