Singzon, Ryan
260397455

COMP 250: Introduction to Computer Science
Fall 2013

Question 5

a)Average running times for list intersections in nanoseconds

| **List Size** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** | **256** | **512** | **1000** | **2000** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |
| **Nested Loops** | 0.51 | 0.70 | 0.63 | 0.35 | 0.54 | 1.7 | 5.6 | 23.6 | 84.8 | 317.6 | 461.2 | 1663.27 |
| **Binary Search** | 1.05 | 2.14 | 3.99 | 3.10 | 7.48 | 6.41 | 14.3 | 33.21 | 66.53 | 156.51 | 330.67 | 667.99 |
| **Parallel Pointers** | 1.06 | 1.91 | 2.31 | 2.50 | 2.92 | 4.04 | 7.73 | 17.02 | 39.46 | 85.85 | 151.30 | 288.58 |
| **Merge and Sort** | 1.43 | 2.47 | 2.96 | 3.06 | 3.52 | 5.21 | 9.16 | 19.81 | 41.14 | 92.31 | 157.73 | 292.19 |

| **List Size** | **4000** | **8000** | **16000** | **32000** | **64000** | **128000** | **256000** |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| **Nested Loops** | 6504.03 | $2.57x10^4$ | $1.02x10^5$ | $4.15x10^5$ | $1.84x10^6$ | $6.65x10^6$ | $2.66x10^7$ |
| **Binary Search** | 1447.48 | 3158.58 | 6765.71 | $1.51x10^4$ | $3.27x10^4$ | $7.14x10^4$ | $1.5x10^5$ |
| **Parallel Pointers** | 606.53 | 1297.19 | 2764.36 | 6016.84 | $1.26x10^4$ | $2.67x10^4$ | $5.82x10^4$ |
| **Merge and Sort** | 627.12 | 1331.56 | 2817.46 | 6272.51 | $1.31x10^4$ | $2.70x10^4$ | $5.8x10^4$ |

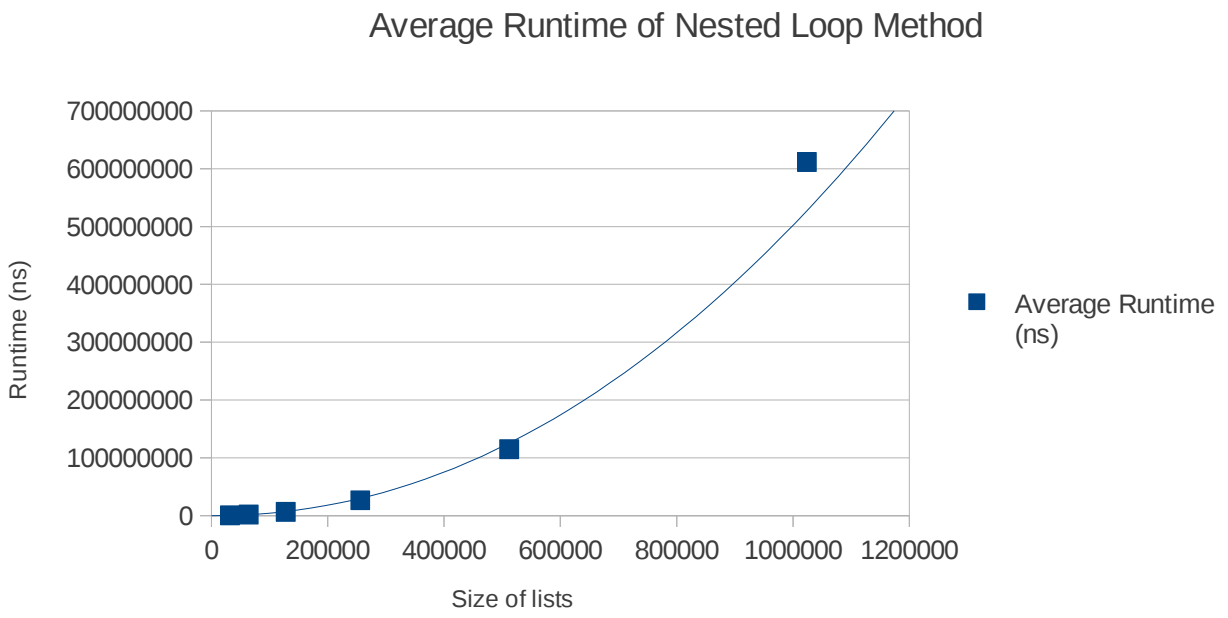
| **List Size** | **512000** | **1024000** |
|---|---|---|
| | | |
| **Nested Loops** | $1.15x10^8$ | $6.12x10^8$ |
| **Binary Search** | $3.28x10^5$ | $7.62x10^5$ |
| **Parallel Pointers** | $1.21x10^5$ | $2.5x10^5$ |
| **Merge and Sort** | $1.28x10^5$ | $2.69x10^5$ |

b) Based on my observations, the running time of of the nested loop method is logarithmic, while the other three are linear.

In order to determine the how the runtime of each method increases, the average runtime of the method was plotted against the size of the input lists. The nested loop algorithm increases as a function of the size of the list squared.

Nested loop method:
$T(n) = Cn^2$ ns

Where C is a constant which scales the runtime based on the computer processor speed




The runtimes of the other three functions could be determined simply by observing the table, since for larger list sizes, the runtime generally increased by a factor of two when the list size increased by a factor of two. Thus, the functions increase linearly.

Binary Search Method:
$T(n) = Cn$ ns

Parallel Pointers Method:
$T(n) = Cn$ ns

Merge and Sort Method:
$T(n) = Cn$ ns

c)

Nested for loop:

| Average running time for L1 > L2 | Average running time for L1 < L2 |
|---|---|
| 13644207.429 ns | 17228149.677 ns |

This method should have around the same running time regardless of which list is larger because they will both have m x n comparisons after iterating through both loops.

Binary Search:

| Average running time for L1 > L2 | Average running time for L1 < L2 |
|---|---|
| 277689.784 ns | 140065.217 ns |

When the first list is longer than the second, the algorithm takes longer because for each ID in the first list, a binary search must be completed. Every time a binary search is completed, log(n) comparisons are made, where n is the length of the second list, but this is multiplied by the length of the first list.

Parallel Pointers:

| Average running time for L1 > L2 | Average running time for L1 < L2 |
|---|---|
| 157878.964 ns | 144217.474 ns |

The time to find the intersection between the lists remains about the same because the number of comparisons is a function of the sum of the lengths of both lists, as the same operations are completed for both.

Merge and Sort:

| Average running time for L1 > L2 | Average running time for L1 < L2 |
|---|---|
| 127051.41 ns | 127279.428 ns |

The time to find the intersection remains about the same because like the parallel pointer method, the same operations are performed on both lists, and the number of comparisons is a function of the sum of the lengths of the lists.