**Joseph Lewis**

**CS 233**

**PA1 Report**

**11567186**

**8/28/2018**

Problem Statement:

The main goal of this problem is to create an algorithm that would read a text file filled with numerical values and organize the numbers in a linked list. Then we had to figure out the largest number, smallest number, and median number in the list. Also, we needed to figure out how much time was used inserting the data from the text file in the list, finding the max value, locating the minimum value, and solving the median value.

Algorithm design:

To figure out the ‘min’, ‘max’, and ‘med’ I first had to create a function that would insert the nodes in the list in order by numeric values. I designed my algorithm to insert nodes in the list from least to greatest by comparing the nodes consecutively in the list until the newest, unadded node is less than a list node already in the list using a while loop. The easiest algorithm was finding the minimum because the head node of the list already had the lowest number. I had to design a getter function, named “getnumber()”, for the list node class. This allowed me to receive an integer in the list nodes. The overall function that found the minimum value only returned the number inside the list node ‘mpHead’ using “getnumber()”.

My function to find the max required to use a ‘listnode’ called ‘pNext’ to consecutively enter all nodes in the list until it reached the node on the tail of the list. The function returned the integer that was in the ‘listnode’ at the tail of the list using the getter function “getnumber()”.

To create a function that would find the return the median of a list of numbers required the size of the linked list in the parameters. To solve this problem, I used counting methods using “count++.” The syntax ‘count++’ was placed on a while loop that would stop when all the numbers where read in the text inputs. After I was able find the size of the list I had to figure out if the amount of entries in the linked list was odd or even using an if statement. To figure this problem out I had to use “x % 2”, which would equal a 0 if and only if the amount of entries in the linked list was even. If the linked list was odd, then I used ‘pNext’ to consecutively pass each node in the list until I was able to obtain the value of the number in the middle of the list. This process required a for loop that was like, “for (int a; a <= (size/2); a++).” After the for loop, the value of the median would be returned from the Node in the middle of the list. If the list was even, we would do the same exact steps, but we would also have to find the value of the number inside the list node after the last node found using the described for loop. To complete this, I assigned an integer to the number in the current node after the for-loop execution. After that, I moved on to the next node and I used “getnumber()” to obtain that numerical value in that list node. (using pCur=pCur->getNext()) Then I obtained the two integer values and found the average by coding basic arithmetic. The function will return the average of the two integers, in the middle of the list, if the size was even.

An alternative solution I could have done is create an integer in the list class that would keep track of the largest integer in the list while inserting nodes in the list in order, which would be the number inside the back node of the list. I would been able to include this solution inside the function that add nodes to the list in order inside the if statement saying “if(pCur-getnext() == nullptr)”. This if statement determines if the current node under comparison is the tail node. This solution would not have required me to browse through every node consecutively in the list to find the largest number, which would have saved some time.

Experimental Setup:

To set up this experiment, I had to use the ssh3.eecs server to code the algorithm in C++. I created two classes to design the required list named ‘list’ and ‘listnode’. The class ‘listnode’ had a constructor, setters, and getters. The numeric values from the text files and the listnode “pNext” where inside the private parameters of the “listnode” class. The class ‘list’ private parameters contained a listnode called “mpHead” and two functions used to delete the list. The ‘list’ class contained a deep copy constructor, a destructor, a setter for mpHead, a getter for mpHead, and a function that inserted the nodes in numerical order from least to greatest. The functions returning the ‘med’, ‘max’, and ‘min’ are also in the public parameters of the ‘list’ class. The main function used an ifstream and asked the user to type in file to search the text using the entered result as a string. The file found in this process is scanned receiving numeric values that enter in the list as listnodes in order from least to greatest. Also, an integer must be keeping track on how much nodes are in the list to calculate the median. After the reading process of the text file, the list functions used to determine ‘med’, ‘min’, ‘max’ are declared. Time it takes to insert the numerical values, find the ‘min’, find the ‘max’, and find the ‘med’ was also recorded in seconds. The lowest digit of time recorded is by a centi-second. After all the data has been recorded, the program prints out the results. I used an application named “Bitvise SSH” to access the “ssh3.eecs” server on a Windows 10 PC. Using the “ssh3.eecs” server, I proceeded to test the code and I used g++ as my compiler environment. I have completed two trials in this experiment for each text file in the programming assignment. After each trial was completed, the data printed out in the program was recorded.

Experimental Results & Discussion:

The first file, “input1.txt”, had the same results in both trials. The first file had a maximum number value of 4000. The minimum value of the first file equaled 1. The median of the first file was 2068. The values of the ‘min’, ‘max’, and ‘med’ made sense since the maximum value has greater than the minimum and the median. Also, the median was less than the minimum. The time it took to insert the numbers in a linked list, to find the minimum value, to find the maximum value, and to find the median value were all less than a centi-second. These time results occurred because the number data entries in “input1.txt” was 1000, which is a small number of entries for the compiler to process.

The second file, “input 2.txt”, had the same results for every value except for the values involving time. The second file had a minimum of 14, a maximum of 7,999,997, and a median of 2068. This result was reasonable because the maximum value has greater than the minimum and the median. Also, the median was less than the minimum. The first test involving time on involving the second file had the insertion time equal to 18953.43 seconds, the time to find ‘max’ equal .10 sec, the time to find ‘min’ equal 0.00 seconds, and the time to find ‘med’ equal .05 seconds. The Second test involving time on involving the second file had the insertion time equal to 21259.70 seconds, the time to find ‘max’ equal .09 sec, the time to find ‘min’ equal 0.00 seconds, and the time to find ‘med’ equal .09 seconds. The second file took way longer to insert the numbers in the list than the first file. This is caused to by the Running time being equal to 0(N^2). This means that every data entry entering the list would take more significant amount of time to enter the list compared to the previous insertion. The size the list in the second file test was 1,000,000 which is way larger than the size of “input1.txt.” The minimum should be less than a centi-second because the function is just returning the numeric value from the list node mpHead. The amount of time it took to compute the median was close to half the time required to find the max value in the list. This is caused by both functions being linear, which causes the run time to equal O(N). This means that the amount of time it took to complete the function increasing linearly as the number of nodes in linked list increases.