Programming Project Report

Jaylon Kiper

CS330-01

11/8/20

The efficiency of my code that constructs the lists is Big-Oh(n^2), or quadratic. These three separate lists consist of 1000, 100,000, and 1,000,000 unique integers. The specific amount of unique integers is set to 25, and the list is timed by using the method, Java.lang.System.currentTimeMillis(). This records the production of the list in terms of milliseconds. By subtracting the difference of two seperate times, I have found the time it takes to produce lists back to back. For a list of 1000, it takes 57,217 milliseconds to produce it. Then for 100,000, it takes 119,463 milliseconds, and finally, 20,815 milliseconds for 1,000,000 numbers. By using the asymptotic efficiency of my algorithm, , we will compare the increase in the runtime of my code as n increases to the expected theoretical increase predicted by the asymptotic efficiency class of the algorithm. With 1000 unique integers, there are 87.30 bo/ms(basic operations per millisecond) being created with this list. Considering this list is the smallest, it is reasonable to say the time is growing at the same rate as the expected number of basic operations. Using 100,000 unique integers, there are 41,853.54 bo/ms being created. This seems normal with this result being below the number of n integers being produced within this list. However, with 1,000,000 unique integers, there’s an oddity. The runtime is smaller than both of the previous lists and produces more basic operations per millisecond. The difference in runtime makes this analysis very odd, even though the list is generating pseudorandom numbers between a million in a list. It seems to be exceeding the time needed to produce the list at a fast pace compared to 1000, and 100,000. This results in 23,021,114 bo/ms being produced with 1,000,000 unique integers. The selection sort used for this algorithm organized these lists at an outstanding rate. To gather my results faster, I ran a for loop ten times to gather all my results to save time. I noticed that all ten for each number of n unique integers were sorted in the exact same timeframe in terms of milliseconds. Sorting a list of 1000 had an average runtime of 3.208043087x10^24 milliseconds. This was surprisingly the slowest of the averages. Sorting a list of 100,000 had an average runtime of 1.604022276x10^12 milliseconds. I found that this average shares similar numbers to the basic results beforehand. Finally, with the list of 1,000,000, the sorting process had an average runtime of 1.315298343x10^12 milliseconds. This is the fastest average among all the lists. After repeating the process, the runtimes do tend to change, so the results could have been different over a period of time. The selection sort has an efficiency of O(n^2) , so it makes sense that sorting these lists is growing at the same rate as the theoretical efficiency of the algorithm. In conclusion, this project’s analysis on sorting lists of pseudorandom numbers and finding their average runtimes shows that the size of a list can express the efficiency that they have in being produced.

**Details of Mathematics**

Coding Runtime in Milliseconds:

List of 1000:

1604018248015-1604018190798

=57217 milliseconds

List of 100,000:

1604018429790-1604018310327

=119463 milliseconds

List of 1,000,000:

1604018513381-1604018492566

=20815 milliseconds

Comparing Asymptotic Efficiency To Theoretical:

Asymptotic Efficiency:

n=1000

basic operations

4995000/57217=87.30 bo/ms

n=100000

basic operations

4999950000/119463=41853.54 bo/ms

n=1000000

basic operations

bo/ms

Selection Sort Based Algorithm’s Average Runtime:

n=1000

1-10:1604021543276=3.208043087\*10^25

n=100,000

1-10:1604022275944=1.604022276\*10^13

n=1,000,000

1-10:1604022342998=1.315298343\*10^13