Title: Algorithm Efficiency and Sorting

Author: Pınar Yücel

ID: 21802188

Assignment: 1

Question 1

a) Insertion Sort (According to slide number 37)

4 **|** 8 3 7 6 2 1 5 Original List

4 8 **|** 3 7 6 2 1 5 After pass 1

3 4 8 **|** 7 6 2 1 5 After pass 2

3 4 7 8 **|** 6 2 1 5 After pass 3

3 4 6 7 8 **|** 2 1 5 After pass 4

2 3 4 6 7 8 **|** 1 5 After pass 5

1 2 3 4 6 7 8 **|** 5 After pass 6

1 2 3 4 5 6 7 8 After pass 7

b) Selection Sort (According to slide number 15)

4 8 3 7 6 2 1 5 **|** Initial Array

4 5 3 7 6 2 1 **| 8** After 1st swap

4 5 3 1 6 2 **| 7 8** After 2st swap

4 5 3 1 2 **| 6 7 8** After 3st swap

4 2 3 1 **| 5 6 7 8** After 4st swap

1 2 3 **| 4 5 6 7 8** After 5st swap

1 2 **| 3 4 5 6 7 8** After 6st swap

1 | 2 3 4 5 6 7 8 After 7st swap

c) Bubble Sort (According to slide number 42)

Pass 1

4 8 3 7 6 2 1 5 **|** Initial Array

4 8 3 7 6 2 1 5

4 3 8 7 6 2 1 5

4 3 7 8 6 2 1 5

4 3 7 6 8 2 1 5

4 3 7 6 2 8 1 5

4 3 7 6 2 1 8 5

4 3 7 6 2 1 5 **| 8**

Pass 2

4 3 7 6 2 1 5 **8**

3 4 7 6 2 1 5 **8**

3 4 7 6 2 1 5 **8**

3 4 6 7 2 1 5 **8**

3 4 6 2 7 1 5 **8**

3 4 6 2 1 7 5 **8**

3 4 6 2 1 5 **| 7 8**

Pass 3

3 4 6 2 1 5 **7 8**

3 4 6 2 1 5 **7 8**

3 4 6 2 1 5 **7 8**

3 4 2 6 1 5 **7 8**

3 4 2 1 6 5 **7 8**

3 4 2 1 5 **| 6 7 8**

Pass 4

3 4 2 1 5 **6 7 8**

3 4 2 1 5 **6 7 8**

3 2 4 1 5 **6 7 8**

3 2 1 4 5 **6 7 8**

3 2 1 4 **| 5 6 7 8**

Pass 5

3 2 1 4 **5 6 7 8**

2 3 1 4 **5 6 7 8**

2 1 3 4 **5 6 7 8**

2 1 3 **| 4 5 6 7 8**

Pass 6

2 1 3 **4 5 6 7 8**

1 2 3 **4 5 6 7 8**

1 2 **| 3 4 5 6 7 8**

Pass 7

1 2 **| 3 4 5 6 7 8**

1 2 3 4 5 6 7 8

d) Merge Sort

Initial array: 4 8 3 7 6 2 1 5

mergesort(array, 0, 7)

mergesort(array, 0, 3)

mergesort(array, 0, 1)

mergesort(array, 0, 0)

mergesort(array, 1, 1)

merge(array, 0, 0, 1)

4 8 3 7 6 2 1 5

mergesort(array, 2, 3)

mergesort(array, 2, 2)

mergesort(array, 3, 3)

merge(array, 2, 2, 3)

4 8 3 7 6 2 1 5

merge(array, 0, 1, 3)

3 4 7 8 6 2 1 5

mergesort(array, 4, 7)

mergesort(array, 4, 5)

mergesort(array, 4, 4)

mergesort(array, 5, 5)

merge(array, 4, 4, 5)

3 4 7 8 2 6 1 5

mergesort(array, 6, 7)

mergesort(array, 6, 6)

mergesort(array, 7, 7)

merge(array, 6, 6, 7)

3 4 7 8 2 6 1 5

merge(array, 4, 5, 7)

3 4 7 8 1 2 5 6

merge(array, 0, 3, 7)

1 2 3 4 5 6 7 8

e) Quick Sort

Initial array: 4 8 3 7 6 2 1 5

quicksort(array, 0, 7)

partition(array, 0, 7)

1 3 2 4 6 8 7 5

quicksort(array, 0, 2)

partition(array, 0, 2)

1 3 2 4 6 8 7 5

quicksort(array, 0, -1)

quicksort(array, 1, 2)

partition(array, 1, 2)

1 2 3 4 6 8 7 5

quicksort(array, 1, 1)

quicksort(array, 3, 2)

quicksort(array, 4, 7)

partition(array, 4, 7)

1 2 3 4 5 6 7 8

quicksort(array, 4, 4)

quicksort(array, 6, 7)

partition(array, 6, 7)

1 2 3 4 5 6 7 8

quicksort(array, 6, 5)

quicksort(array, 7, 7)

Question 2

Mergesort

$T(1) = \Theta(1)$

$T(n) = 2T(n/2) + \Theta(n)$

$T(n) = 2(2T(n/4) + \Theta(n/2)) + \Theta(n)$

$T(n) = 4T(n/4) + 2\Theta(n/2) + \Theta(n)$

$T(n) = 4(2T(n/8) + \Theta(n/4)) + 2\Theta(n/2) + \Theta(n)$

$T(n) = 8T(n/8) + 4\Theta(n/4) + 2\Theta(n/2) + \Theta(n)$

.

.

.

$T(n) = 2^k T(n/2^k) + 2^{k-1} Θ(n/2^{k-1}) + 2^{k-2}Θ(n/2^{k-2}) + ... + 2Θ(n/2) + Θ(n)$

when k = logn

T(n) = nT(n/n) + (n/2)Θ(n/(n/2)) + (n/4)Θ(n/(n/4)) + ... + 2Θ(n/2) + Θ(n)

T(n) = nT(1) + Θ(n) + Θ(n) + ... + Θ(n) + Θ(n)

T(n) = nΘ(1) + kΘ(n)

plugging the value of k into the equation

T(n) = nΘ(1) + lognΘ(n)

Answer: Θ(nlogn)

Quicksort

T(1) = Θ(1)

T(n) = T(n-1) + Θ(n)

T(n) = T(n-2) + Θ(n-1) + Θ(n)

T(n) = T(n-3) + Θ(n-2) + Θ(n-1) + Θ(n)

.

.

.

T(n) = T(n-k) + Θ(n-k+1) + Θ(n-k+2) + ... + Θ(n-2) + Θ(n-1) + Θ(n)

when k = n - 1

T(n) = T(1) + Θ(2) + Θ(3) + ... + Θ(n-2) + Θ(n-1) + Θ(n)

T(n) = Θ(1) + Θ(2) + Θ(3) + ... + Θ(n-2) + Θ(n-1) + Θ(n)

T(n) = Θ(n(n+1)/2)

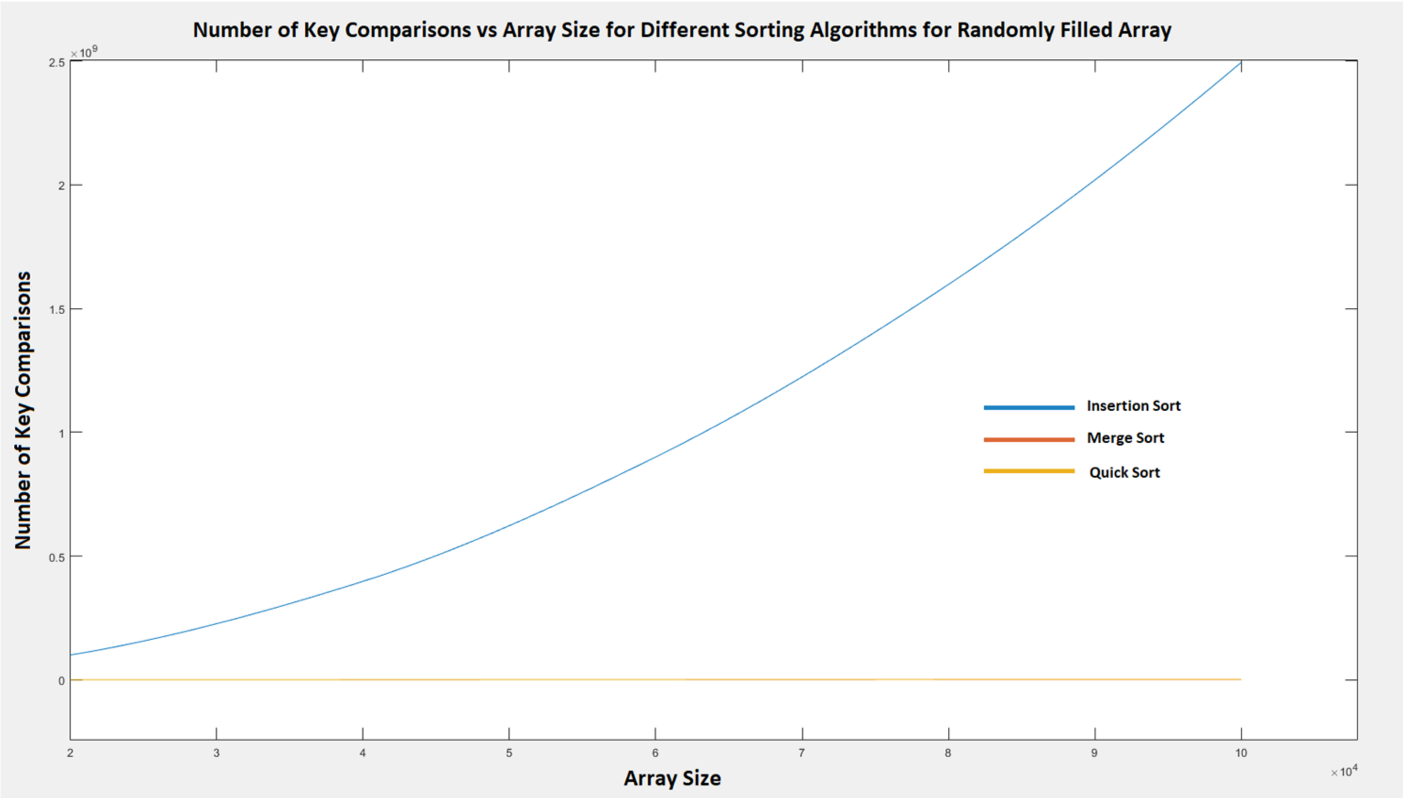
$T(n) = Θ(n^2/2) + Θ(n/2)$

Answer: $Θ(n^2)$

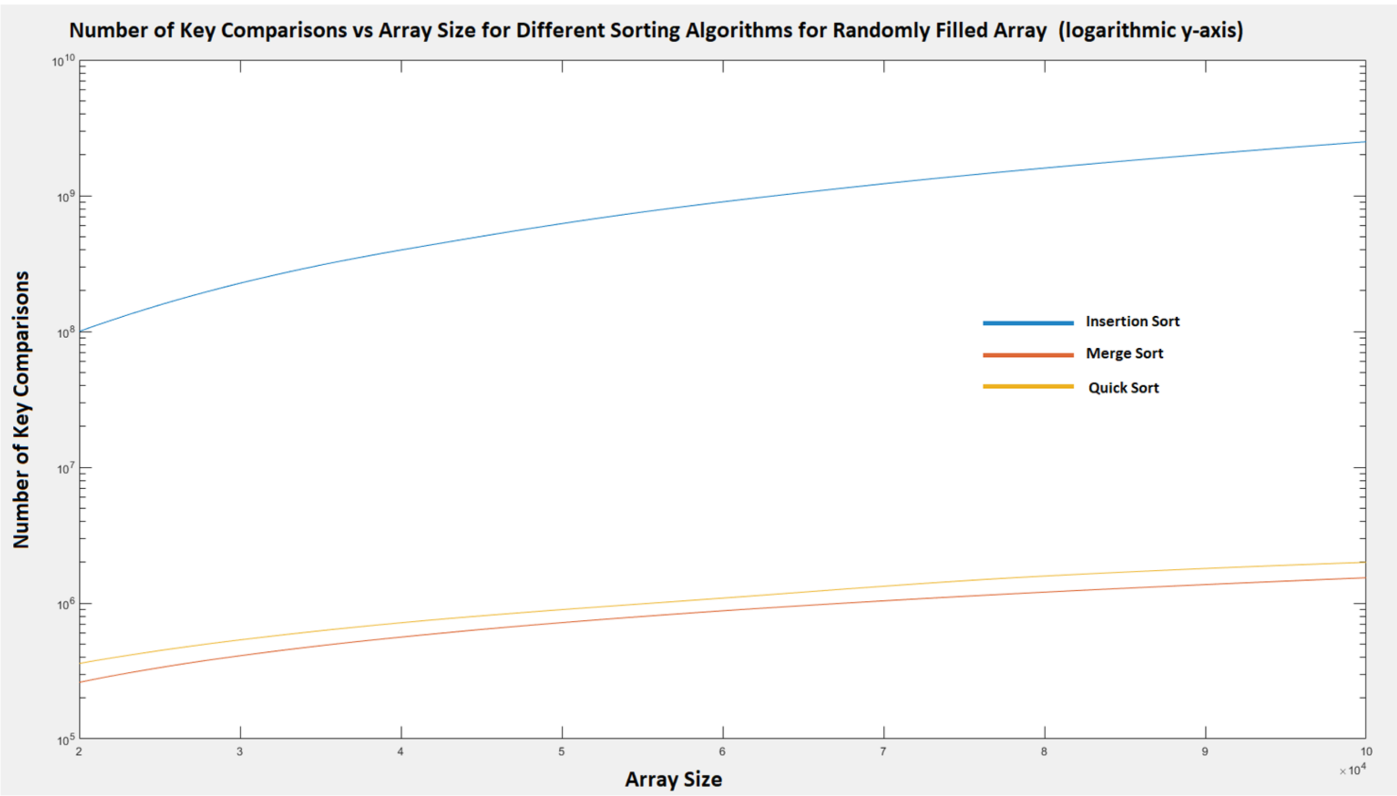
Question 3

In the following graph, merge sort and quick sort curves look as if they overlap so I also plotted the same graph with logarithmic y-axis.

Graph 1

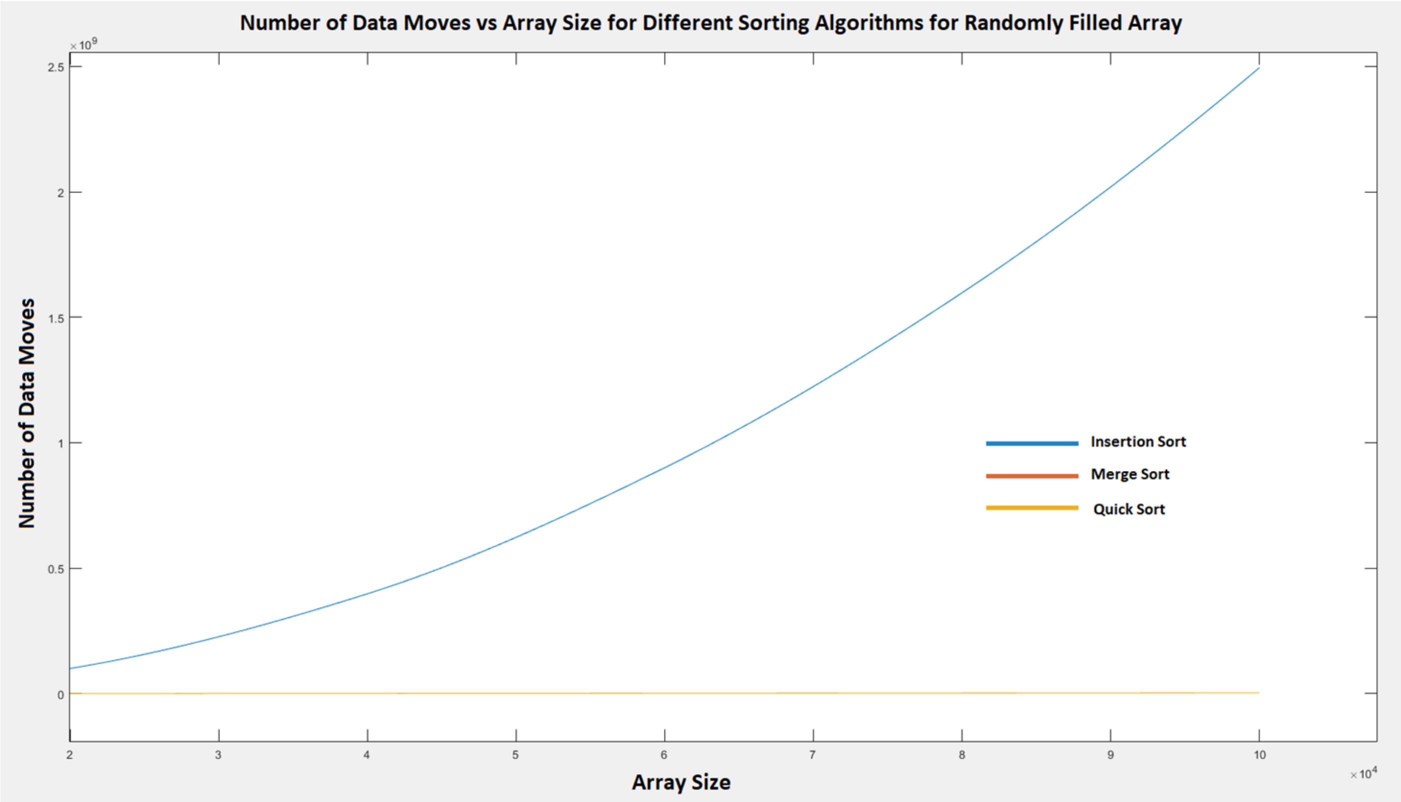


Graph 2

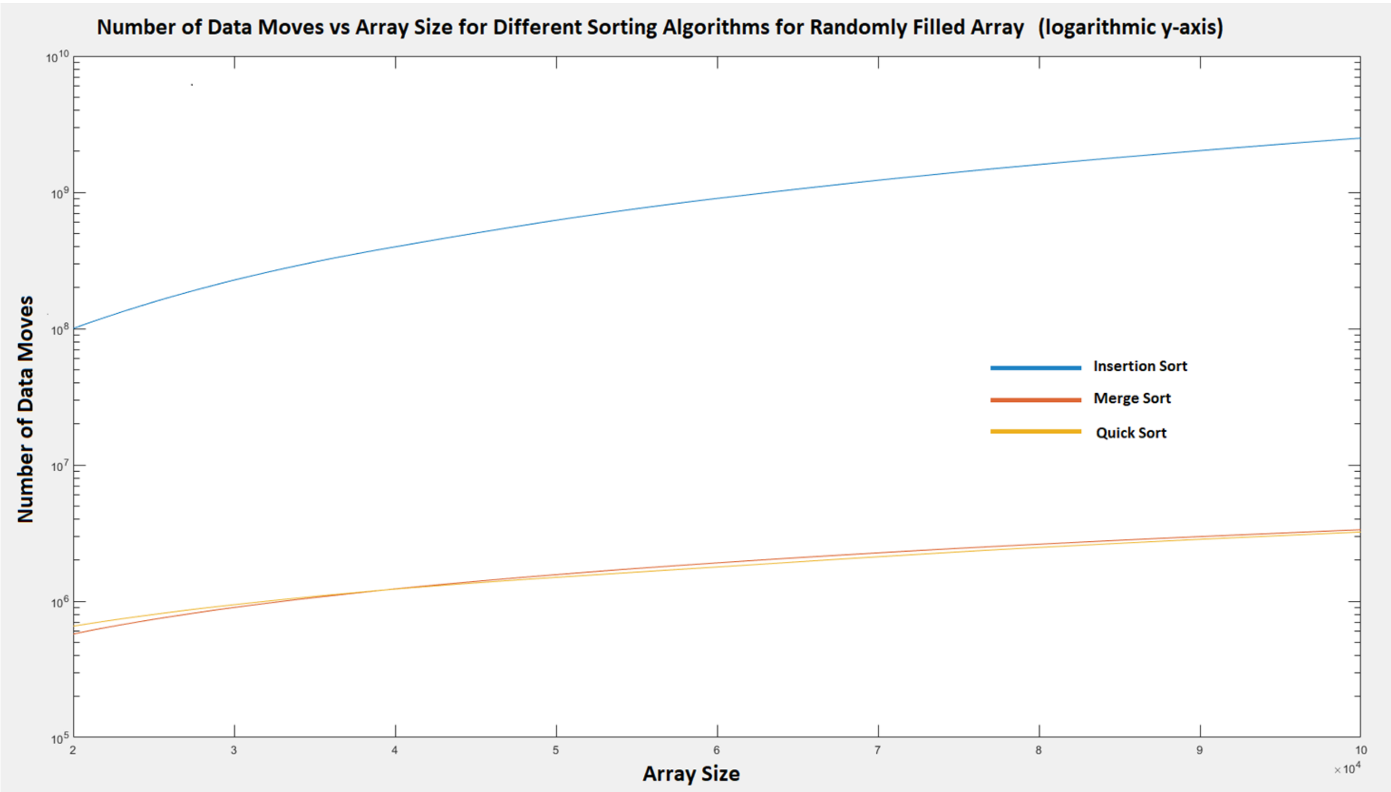


In the following graph, merge sort and quick sort curves look as if they overlap so I also plotted the same graph with logarithmic y-axis.

Graph 3

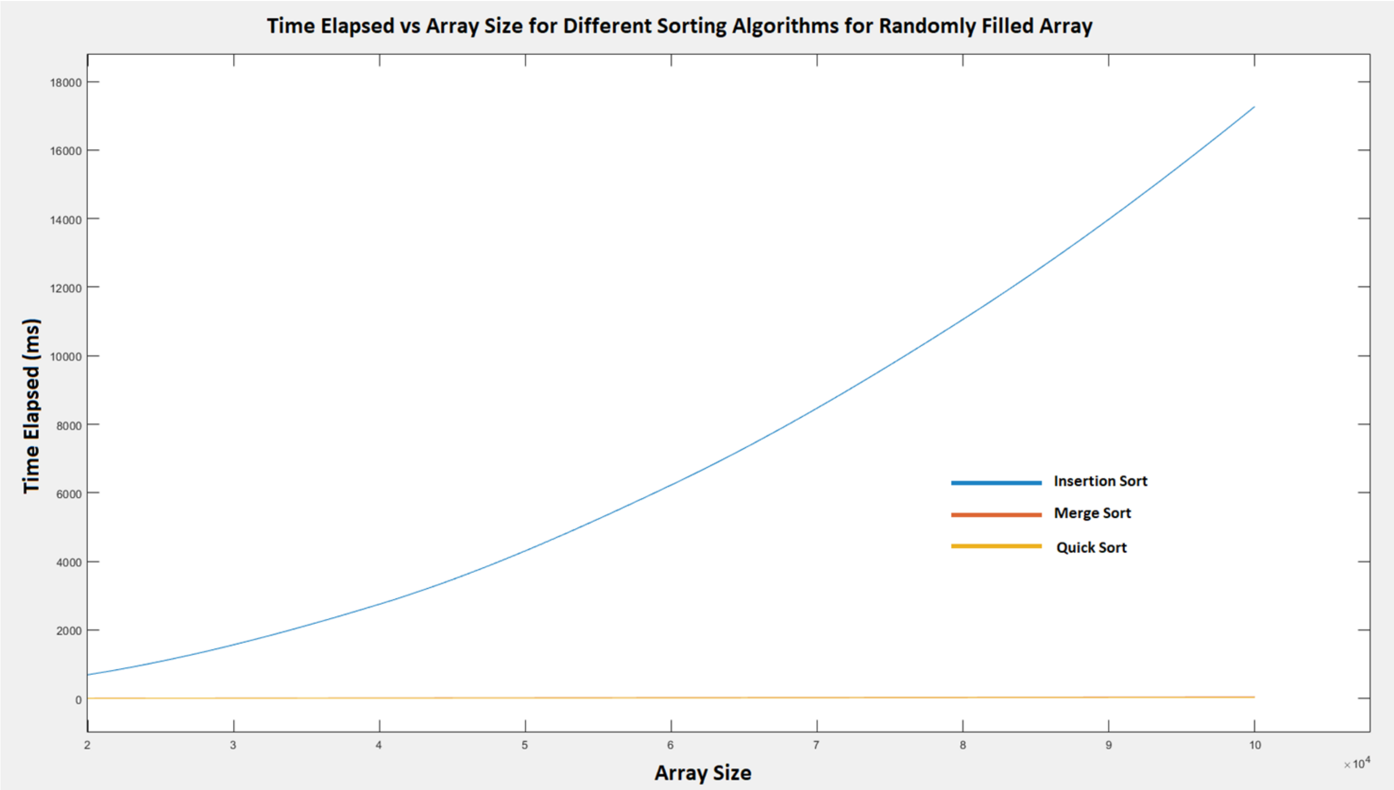


Graph 4

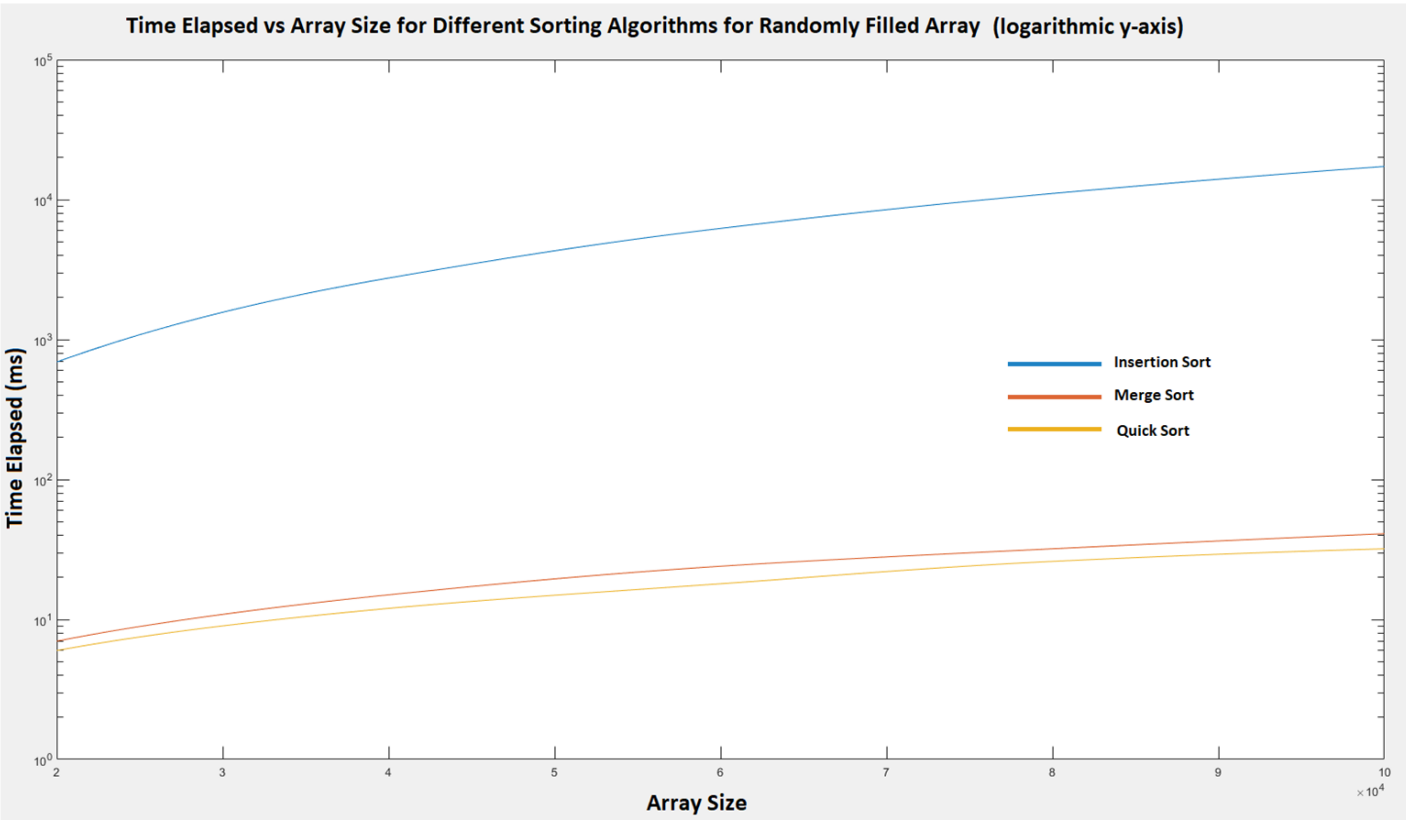


In the following graph, merge sort and quick sort curves look as if they overlap so I also plotted the same graph with logarithmic y-axis.

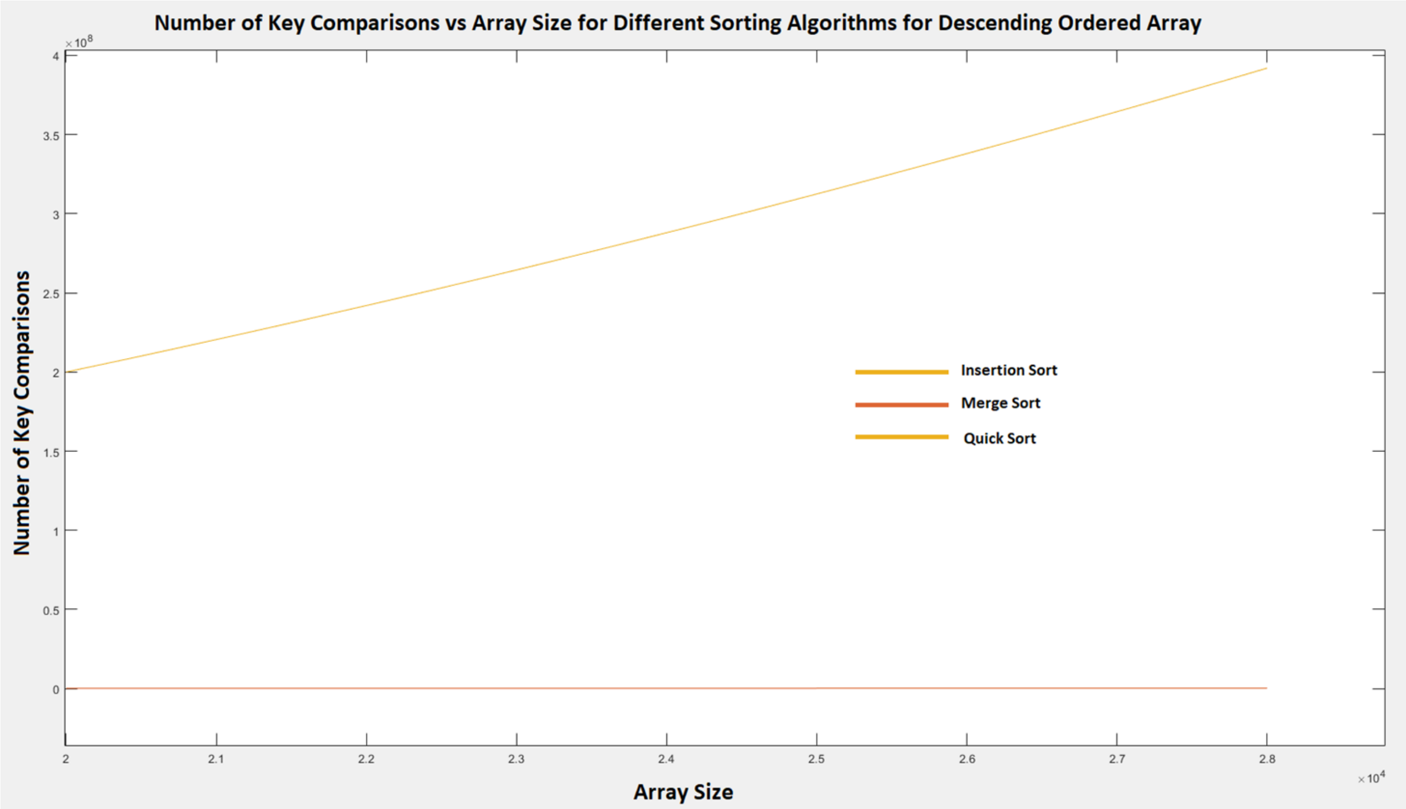
Graph 5



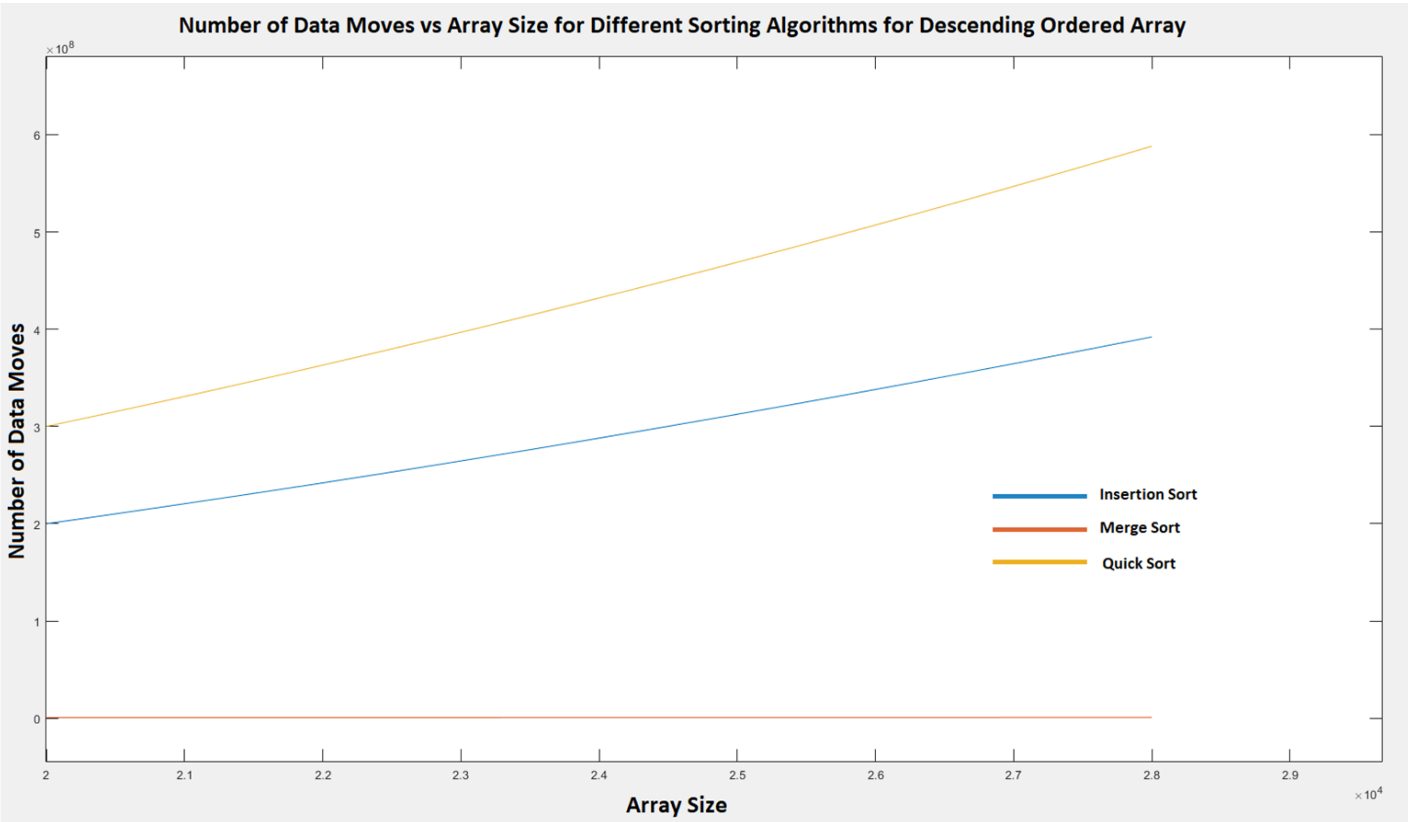
Graph 6



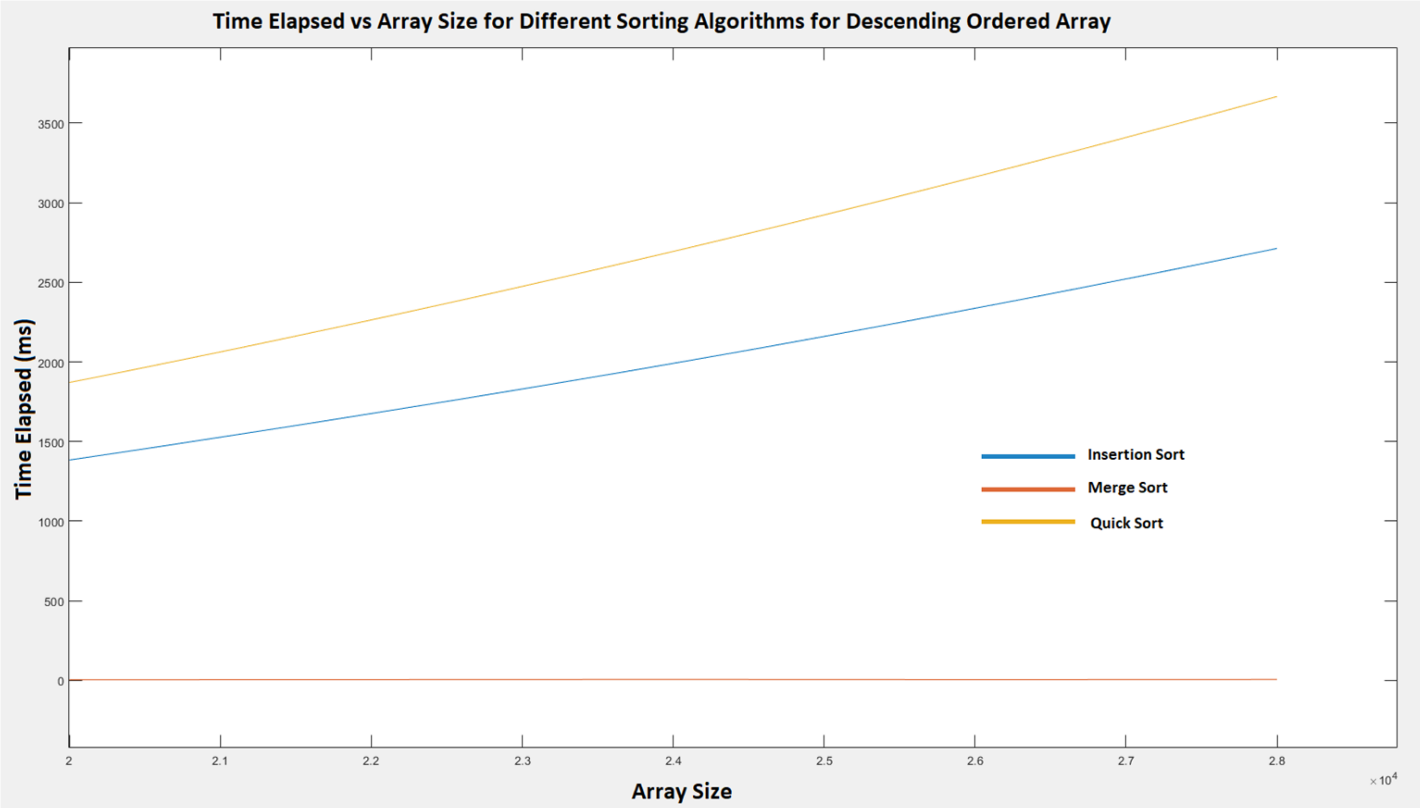
Graph 7



Graph 8

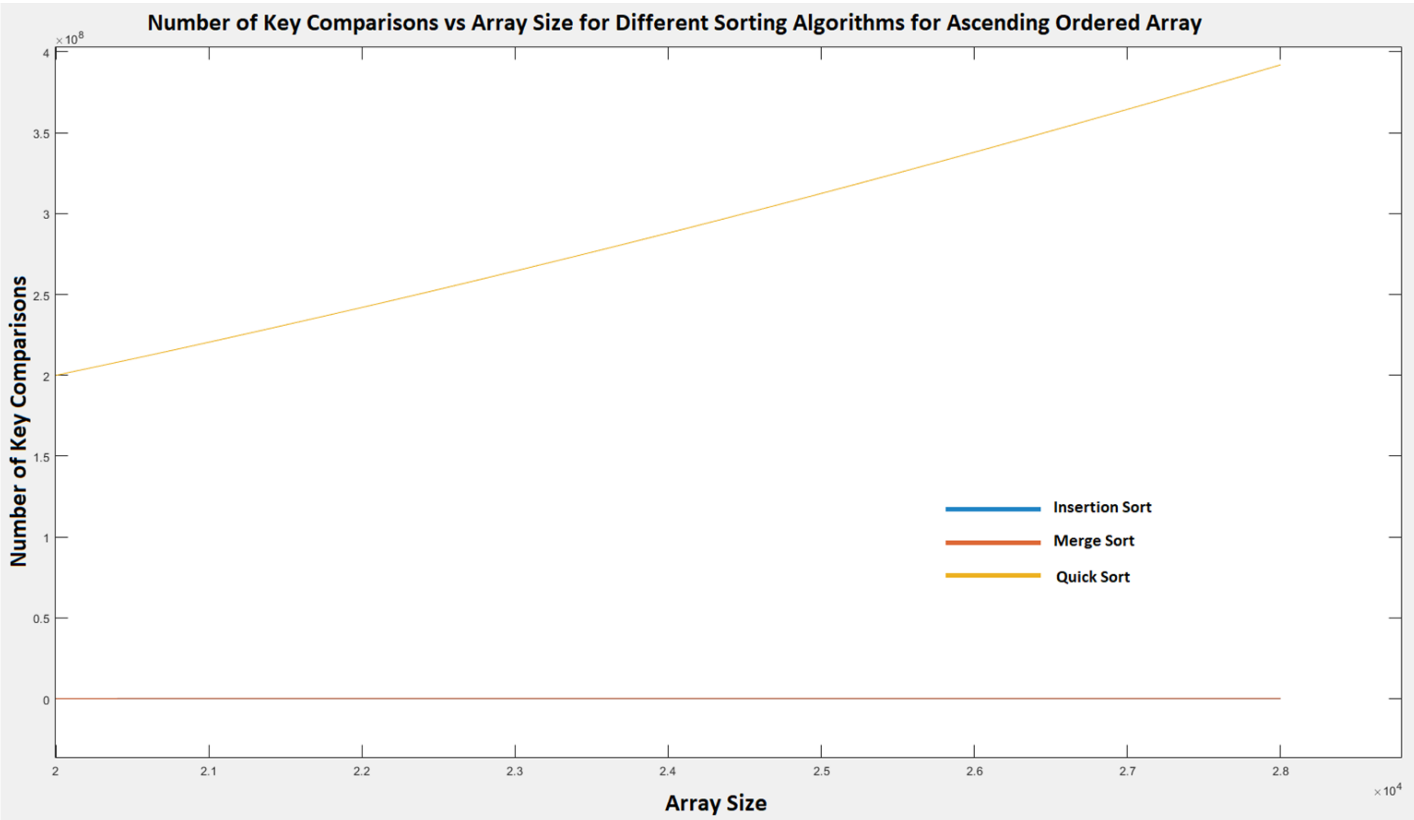


Graph 9

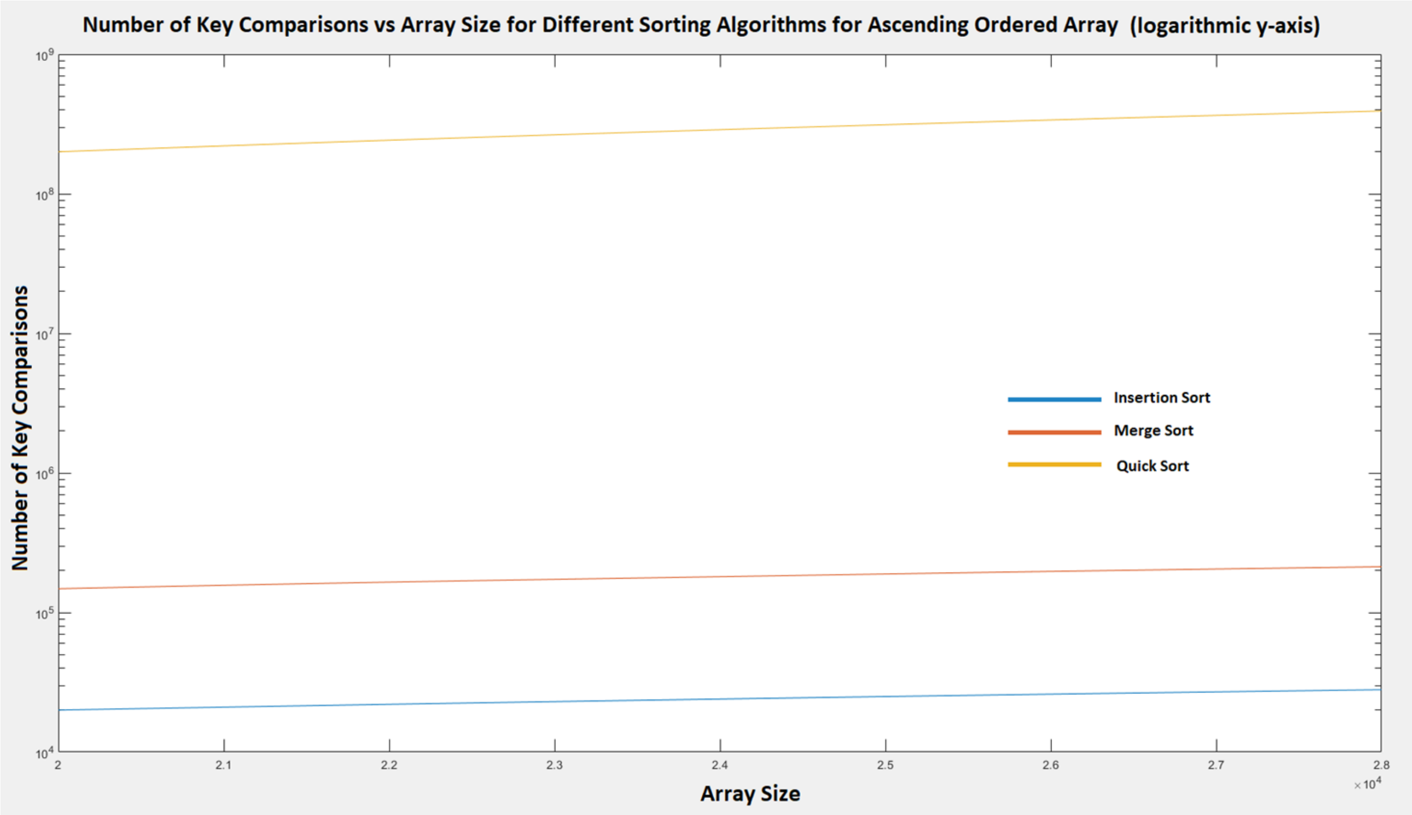


In the following graph, merge sort and insertion sort curves look as if they overlap so I also plotted the same graph with logarithmic y-axis.

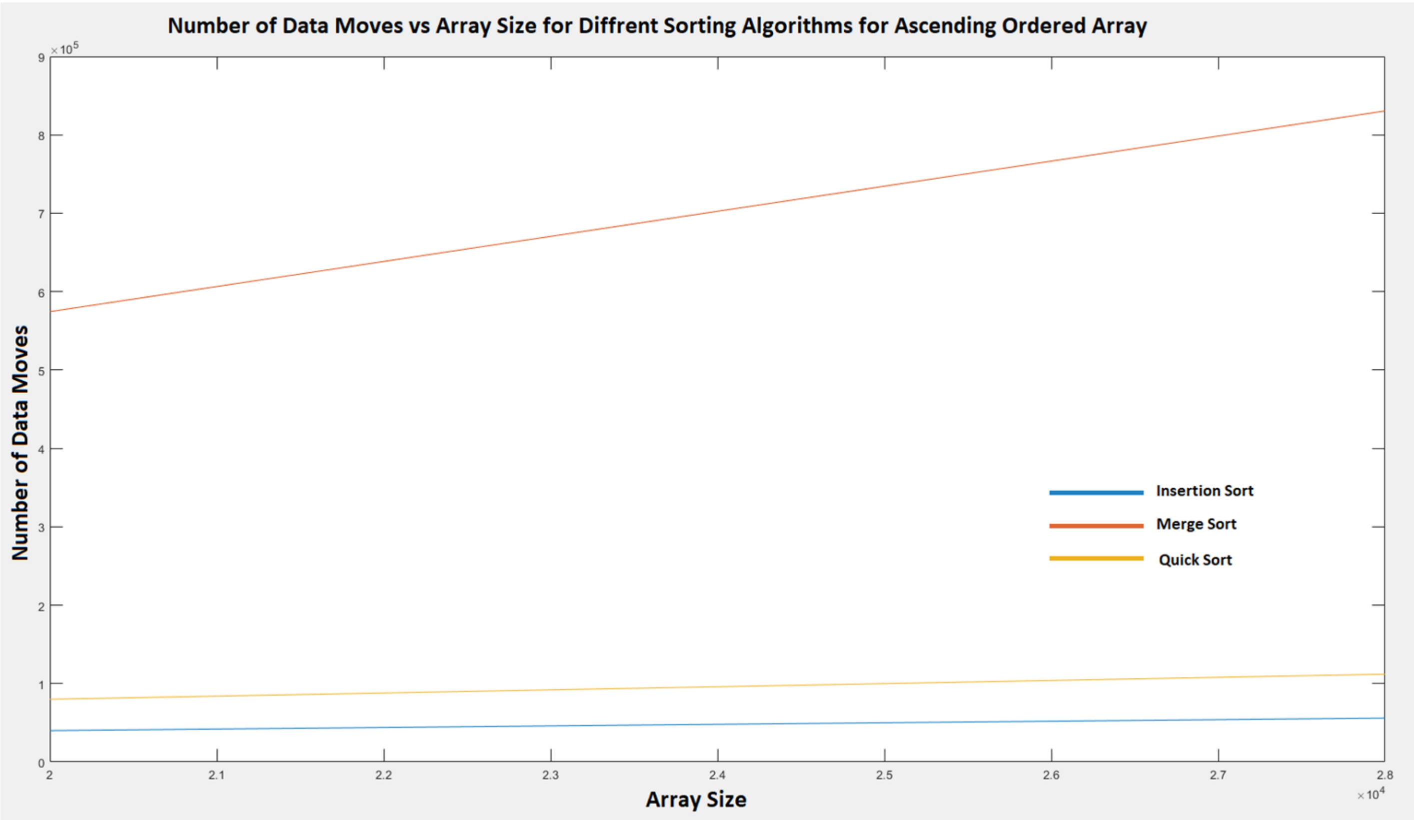
Graph 10



Graph 11

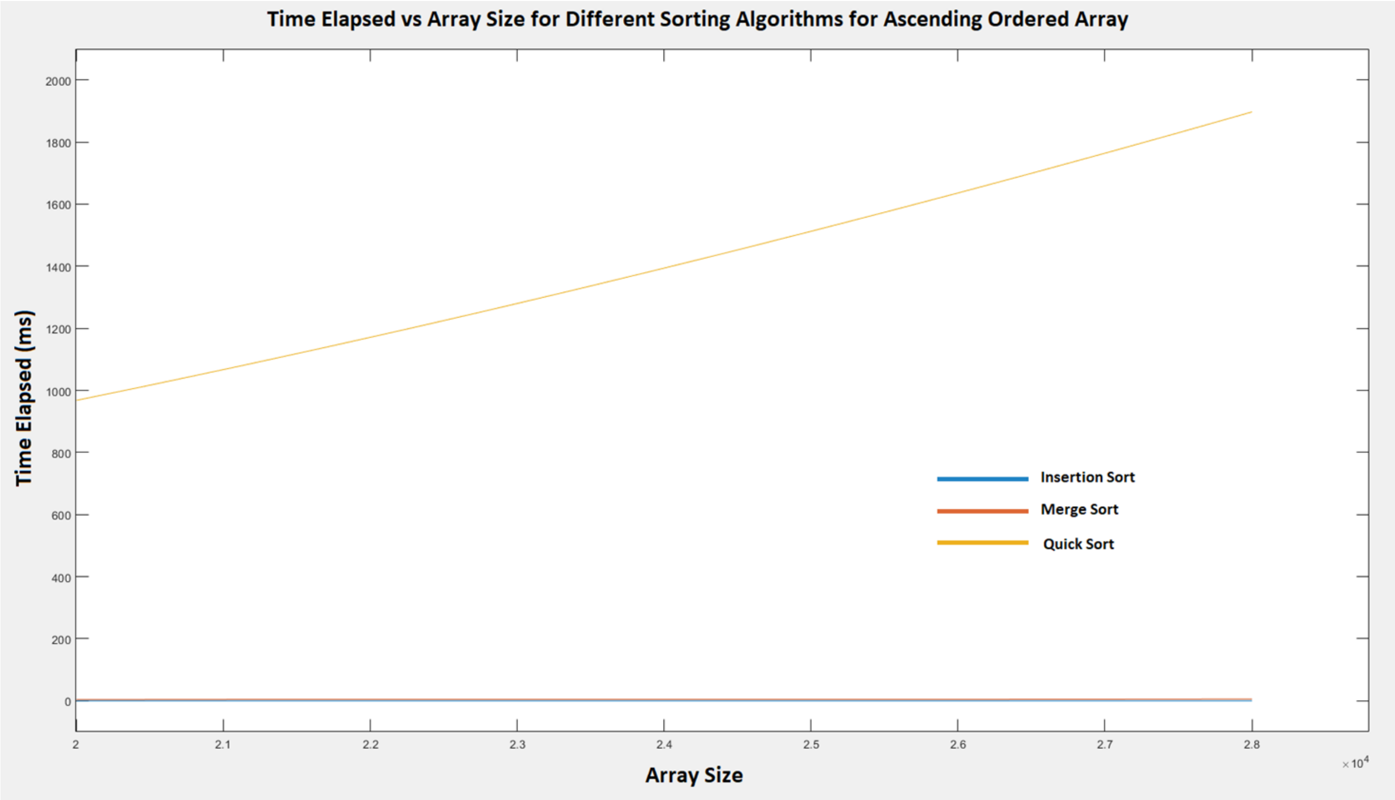


Graph 12

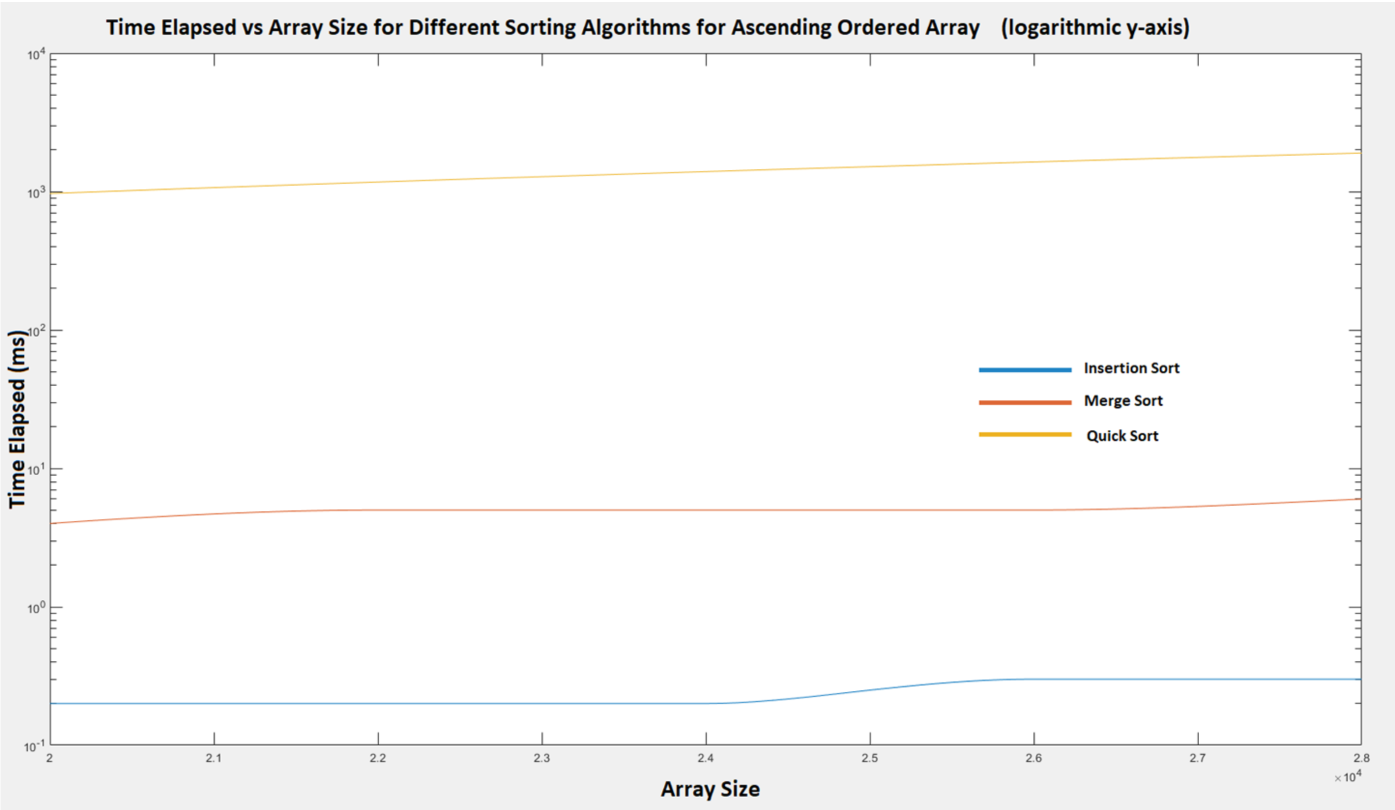


In the following graph, merge sort and insertion sort curves look as if they overlap so I also plotted the same graph with logarithmic y-axis.

Graph 13



Graph 14



Sample Output

```
----------------------------------------------------------------------------
Array Size: 20000 (Randomly filled)

Algorithm         Key Comparisons   Data Moves        Time Elapsed (ms)
Insertion Sort    100232986         100252998         692.0
Merge Sort        260952            574464            7.0
Quick Sort        359227            655561            6.0
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Array Size: 40000 (Randomly filled)

Algorithm         Key Comparisons   Data Moves        Time Elapsed (ms)
Insertion Sort    397895944         397935956         2751.0
Merge Sort        561980            1228928           15.0
Quick Sort        716397            1225041           12.0
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Array Size: 60000 (Randomly filled)

Algorithm         Key Comparisons   Data Moves        Time Elapsed (ms)
Insertion Sort    900510601         900570610         6228.0
Merge Sort        877225            1908928           24.0
Quick Sort        1088346           1780103           18.0
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Array Size: 80000 (Randomly filled)

Algorithm         Key Comparisons   Data Moves        Time Elapsed (ms)
Insertion Sort    1597421811        1597501828        11060.0
Merge Sort        1203434           2617856           32.0
Quick Sort        1580845           2477748           26.0
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Array Size: 100000 (Randomly filled)

Algorithm         Key Comparisons   Data Moves        Time Elapsed (ms)
Insertion Sort    2494526740        2494626751        17268.0
Merge Sort        1536068           3337856           41.0
Quick Sort        1997379           3215516           32.0
----------------------------------------------------------------------------
```

Array Size: 20000 (Descending order)

Algorithm	Key Comparisons	Data Moves	Time Elapsed (ms)
Insertion Sort	199990000	200029998	1384.0
Merge Sort	139216	574464	4.0
Quick Sort	199990000	300079996	1871.0

Array Size: 22000 (Descending order)

Algorithm	Key Comparisons	Data Moves	Time Elapsed (ms)
Insertion Sort	241989000	242032998	1676.0
Merge Sort	154208	638464	5.0
Quick Sort	241989000	363087996	2264.0

Array Size: 24000 (Descending order)

Algorithm	Key Comparisons	Data Moves	Time Elapsed (ms)
Insertion Sort	287988000	288035998	1991.0
Merge Sort	170624	702464	6.0
Quick Sort	287988000	432095996	2694.0

Array Size: 26000 (Descending order)

Algorithm	Key Comparisons	Data Moves	Time Elapsed (ms)
Insertion Sort	337987000	338038998	2337.0
Merge Sort	186160	766464	5.0
Quick Sort	337987000	507103996	3162.0

Array Size: 28000 (Descending order)

Algorithm	Key Comparisons	Data Moves	Time Elapsed (ms)
Insertion Sort	391986000	392041998	2713.0
Merge Sort	202512	830464	6.0
Quick Sort	391986000	588111996	3667.0

```
-----------------------------------------------------------------------------
Array Size: 20000 (Ascending order)

Algorithm          Key Comparisons    Data Moves         Time Elapsed (ms)
Insertion Sort     19999              39998              0.2
Merge Sort         148016             574464             4.0
Quick Sort         199990000          79996              968.0
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Array Size: 22000 (Ascending order)

Algorithm          Key Comparisons    Data Moves         Time Elapsed (ms)
Insertion Sort     21999              43998              0.2
Merge Sort         165024             638464             5.0
Quick Sort         241989000          87996              1171.0
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Array Size: 24000 (Ascending order)

Algorithm          Key Comparisons    Data Moves         Time Elapsed (ms)
Insertion Sort     23999              47998              0.2
Merge Sort         180608             702464             5.0
Quick Sort         287988000          95996              1394.0
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Array Size: 26000 (Ascending order)

Algorithm          Key Comparisons    Data Moves         Time Elapsed (ms)
Insertion Sort     25999              51998              0.3
Merge Sort         197072             766464             5.0
Quick Sort         337987000          103996             1636.0
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
Array Size: 28000 (Ascending order)

Algorithm          Key Comparisons    Data Moves         Time Elapsed (ms)
Insertion Sort     27999              55998              0.3
Merge Sort         212720             830464             6.0
Quick Sort         391986000          111996             1897.0
-----------------------------------------------------------------------------
```

Question 4

It can be seen from Graph 1 and Graph 2 that insertion sort requires more key comparisons than quick sort and quick sort requires more key comparisons than merge sort for randomly filled arrays.

It can be seen from Graph 3 and Graph 4 that insertion sort requires more data moves than both quick and merge sort for randomly filled arrays. Quick sort requires more data moves than merge sort up to some array size which is ~40000 . For array sizes which are greater ~40000 merge sort requires more data moves than quick sort.

Theoretical analysis suggests that while insertion sort algorithm has $O(n^2)$ time complexity, merge sort and quick sort algorithms have O(nlogn) time complexity on average case. These theoretical foundings are parallel to the foundings of my test program according to Graph 5 and Graph 6. Even though merge sort and quick sort have the same time complexity, quick sort has a better performance in practice in average case. One of the reasons for this is that quick sort is an inplace algorithm which does not require extra space like merge sort (Getting extra memory consumes time). Another reason could be that quick sort algorithm has a better cache locality since it frequently uses data that are close to each other.

It can be seen from Graph 7 that insertion sort and quick sort requires equal amount of key comparisons which is larger than the amount of key comparisons required by merge sort algorithm for arrays which are ordered in descending order.

It can be seen from Graph 8 that quick sort requires more data moves than insertion sort and insertion sort requires more data moves than merge sort for arrays which are ordered in descending order.

It can be seen from Graph 9 that quick sort requires more time than insertion sort and insertion sort requires more time than merge sort. From theoretical analysis it is known that quick sort algorithm has the worst case performance when the array is sorted in ascending or descending order (since the pivot will always be the largest or smallest). Theoretical analysis suggests that insertion sort has the time complexity of $O(n^2)$ on worst and average cases and O(n) on the best case (array already sorted). Since merge sort always runs in O(nlogn) complexity, it makes sense that insertion sort requires more time than merge sort when the array is sorted in descending order. So it could be said that theoretical analysis is parallel to experiment results.

It can be seen from Graph 10 and Graph 11 that quick sort requires more key comparisons than merge sort and merge sort requires more key comparisons than insertion sort when the array is ordered in ascending order.

It can be seen from Graph 12 that merge sort requires more data moves than quick sort and quick sort requires more data moves than insertion sort when the array is ordered in ascending order.

Graph 13 and Graph 14 suggest that quick sort requires more time than merge sort and merge sort requires more time than insertion sort when the array is ordered in ascending order. It makes sense that quick sort takes a lot more time than others since when the array is ordered in ascending order quick sort performs in its worst case. Insertion sort runs in O(n) time when the array is already sorted, it outperforms merge sort and quick sort. Since merge sort always runs in O(nlogn) time, my experimental ranking of the required times by each of the sorting algorithm is parallel to theoretical analysis.

For array size ~33000, I started getting stack overflow error for arrays ordered in ascending or descending order because of quick sort on my computer so I limited the array size for arrays ordered in ascending or descending order to 28000.

I was having integer overflow problems with the data type int for move and comparison count so I changed the data type for move and comparison count to unsigned long long.