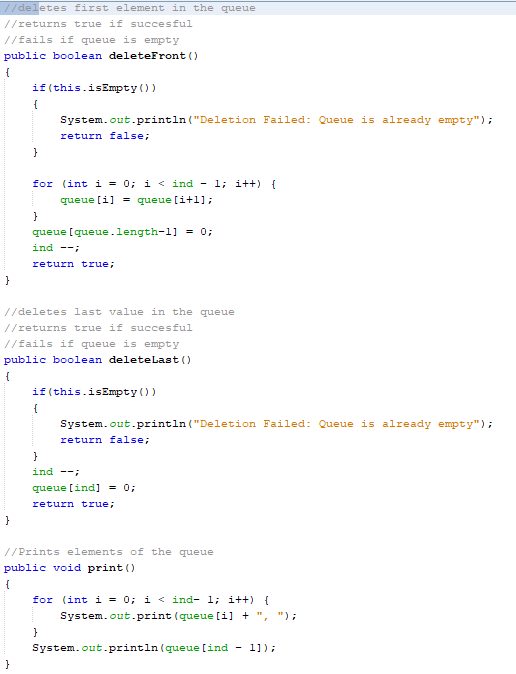
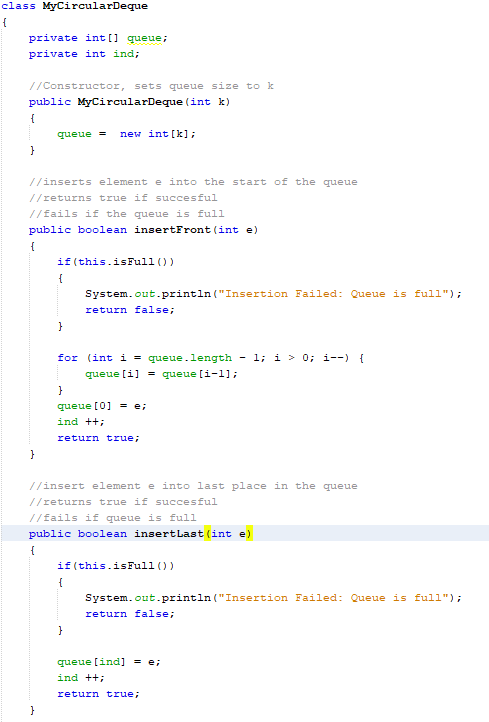
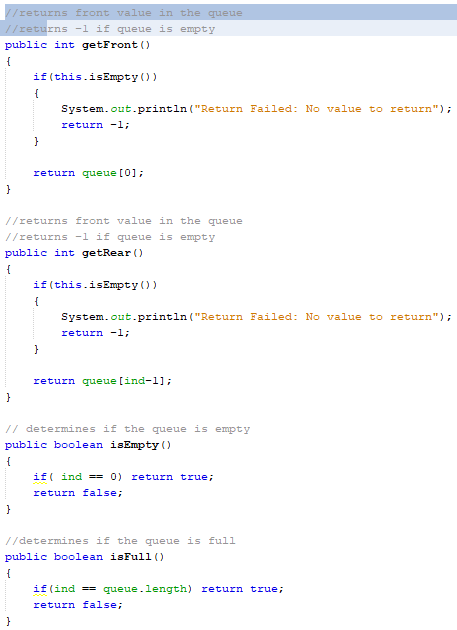
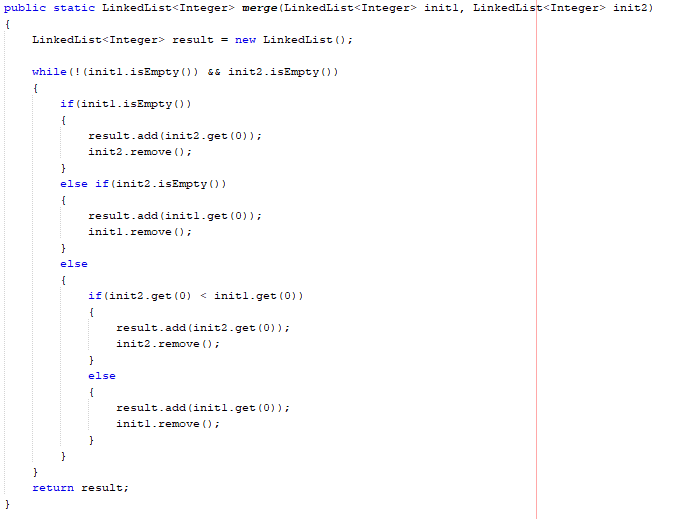
***well-documented summary that includes the annotation/justification of your algorithm and programming solution and the experimental results with the screenshots of your program testing you test. It should be at least 2 pages MS WORD or PDF file, (the more formal the better.) Note that screenshots should be readable to gain the full credits. The instructor will use the screenshots for student’s program demo.***

This document will rationalize the algorithms and functions used in my HW4 for CSC 310

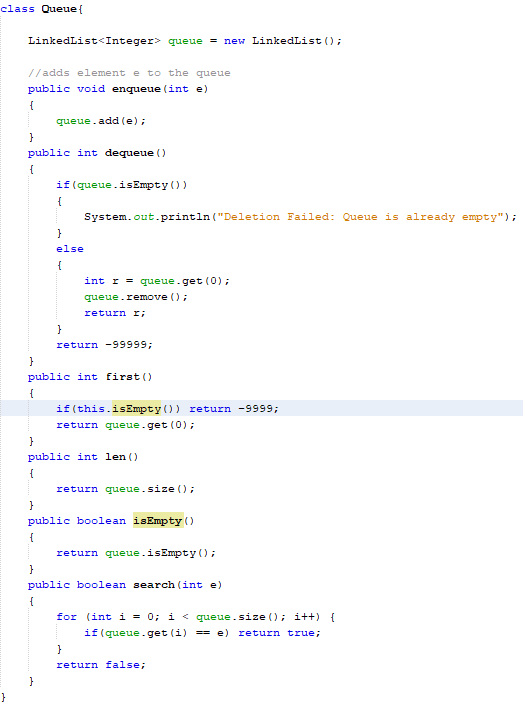




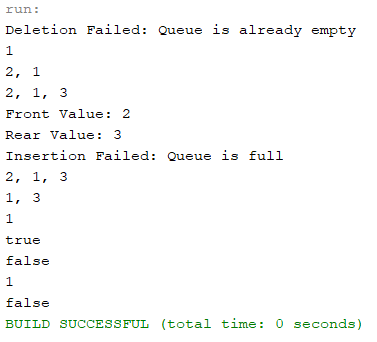
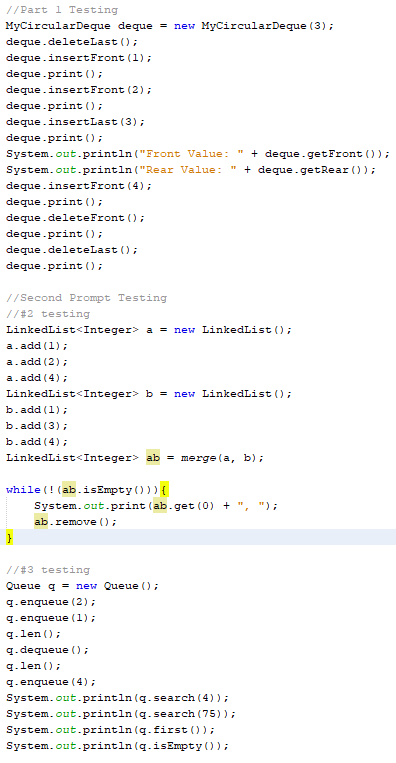
1. My circular Deque is an array with a specific set of rules to it’s use. It is initialized at a given size k. InsertFront(e) fails if the queue is already full to it’s max size, or moves any existing elements to the right one space, before the front value is set to the specified value, and the ind variable(which references the next index which will be added) is incremented by one since the number of elements within the queue has increased by one. InsertLast(e), similarly fails if the queue is full, and it simply adds the element into queue[ind] and then increments ind by 1. DeleteFront fails if the queue is empty, and shifts all values in the queue to the left, thus removing the first element from the queue, and sets whatever the last value was to be a 0. Ind is then decreased by 1. DeleteRear also fails if the queue is empty, and decreases ind by 1 (i.e. From 3 to 2), and so whatever value was in that place in the queue(3) will not be seen again until it is reset with a new value. Print() simply loops through the elements in the queue and outputs them all, used to show the user the state of the queue. Get front returns the element at the beginning of the queue, and fails if the queue is empty. GetRear fails with an empty set, and returns the last element to be added to the queue. IsEmpty and IsFull determine if a queue is empty or full, respectively.



1. The merge algorithm was implemented as it’s own method., and assumes the given lists are pre-sorted. If either given list is empty, it takes the remainder of the no-empty list and puts it into the result list. If neither list is empty it determines which of the next elements is smaller and adds that element to the result list. If they are equal, it takes one of the values and adds it to the result list. If both lists are empty, the loop ends and you have your result. As elements are added to the result list, they are removed from their list of origin



1. This queue implimentation allows the addition of any integer e at the end of the queue, and the deletion of any element at the beginning of the queue. The deletion of an element fails if the queue is empty, returning a value of -99999, and returns the value removed if successful. First() returns the element at the start of the queue, and fails if the queue is empty, thus returning -9999.Len returns the length of the queue, where isEmpty determines if that length is 0. Search returns true if a specified element e is found in the queue, and false if it is not



This is the testing of my implementation, and all that it returns