CS 3310 - Data and File Structures

Instructor: Ajay Gupta, Western Michigan University

Lab TA: Yu Guo

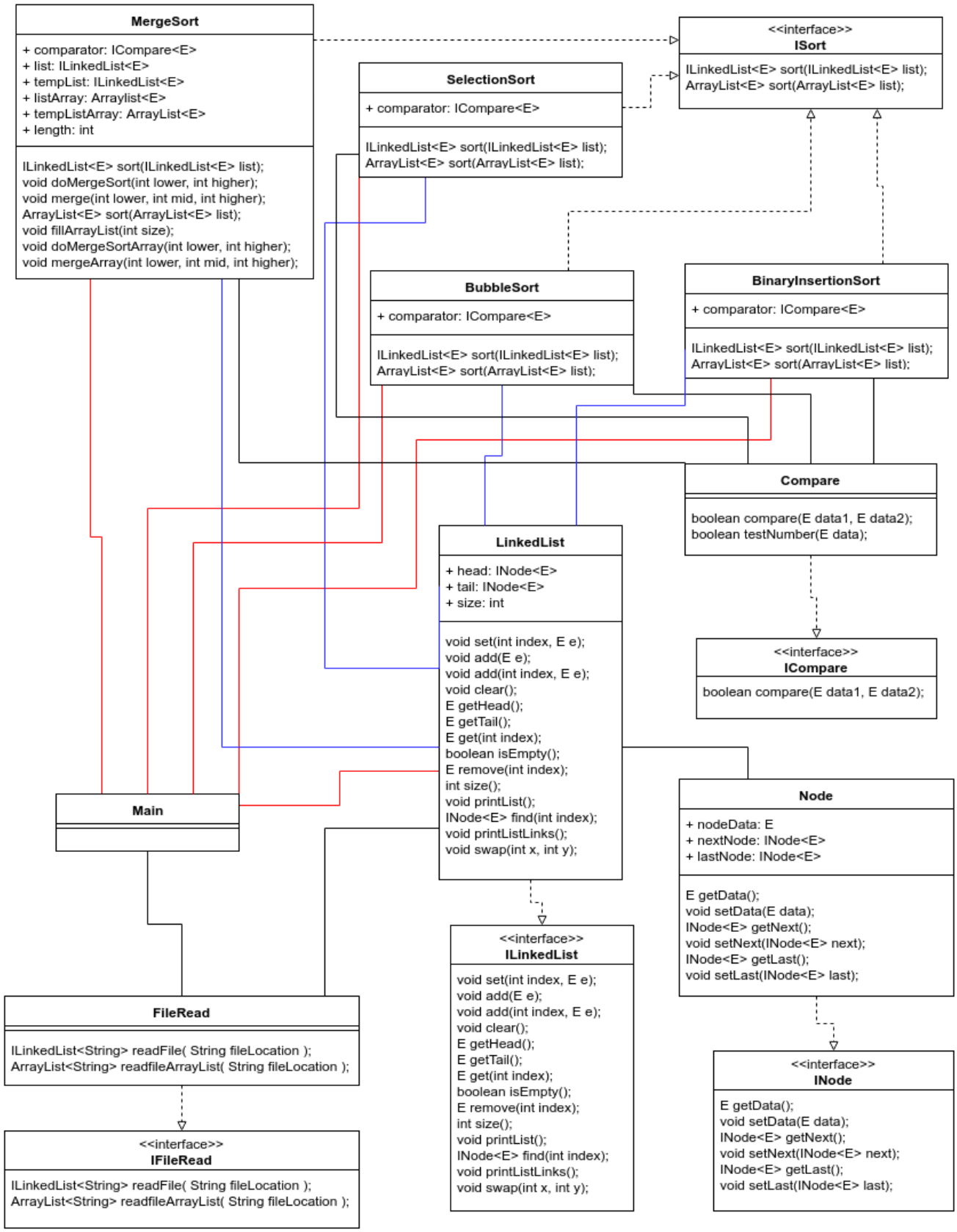
November 10th, 2016

Tyler Thompson

"I do NOT give permission to the instructor to share my solution(s) with the class."

SOFTWARE LIFE CYCLE REPORT - FOR HOMEWORK ASSIGNMENT 4

DESIGN



DESIGN JUSTIFICATION

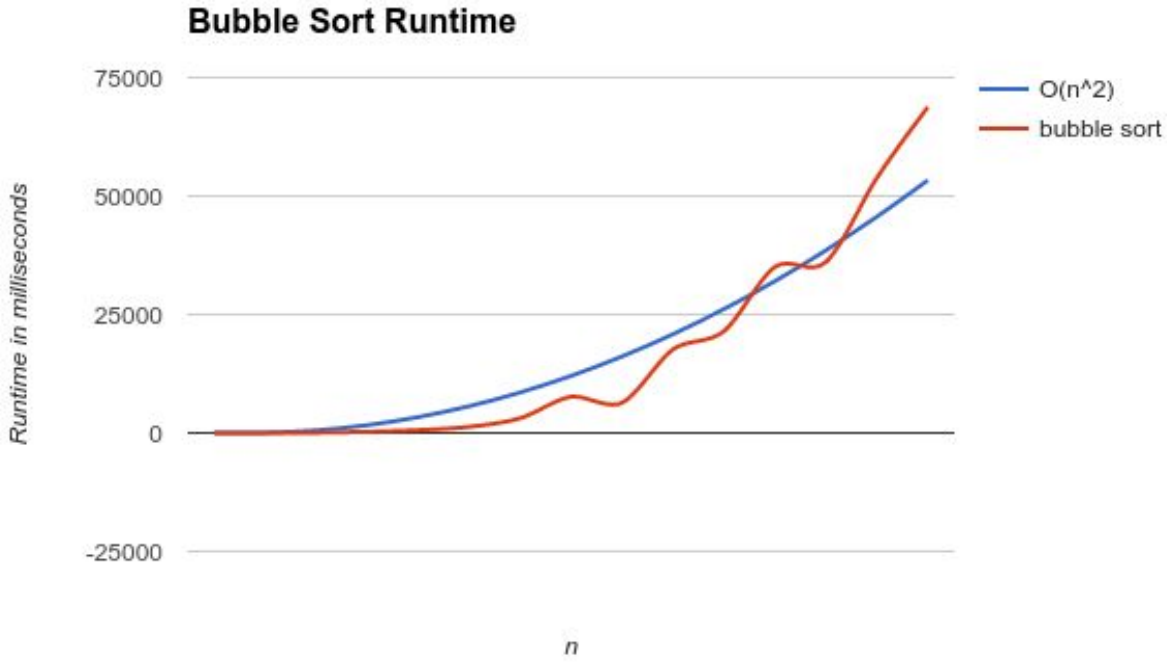
The design of my program is laid out in the UML diagram above. The red and blue lines should be treated the same as the black and are only different to help distinguish the difference between them and the line they cross. The main method does not show an methods because those methods are irrelevant to understanding the structure of the program. In reality the main method runs various tests through the different sorting algorithms.

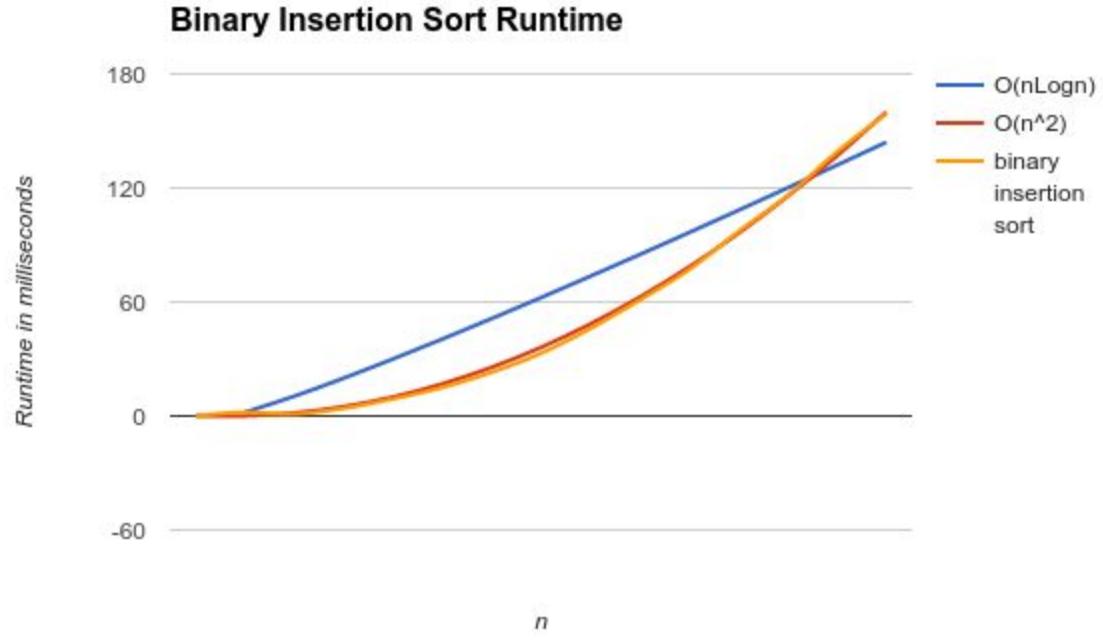
THEORETICAL COMPLEXITY ANALYSIS

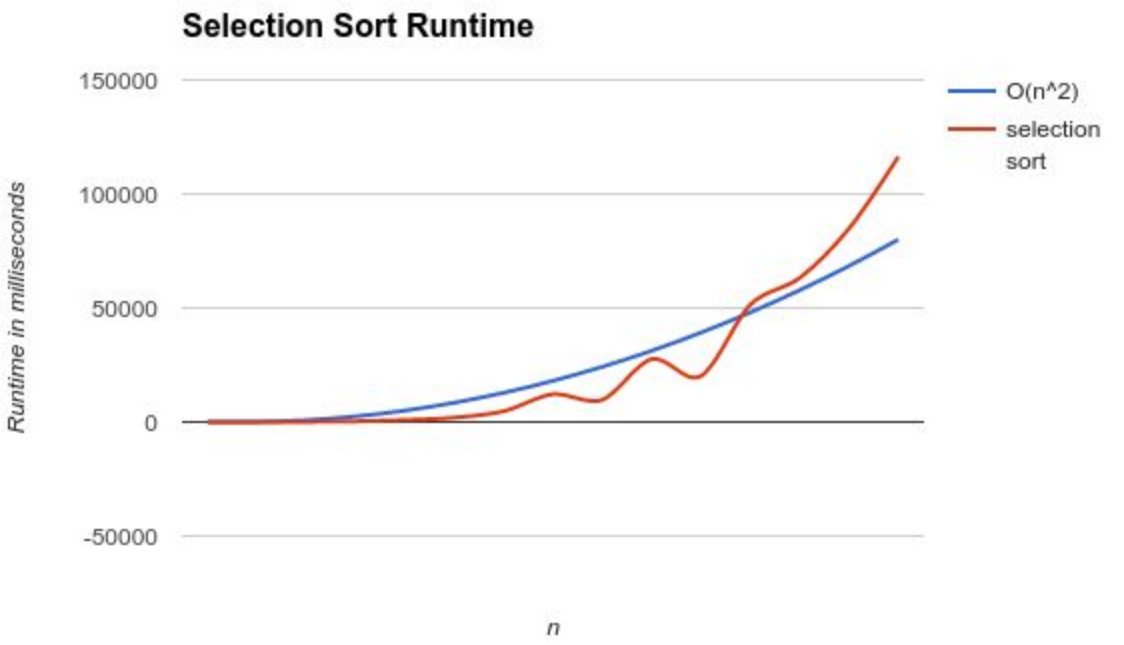
- **Merge Sort**
  This sort has a time complexity for all cases of O(n Log(n)). It has a recurrence relation of T(n) = 2T(n/2) + n and has a space complexity of O(Log(n)).
- **Binary Insertion Sort**
  This sort has a worst & average case runtime of O(n^2) and a best case runtime of O(n Log(n)). The recurrence relation is T(n) = T(n-1) + T(n/2) + n and has a space complexity of O(n).
- **Selection Sort**
  This sort has a time complexity of O(n^2) for all cases and a space complexity of O(n). It has a recurrence relation of T(n) = T(n-1) + n.
- **Bubble Sort**
  This sort has a best case time complexity of O(n) and an average & worst case of O(n^2). Its space complexity is O(1) and has a recurrence relation of T(n) = T((n (n-1))/2)

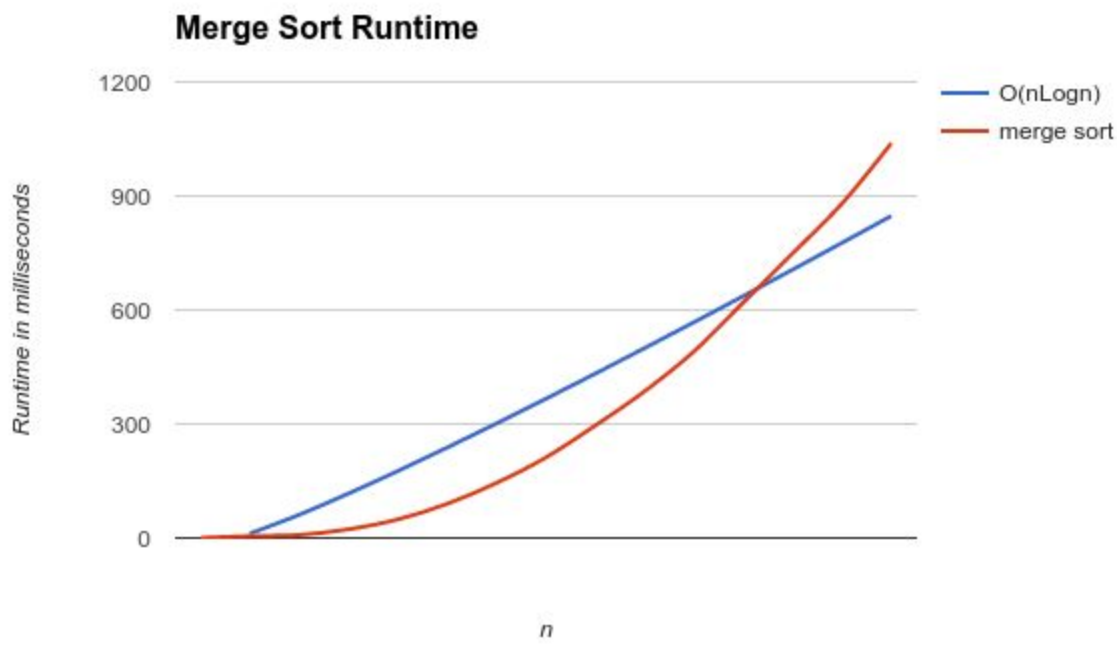
EMPIRICAL COMPLEXITY ANALYSIS

The assignment had us implement each sort using a LinkedList built by us, as well as, the built in ArrayList. Without a doubt, the ArrayList implementation runs much faster than the LinkedList implementation. This also made the sorting algorithms run slower with a more reasonable list size when using a LinkedList and because of this, the following data presented on the graphs was calculated using a LinkedList implementation. For testing purposes, lists of randomly generated integers were used to calculate these runtimes. Notice that on each graph the runtime of the sorting method is being compared to their corresponding Big O.


Bubble Sort Runtime
75000
50000
25000
0
-25000
Runtime in milliseconds
n
O(n^2)
bubble sort


Binary Insertion Sort Runtime
180
120
60
0
-60
Runtime in milliseconds
n
O(nLogn)
O(n^2)
binary insertion sort


Selection Sort Runtime
150000
100000
50000
0
-50000
Runtime in milliseconds
n
O(n^2)
selection sort


Merge Sort Runtime
1200
900
600
300
0
Runtime in milliseconds
n
O(nLogn)
merge sort