Richie Lazear

Mr. Ritter

Programming Data Structures (H)

December 8, 2014

**AS3 Performance Analysis Report**

This program reads in a text file to an array of a custom object type. This custom object (Word) has a field for the word and a field for the number of times that word is found in the text file. The program then can do a series of sorts and searches and displays the statistics for each so that the different algorithms can be compared.

I chose to make the program convert all of the strings it reads in to lower case and remove punctuation from the words. However, dashes (-) are initially left in so that they can be further processed. This further processing leaves words that are connected by a single dash (ex. “light-colored”), but removes double dashes (“--“) in compound sentences, splits the connected words, and individually processes each word.

**Data**

**As3Small.txt contents:** the kid walks down the street to say hello to his neighbors and their dogs

**Data from sorting array read in from As3Small.txt**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sort | Comparisons | Swaps | Time (nanoseconds) | Big O Worst Case |
| Selection | 81 | 8 | 167,943 | O(n2) |
| Insertion | 63 | 51 | 121,759 | O(n2) |
| Bubble | 78 | 51 | 150,682 | O(n2) |
| Merge | 51 | 19 | 196,400 | O(nlog(n)) |
| Quick | 41 | 28 | 108,697 | O(n2) |

**Data from searching array read in from As3Small.txt**

**Searching for the word “kid”**

|  |  |  |  |
| --- | --- | --- | --- |
| Search | Comparisons | Time (nanoseconds) | Big O Worst Case |
| Linear | 6 | 30,323 | O(n) |
| Binary | 4 | 29,390 | O(log(n)) |
| Quadratic | 5 | 877,505 | O(log2(n)) |

**Data from sorting array read in from “War and Peace” (As3Large.txt)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sort | Comparisons | Swaps | Time (nanoseconds) | Big O Worst Case |
| Selection | 138,089,193 | 10,361 | 1,283,947,479 | O(n2) |
| Insertion | 86,101,855 | 86,083,349 | 1,052,116,650 | O(n2) |
| Bubble | 171,234,096 | 86,083,349 | 3,821,188,052 | O(n2) |
| Merge | 356,924 | 121,054 | 39,375,791 | O(nlogn) |
| Quick | 312,520 | 167,610 | 16,564,843 | O(n2) |

**Data from searching array read in from “War and Peace” (As3Large.txt)**

**Searching for the word “war”**

|  |  |  |  |
| --- | --- | --- | --- |
| Search | Comparisons | Time (nanoseconds) | Big O Worst Case |
| Linear | 17,850 | 14,021,430 | O(n) |
| Binary | 14 | 33,588 | O(log(n)) |
| Quadratic | 166 | 425,457 | O(log2(n)) |

**Observations**

When sorting the array from the small file, the difference in number of comparisons is relatively insignificant. The lowest number, from quick sort, is 41, while the highest number, from selection sort, is 81. Additionally, the difference in the number of swaps is relatively insignificant as well. The least number of swaps, by selection sort, was 8, and the most number of swaps was 51, by both insertion and bubble sort. The times for these algorithms were also fairly close to each other: the fastest algorithm was quick sort, at 108,697 nanoseconds and the slowest algorithm was 196,400 nanoseconds.

When searching the array from the small file, the number of comparisons by each algorithm were very close: 6 for linear, 4 for binary, and 5 for quadratic. The time taken to complete each search, however, was drastically different. Linear and binary were right around 30,000 nanoseconds, but quadratic search took 877,505 nanoseconds to complete.

When sorting the array from the large file, selection and bubble sort were very close in their number of comparisons at around 150,000,000 comparisons. Insertion sort used about half as many comparisons as selection and bubble sort. Merge and quick sorts had around 340,000 comparisons. Selection sort had very few swaps at only 10,361, while insertion and bubble sorts both had 86,083,349, and quick and merge were very close at around 140,000 swaps. Bubble sort took the most time at nearly 3.8 billion nanoseconds, while quick sort took only 16.5 million nanoseconds.

When searching the array from the large file, binary search had only 14 comparisons, quadratic search had 166 comparisons, and linear search had 17,850 comparisons. Binary also was the fastest search method, as it took only 33,588 nanoseconds, while linear took the longest at 14,021,430 nanoseconds.

**Analysis**

Because the smaller text file had so few words, the differences for both the sorts and searches are insignificant. However, for the larger text file, which had close to 600,000 words there were decidedly better and worse algorithms to use. For sorts, merge and quick were by far the fastest, so they are most suited for relatively large arrays. Bubble sort would be the worst choice (from the tested sorts) to use on a relatively large file because of the way it floats larger numbers or strings to the end of the array. For searches on larger arrays, quadratic is faster than linear, but binary is decidedly faster than either algorithm. This is because the binary algorithm will search no more than half of the array, even in the worst case.

**Reflection**

Unfortunately, this project took 3 more weeks than you had wanted. Coming into this project, I had completed all of the sorts and searches, so I was not behind on that aspect of the project. It took so long because I focused a lot on minor details like removing punctuation and handling double dashes while reading in the file, among other things. Additionally, I spent time trying to run the program from the command line so that I could make sure the words were being read in properly. In retrospect, I did not need to spend the time doing it.

I spent a considerable amount of time working on this project from home, but it ultimately took so long because it was the most complicated program I have written to date, and I ended up having to redo large sections of code to make the program more efficient and functional.