Phearak Both Bunna

Report for PA1 (Benchmarking)

Problem Statement:

The goal of this programming assignment is to benchmark or in other words to compare the time it takes for the program/algorithm to read integers from input files and store it in a singly linked list by keeping the list sorted at all time as well as the time to find minimum, maximum and median from the list. We can test it by comparing the results from 2 different input files (by which one of them contains a lot more elements than the other one) and analyze it.

Algorithm design:

- Sorting: I decided to create my own linked list class, and sort function for this project. In order for the list to be sorted at all time (sort each time new element is added), I need to first let the 1st element added to the list to be the head of the linked list. After that, all I do is to use a while loop to compare the next element with the one in the head node and keep the smaller one in the head. So, this will make sure every time an element is added, the list will remain sorted. This repeats until all elements are added.

- Min: I think that finding the minimum number from the sorted (ascending) singly linked list is really simple because I know that the number in the front of the list (head of the list) will be our minimum number and therefore, all I have to do is to return the value store in the head node of the list (It could've been much worst if I decided to use while loops and compare all the numbers again to find the min)

- Max: Finding the max number requires more work than finding min. I decided to use a while loop in order to move throughout the list until I reach the last element in that list because the last element will be the largest number in a sorted singly linked list

- Median: I initialized 3 nodes as the head to make my finding less complex. Also, I used a while loop to move through the list until we run out of elements. Each time the loop runs, the last node is the same as the 1st node, the 1st node points to the next and the 2nd node points to the element after the next (meaning next.next). There will be 2 cases for finding median.
  When the length of the list is odd, all I have to do is return the data in 1st node.
  But when the length of the list is even, then I have to return (the addition of integer in the 1st node with the integer in the 3rd node) / 2 (two elements in the middle add together and divide by 2).

Experimental setup:

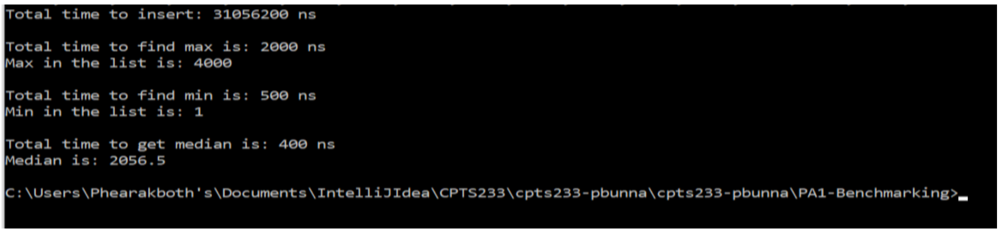
For the experiment, I used an Intel Core i5 CPU @2.5GHz, 256GB Solid State Drive (SSD) laptop with an 8GB ram.

In order to be as accurate as possible, I ran the experiment 6 times before reporting my final timing statistics.

I wrote my code on IntelliJ IDE and test the program on my Windows Command Prompt.

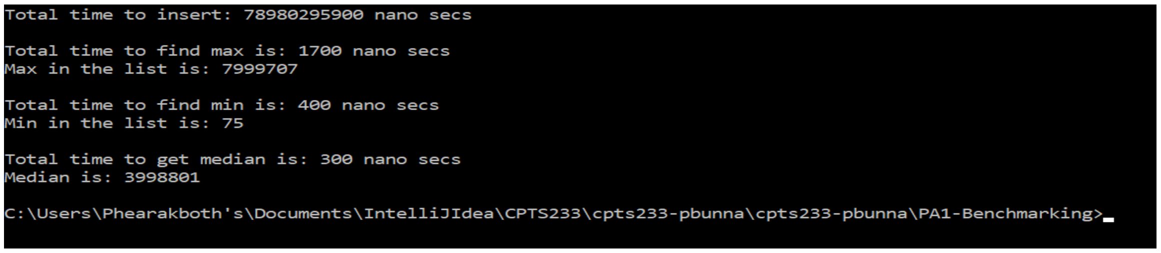
Experimental Results & Discussion: (my time is in nanoseconds)

- Input1:
    - time_insert = 31056200 ns
    - Max = 4000 & time_max = 2000 ns
    - Min = 1 & time_min = 500 ns
    - Median = 2056.5 & time_med = 400ns



```
Total time to insert: 31056200 ns

Total time to find max is: 2000 ns
Max in the list is: 4000

Total time to find min is: 500 ns
Min in the list is: 1

Total time to get median is: 400 ns
Median is: 2056.5

C:\Users\Phearakboth's\Documents\IntelliJIdea\CPTS233\cpts233-pbunna\cpts233-pbunna\PA1-Benchmarking>
```

- Input2:
    - time_insert = 78980295900 ns
    - Max = 7999707 & time_max = 1700 ns
    - Min = 75 & time_min = 400 ns
    - Median = 3998801 & time_med = 300 ns



```
Total time to insert: 78980295900 nano secs

Total time to find max is: 1700 nano secs
Max in the list is: 7999707

Total time to find min is: 400 nano secs
Min in the list is: 75

Total time to get median is: 300 nano secs
Median is: 3998801

C:\Users\Phearakboth's\Documents\IntelliJIdea\CPTS233\cpts233-pbunna\cpts233-pbunna\PA1-Benchmarking>
```

When comparing the results from input 1 with input 2, I can say that my codes and algorithm are efficient because the time in execution time in nanoseconds are not that much difference when finding max, min and median. They are almost identical. For the time taken to sort the linked list, it takes a bit longer to run input2 (took around 78 seconds to run) because there are a lot more elements in input2 and the run time complexity of the algorithm needed to sort the singly linked list can't be constant. That's the algorithm with the best runtime I can come up with.