Assignment 3 Lab Report 2

Part 2: Experimental Identification of Sorting Algorithms

Robert Skelton && Devin House

09.13.2012

**Abstract.**

Our assignment was to identify the sorting algorithms. In order to determine the algorithm used in each of the individual sort methods, we passed through ascending and descending arrays varying in size (from 10,000-50,000) and recorded the time it took for each method to sort the array. Based on the time it took to sort the array and how the array was ordered we were able to determine the type of algorithm used.

**Introduction.**

In Part 2, we determine what algorithm is used by each method to sort an array. We were provided no information on the source code used in each method and had to determine the algorithm type through experimentation only. Different arrays needed to be passed through to accurately analyze the methods and the algorithms used within them. Based on the speed of the sort of each array the algorithm type will be determined.

**Procedures.**

We first created the arrays to be passed through the methods; a for loop was created in order to make large arrays in an efficient manner. The first test was done with an array of size 10,000 in ascending order; we then increased the size multiple times and recorded times for each of the methods. Next we pass arrays of the same sizes as the ascending arrays; except this time, we passed a descending array through the methods. After all times were recorded we added them to a chart and a graph in order to compare the times and make decisions on which algorithms were used in each method.

**Results and Discussion.**

Through comparing the times of each sort we were able to accurately determine the algorithm used in each method. In sort one, we determined that the best case was ascending while the worst case was descending; we did so because the times for ascending were way faster than the descending times. Based on the discovery, we were able to prove that sort one used the insertion sort algorithm. After further looking at the recorded times we determined that both sort 2 and sort 3 were not affected by the ordering of the array. This discovery meant that one of them was selection and one of them was merge sort. We then compared the speeds of the two; merge sort is a far more efficient sorting algorithm and should have returned a quicker sort time. Based on this information we determined that sort two was selection and sort three was merge. This is because sort three was much faster at sorting the arrays passed through it. It was then determined that sort four was a quicksort without any randomization, and sort five was the quicksort with randomization, or a shuffle. This is because sort four and five were significantly faster than the other sorts, meaning that they would both be the most efficient: quicksort. However, sort five was slightly quicker than sort four, leading us to agree upon sort five having a shuffle. This is because, with a shuffle, the quicksort is less likely to pick a bad pivot value, leading to it having quicker times than a non-shuffled quicksort. Therefore, the sort 4 was a normal quicksort.

**Conclusions.** The purpose of Part 2 of this lab was to learn how to analyze, experiment, and use the scientific method to find out different sorting algorithms without the source code. We learned which algorithms are more efficient in certain scenarios. We also learned how to determine what algorithm is used to sort a list through experimentation.

**Appendices.**

**Table 1 Ascending Array Values**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N | sort1 | sort2 | sort3 | sort4 | sort5 |
|  | 10000 | 0.874 | 1.631 | 0.204 | 0.034 | 0.01 |
|  | 20000 | 3.134 | 7.04 | 0.313 | 0.147 | 0.021 |
|  | 30000 | 8.301 | 15.405 | 0.562 | 0.186 | 0.048 |
|  | 50000 | 11.773 | 27.203 | 0.221 | 0.104 | 0.04 |

**Figure 1 Results of Table 1 in a Graph**

**Table 1 Descending Array Values**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | sort1 | sort2 | sort3 | sort4 | sort5 |
| 10000 | 0.443 | 0.91 | 0.162 | 0.032 | 0.008 |
| 20000 | 1.714 | 3.816 | 0.154 | 0.07 | 0.021 |
| 30000 | 3.938 | 9.101 | 0.195 | 0.08 | 0.03 |
| 50000 | 11.925 | 26.797 | 0.214 | 0.101 | 0.04 |

**Figure 1 Results of Table 2 in a Graph**