Mauricio S. Perez
3.22.2016
Professor Conrad
CSC 18C Quiz #2 (25 pts) – March 15th, 2016

All answers must be of your own work. Type out your answers and submit your quiz through Blackboard for Quiz #2 as either a MS Word document or PDF file (Mac users using Pages, make sure to Export to MS Word).

1.) (15 points) When considering time complexity and space (storage) complexity, which sorting algorithm would be better, **merge sort** or **heap sort**? Justify your **answer in your own words in a minimum of five sentences**!

merge sort: Big O – n Log n **heap sort:** Big O – n Log n.

Heap Sort relies on a heap tree, where Merge Sort is more or less an array. Where each are of big o of n Log n, the time and efficiency is strictly dependent of how unsorted the elements being sorted are. While an unsorted heap will vary the heap sort significantly, the merge sort always splits and sorts each half to bring them together in the end. The heap sort will constantly compare and sort top to bottom of the tree, in my opinion making it the more time consuming.

2.) (10 points) In the add method for the LinkedList demonstrated in class, fix the code so that it adds the new node to the end of the LinkedList in constant time (the Big O(1) – hint: the while loop is the problem).

```
// add a node with the specified element to the end of this list
  public void add(int dataValue)
  {
      Node temp = new Node(dataValue);
      Node current = head;
      // start at the head node, loop to the end of the list
      while (current.getNext() != null) {
          current = current.getNext();
      }
      // when we reach the end of our current list
      // set the current node's next
      // "pointer" to temp
      current.setNext(temp);
      // increment our counter for number of elements in linked list
      listCount++;
  }
```

NO SOLUTION ATTEMPTED