Benchmarking Report

Section A. Problem Statement

- We do this Benchmarking project to understand the fundamental of LinkedList by implementing insertion, findMax, findMed, and findMin algorithms; We also learn the procedures of analyzing the performance of the program by implementing the time_insert, time_max, time_min, and time_med.

Section B. Algorithm Design

Insertion: The difficult part of this project is insertion, because we have to make the LinkedList sorted at any given point of the insertion. The most intuitive idea that comes in mind is to compare the read-in value with the existing LinkedList(Java Library)'s element from head to tail, if the read-in value is smaller than a specific element than use the add(index, val) in Java Library to link the read-in value to the front of the current element. However, this approach works for input1.txt but got no response after long time execution for input2.txt. I then implementing the singly LinkedList in my own, the idea is the same, for insertion I will compare the read-in value to the existing linked node's value, if read-in value is smaller than a specific node's value I will performing the previous node's next pointer to the read-in value and read-in value's next pointer to that specific node to achieve inserting at the front of that specific node. By implementing my own singly LinkedList input2.txt finally works. Overall, the idea of my two approaches is the same, the reason of first approach not working for input2.txt might be the runtime error because of the big load of data, because we have to first looping through the list then call the add method to insert so then the runtime complexity of the insertion will be O(N*N) = O(N^2). While the second approaches we traverse the link by the next pointer build in side of the insertion method and once we encounter the specific node that is greater than the data pass in, we just adjusting the pointer to achieve insertion.

For this algorithm(the second approach) I would say its runtime is O(N)

FindMax: Since we've made the LinkedList sorted, the max value of course is the last node, my algorithm of FindMax traverse to the tail node and return the tail node's value, I would say this runtime is O(N).

FindMin: Since we've made the LinkedList sorted, the min value of course is the first node, my algorithm simply just returns the head node's value. I would say this runtime is O(1).

FindMed: Two cases to find median. If the size of numbers is even the formula is $\left[\left(\frac{n}{2}\right) + \left(\frac{n}{2} + 1\right)\right)\ position's\ value * \frac{1}{2}$ and when size of numbers is odd the formula is $\left[\left(\frac{n+1}{2}\right)\right)\ position's\ value$. Notice that the n = size of number = list.size(), but the position need to be -1 to be balanced because in List we have additional number in index of 0. For example, [1,1,2,5,6] the median is = 2, and applied formula (n+1)/2 = 7/2= 3 then for index 3, the value is = 5. We actually get mess up, so to be accurate position must be (n+1)/2 -1

= 2 for index 2; Therefore, my implementation of finding median in codes is $\left[\left(\frac{n}{2}\right) - 1 + (\frac{n}{2} + 1 - 1)\right]$ $position's\ value * \frac{1}{2}$ for even size, and $\left[\left(\frac{n+1}{2}\right) - 1\right]$ $position's\ value$ for odd size. For this algorithm I would say it runtime is O(n/2) =O(n)

Section C. Experiment Setup

- **CPU:** Inter I7-8500 @1.80 Ghz @1.99 Ghz, **RAM:** 8GB, **Hard Drive:** 256 SSD
- Three times of experiment, the final result is averaged from them.
- **Operating System:** window 10, **Compiler Environment:** javac version

Section D. Experimental Result & Discussion

Average Result for input 1.txt

	Time		
Time_Insert	5651399 (ns)		
Time_Max	17300 (ns)	Max value	4000
Time_Min	1100 (ns)	Min value	1
Time_Med	14900 (ns)	Med value	2056.0

Average Result for input 2.txt

	Time		
Time_Insert	47165048700 (ns)		
Time_Max	4283000 (ns)	Max value	7999707
Time_Min	4301 (ns)	Min value	75
Time_Med	2864700 (ns)	Med value	3998801.0

The above result shows that time_insert is significantly large when we are doing the experiment for input 2.txt compare to the input1.txt. So, as mention in Section B. that the insertion method runtime is O(N), however, my method of insertion is actually nested inside of the BufferReader and my time_insert is actually the time calculated for all values inserted(not for one insertion), so more accurately said time_insert is calculated base on the runtime complexity of O(N^2) (this is the runtime complexity for a whole completed insertion of the data from the txt file, not for a single value insertion).

Overall, other than of stated that time_insert is time measured for all value inserted, time_max, time_min, and time_med, shows a very reasonable result comparing to the runtime complexity we've discussed in the Section B. algorithm design.