Alexander Apollo Salazar Professor Ana Salvati

CSC 430 10/4

Project 1

**Sorting algorithms analysis report**

Intro: In this report, I will be using six sorting algorithms. Each algorithm will be fed six different datasets; The purpose is to observe the behavior of each algorithm by recording all of the comparisons and swaps that are made, and then with the empirical data determine which algorithm is more “efficient”.

**Selection Sort**

Selection sort seems to crumble with huge arrays, the amount of comparisons and swaps that are made is increased by a multiple of a 100 which can be seen below on the data chart.

Inverse 100 and 1000 & the Random 100 and 1000 all yield the same number of comparisons.

The number of swaps are the only thing that differs between the four sets, that is of course dependent on the contents of the array. For both the almost 100 and 1000 no swaps are made, the contents are linear from 1-100 and 1-1000, what I mean by that is none of the numbers are unsorted most of them are already sorted by the design of the type of file that it is. With that being said the algorithm makes no unnecessary swaps Ergo, no swaps are counted for. I am certain that if almost 100 txt files contained 100 or 1000 integers, the comparison count would be 4950, or 499500 across the board. This algorithm is considered to be unstable.

Best Case Performance: O(n2)

Worst Case Performance: O(n2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Selection Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 4950 | 499500 | 3570 | 305371 | 4950 | 499500 |
| Swaps | 4950 | 499500 | 0 | 0 | 1124 | 11129 |

**Insertion Sort**

Insertion sort is very much like a cousin to selection sort, for the inverse 100 and 1000 the comparison count is almost homogenous to that of the selection sort. Insertion makes less comparisons than that of bubble, and selection sort. However, for insertion, comparing and swapping are sort of like a unit. The only set where this behavior seems irrelevant is in both almost 100 & 1000 files. This algorithm is in my opinion an improvement on selection and bubble. This algorithm is considered to be stable, and performs best with smaller arrays, that are nearly sorted.

Best Case Performance: O(n)

Worst Case Performance: O(n2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Insertion Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 4951 | 499501 | 84 | 781 | 2519 | 250430 |
| Swaps | 4950 | 499500 | 0 | 0 | 2514 | 250424 |

**Bubble Sort**

I find nothing romantic to say about Bubble sort. It is an egregious algorithm; just like selection sort, it too grows by a multiple of a 100. The only nice thing to say is that it doesn’t make unnecessary swaps with an almost sorted 100 & 1000 array. Since the contents of a Random txt file are purely arbitrary, not much can be said except of course that it’s still bad to see that on a 1000 element sorting it’s still almost half a million comparisons. This algorithm is considered to be unstable.

Best Case Performance: O(n2)

Worst Case Performance: O(n2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bubble Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 4950 | 499500 | 84 | 781 | 4845 | 498324 |
| Swaps | 4950 | 499500 | 0 | 0 | 2514 | 250424 |

**Heap Sort**

Heap sort is a comparison based sorting algorithm, it is part of the selection sort family. Heap sort is like the combination of Merge sort, and quick sort. It uses time efficiently like merge sort, and storage efficiently like quick sort. It is the best performer in the selection sort family, but it is considered unstable. In terms of comparisons, Heap sort did beautifully, In the inverse 100, Random 100, and Almost 100. Those three almost have the same number of comparisons except for random which came out 1252. Heap sort started to crumble with all the txt files that have a 1000 elements, this leads me to conclude that although it doesn’t take as much time as other selection sorting algorithms, it does not perform well with huge data sets that are random in nature.

Best Case Performance: O(nlogn)

Worst Case Performance: O(nlogn)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Heap Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 1134 | 17634 | 1134 | 15532 | 1252 | 19218 |
| Swaps | 517 | 8317 | 525 | 7375 | 576 | 9109 |

**Merge Sort**

Merge sort is known for using the “Divide & conquer technique”, the way that phrase is relevant to the method is that it breaks an array into two sub lists and sorts them, then it merges those sub lists into one sorted array. In this experiment merge sort performed the worst with the inverse 100, and the random 1000. The reason is because it has to compare and swap everything in those sub lists then merge by comparing. Those specific txt files have both arbitrary integers and numbers running inversely. This makes it all the more difficult for the algorithm that requires small subsets of itself to be sorted before merged. For the Almost 100 & 1000 although the numbers in that file are already sorted, merge sort still requires the array to be broken up, compared and re-sorted. Regardless of this behavior, merge sort is still considered a stable algorithm that doesn’t deteriorate over a huge data sets.

Best Case Performance: O(nlogn)

Worst Case Performance: O(nlogn)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Merge Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 680 | 9987 | 560 | 7589 | 680 | 9987 |
| Swaps | 320 | 4939 | 301 | 3952 | 549 | 8753 |

**Quick Sort**

Quick sort, in theory is considered to be an unstable algorithm. However, out of all the algorithms I used in this experiment, comparatively Quick sort performed the best on every occasion. Its most difficult sort was with random 1000, summing almost 10000+ comparisons, and over 6500+ swaps. Like merge sort, Quick sort also implements the divide and conquer technique. However, its method differs with its use of partitioning. The disadvantage is that quick sort requires O(n) extra storage space, which can make it as bad as merge sort. The need of extra space causes the algorithm to take longer to sort.

Best Case Performance: O(nlogn)

Worst Case Performance: O(n2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Quick Sort | Inverse 100 | Inverse 1000 | Almost 100 | Almost 1000 | Random 100 | Random 1000 |
| Comparisons | 531 | 8653 | 390 | 6025 | 606 | 10858 |
| Swaps | 407 | 5772 | 280 | 3882 | 415 | 6608 |