Erick Perchez

80582912

CS 2302 Data Structures

Lab 1

**Introduction:** In this lab we were given one file that defined both a linked list called Node and a list called List. We were supposed to create lists of n length and fill them with random numbers, then sort them using three sorting algorithms.

**Proposed solution:**

1. MergeSort- The premise of merge sort was to split the list into two until there is one element left, then compare that to the next list then put them together. This would repeat until the lists were put into one and they were sorted. To split the list I created a method called SplitList that took the length of the list then divided it by two. The first half would be appended to a new list and the second half to a second list. This then would call the method merge. This method took two lists as parameters, then compared the head.item of each list. The smallest element of that comparison would get appended first, then that head would move onto the next element to be compared to the previous bigger element. This was repeated until one list was completely done, then the rest of the elements would be appended to the new list. In every comparison there was a count as well. O(nlogn)
2. QuickSort- The premise of quicksort was to get a pivot point which in my case was the head. Then the list would split into two, the elements smaller than the pivot would go to a list then the others to another list. After this was done, a recursive call would be done and the process would repeat to both lists until the lists were completely sorted. Then the actual pivot would be appended to the left list so it would be at the end of the list that was smaller than the pivot, so it would be in the right place. A counter was also added in every comparison. O(2^n)
3. BubbleSort- The premise of bubble sort was to go element by element starting at the beginning of a list and comparing the previous element with the next one. If the next element was smaller than the previous, they would swap places, if not, then the elements would keep moving forward in the list. This would also contain a counter to count the comparisons. O(n^2)

**Experimental Results:**

I ran the code with a list of n = 10 all the way up to n = 50. This went to show how efficient quicksort was compared to the other two sorting methods. This showed that the bubblesort method was the most inefficient.

**Conclusion:** I learned a lot on this lab. Efficiency really does make a difference when the method is being ran multiple times, especially when the number of times increases fast. This allowed me to see the effects of loops and to try to avoid them for efficiency,

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class”

