4)

a)

**Sort1: Insertion Sort:**

Experimentally, the insertion has runtime complexity of aO(n^2) + bx + c => O(n^2). The constants are: a= 1E-10, b=5E-07, and c=- 2E-05.

**Sort2: Merge Sort:**

Experimentally, the merge sort has runtime complexity of aO(nlog(n)) + c => O(nlog(n)). The constants are: a=2E-08 and c = 2E-07

**Sort3: Bubble Sort:**

Experimentally, the bubble sort has runtime complexity of aO(n^2) + bx + c => O(n^2). The constants are: a = 1E-09, b = 3E-08, and c = 0.0005.

b)

|  |  |
| --- | --- |
| x | f(x) |
| 10 | 0.000002 |
| 50 | 0.000012 |
| 100 | 0.00003 |
| 500 | 0.00067 |
| 1000 | 0.002706 |
| 2000 | 0.008759 |
| 4000 | 0.029658 |
| 8000 | 0.09584 |

Where x is the number of entries and f(x) is the runtime. Please note the runtime is only measuring the algorithm time. Other stuff such as extracting the primary keys and printing them out is not calculated. I put the data into an excel project and generated a graph. Then I used the **trendline** function to generate an equation that matches the data points as accurate as possible.

c)

|  |  |  |
| --- | --- | --- |
| Algorithm | Growth Function | Constant |
| Insertion Sort | y = 1E-10x2+ 5E-07x - 2E-05 | + 1E-10 |
| Merge Sort | y = 2E-08xln(x) + 2E-07 | + 2E-08 |
| Bubble Sort | y = 1E-09x2 + 3E-06x - 0.0005 | + 1E-09 |
| Optimized Bubble Sort | y = 1E-09x2 + 4E-07x - 0.0001 | + 1E-09 |

d)

For the bubble sort algorithm, a simple optimization can be realized. Here are the results:

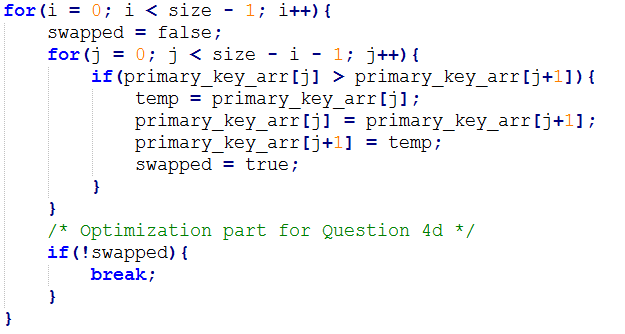
|  |  |
| --- | --- |
| x | f(x) |
| 10 | 0.000002 |
| 50 | 0.000007 |
| 100 | 0.000019 |
| 500 | 0.000391 |
| 1000 | 0.001532 |
| 2000 | 0.005651 |
| 4000 | 0.024256 |
| 8000 | 0.091932 |

e)

Divide and conquer algorithms require less complexity than nested for loop algorithms

f)

[1] <https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm>



Before this optimization, the code would have ran until variable ‘*i’* has the same value ‘*size’*. But often it doesn't need that long. Therefore, outside the inner for loop, I placed a boolean variable and had it set to false. If a swap was executed, the boolean variable is set to true. After the for loop, the boolean variable is checked, and if it's still false, no swaps were made so the array is sorted.