A: *Problem statement :*

The goal of this project is to become more familiar with the programming and submission environments in this course and tasked with benchmarking the performance of a linked list. Programatically, this project aims to read and sort a large text file of integers, return the value and run time for the maximum, minimum, and median from the output linked list.

B: Algorithm design:

I used the Java standard library LinkedList class to reduce test and debug time. The project requires the linked list should contain 'sorted' values at any point, then I create an insert method to accomplish it. This insert method will add the first element into the linked list from the read-in values, compare each next read-in value with the existing linked list elements from head to tail. I first used a for loop within the insert method to go through the linked list each insertion time to compare and add the elements in the appropriate index. However, a very poor timing performance occurs for the input2.txt. The time complexity for this algorithm with a for loop takes $O(N\hat{}3)$, which is too inefficient to dealing with a large input file. Instead of using the for loop and adding each element within the insert method, I used a linked list iterator for integer and while loop to compare the read-in element with each existing element in the linked list and increase the index. Adding the elements by index outside the while loop will simply reduce the time complexity. The time complexity for this new design became $O(N\hat{}2 + N) = O(N\hat{}2)$, improves efficiency from the previous one.

To find the minimum, I first used the get(int index) method from the java standard library for LinkedList, set the index to zero, and get the minimum element. The run-time complexity for get(int index) is $O(N)$, but $O(1)$ for index = 0. However, by testing and comparing, getFirst() method will be a bit more time-efficient. I used getFirst() method for finding the minimum and getLast() method for finding the maximum element. To find the median, we need to get the size of the list and split the cases into odd and even values. If the size is an even value, use the get(int index) method to get the value for the index of size divided by 2 and the value for the index of size divided by 2 minus 1, divide these two got values by 2. If the size is an odd value, use the get(int index) method to get the value for the index of size divide by 2 then divide the got value by 2.

C: Experimental setup:

CPU：Apple M1

Processor Speed: 3.2GHz

RAM: 8GB

Hardware Type: 500 SSD

Repeat four times before reporting the final timing statistics.

Operating System: macOS Big Sur

Complier Environment: Javac

D: Experimental Results & Discussion:

Input1:

Unit: Nanosecond	Trial #1	Trial #2	Trial #3	Trial#4	Average
Min	1	1	1	1	1
Max	4000	4000	4000	4000	4000
Median	2056	2056	2056	2056	2056
Insert Time	43202125	253978709	313087083	268315250	219645792
Find Max Time	1875	1666	2000	1709	1813
Find Min Time	3583	4500	4166	167	3079
Find Median Time	21250	21793	56625	17333	29250

Input2:

Unit: Nanosecond	Trial #1	Trial #2	Trial #3	Trial#4	Average
Min	75	75	75	75	75
Max	7999707	7999707	7999707	7999707	7999707

Median	1999400	1999400	1999400	1999400	1999400
Insert Time	32657367500	30826683417	31012148083	335389864116	32008796350
Find Max Time	6959	9500	9292	7584	6198
Find Min Time	3208	3125	3084	9958	4844
Find Median Time	214375	230083	231791	213333	22396