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**!

When looking at the time complexity of the heap and merge sorts, the outcome of both is the same BigO(n log n) performance; however, when you take into account the space complexity, this is where the heap sort wins out. The reason for this is because the heap sort is sorting in place by comparing values with the parent value and swapping places. So the extra space needed would be constant (a temporary node). In contrast, when looking at merge sort, the worst case in space requirement is BigO(n).

As to which one would be better, it would have to be the heap sort because it does not require much mode space to perform its function.

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
    current = getLastNode(current);
    // 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++;
}
public Node getLastNode(Node list){
    while (list.getNext() != null){
       list = list.getNext();
    }
    return list;
}
```