Erik Gabrielsen

CSE 2341

Program 3: Sorting Competition

**BOZO Sort:**

The first algorithm I tried was a Bozo sort that would randomly switch elements in the array until the order was correct. This type of algorithm works with very small data sets but takes an infinite amount of time with large data sets, such as one with more than 100 elements in the array. With sorting a file size of 4.4MB or 800,000 words, it took my program more than 10 minutes to run and I inevitably had to terminate it.

Moral of the story is, don’t use Bozo sort, ever.

**Selection Sort:**

After the failure that was bozo sort, I implemented selection sort, of BigO(n^2).

This sorting method requires the computer to go through every element of the array n times in order to compare them to sort. This algorithm method is useful for small and medium data sets but for larger sets about a few MB, the run time for the program took more than a few seconds

Data: Sorting data set of 2.1 MB, 262,000 words

Trials Time

|  |  |
| --- | --- |
| Trial 1 | 438 seconds |
| Trial 2 | 451 seconds |
| Trail 3 | 432 seconds |
| Trial 4 | 408 seconds |
| Trail 5 | 425 seconds |

AVG Run time: 430.8 seconds

**Quicksort (Partition first element):**

This implementation would take the first element of the array and divide the following elements into groups based on whether the value is greater than or smaller then the partition value. This method is really fast for small and medium data sets but again it struggles to produce good results with large data sets, taking a little over a minute for each computation. This method uses a BigO(n\*logn) at best but in the worse case is BigO(n^2).

Data: Using data set of 2.1 MB, 262,000 words

Trial Time

|  |  |
| --- | --- |
| Trial 1 | 85.416 seconds |
| Trial 2 | 75.590 seconds |
| Trial 3 | 76.100 seconds |
| Trial 4 | 76.080 seconds |
| Trial 5 | 76.510 seconds |

AVG Run Time: 77.939 seconds

**Merge Sort:**

Merge sort will first divide the list into the smallest unit element (one word) then compare each element with its neighbor and merge the two together in the correct order. It will continue to run through this process at varying levels until it returns back to the full array, which will be sorted. Performs at BigO(n\*logn) and worst case at BigO(n). Performed better than regular Quicksort but still not by much

Data: Sorting the Bible, 4.4 MB, 800,000 words

Trial Time

|  |  |
| --- | --- |
| Trial 1 | 60.576 seconds |
| Trial 2 | 59.927 seconds |
| Trial 3 | 61.287 seconds |
| Trail 4 | 60.569 seconds |
| Trail 5 | 60.787 seconds |

AVG Run Time: 60.628 seconds

**Quicksort (Partition by 3):**

Quicksort with partition by 3 takes 3 partition numbers in a set of data and compares them to get the best partition, then separates unites based on whether it is greater than or less than the partition value and continues this process in BigO(n\*logn). This works really well on both big and small sets of data.

Data: Sorting the Bible, 4.4 MB, 800,000 words

Trial Time

|  |  |
| --- | --- |
| Trail 1 | 519 milliseconds |
| Trial 2 | 522 milliseconds |
| Trial 3 | 512 milliseconds |
| Trial 4 | 515 milliseconds |
| Trial 5 | 508 milliseconds |

AVG Run Time: 515 milliseconds