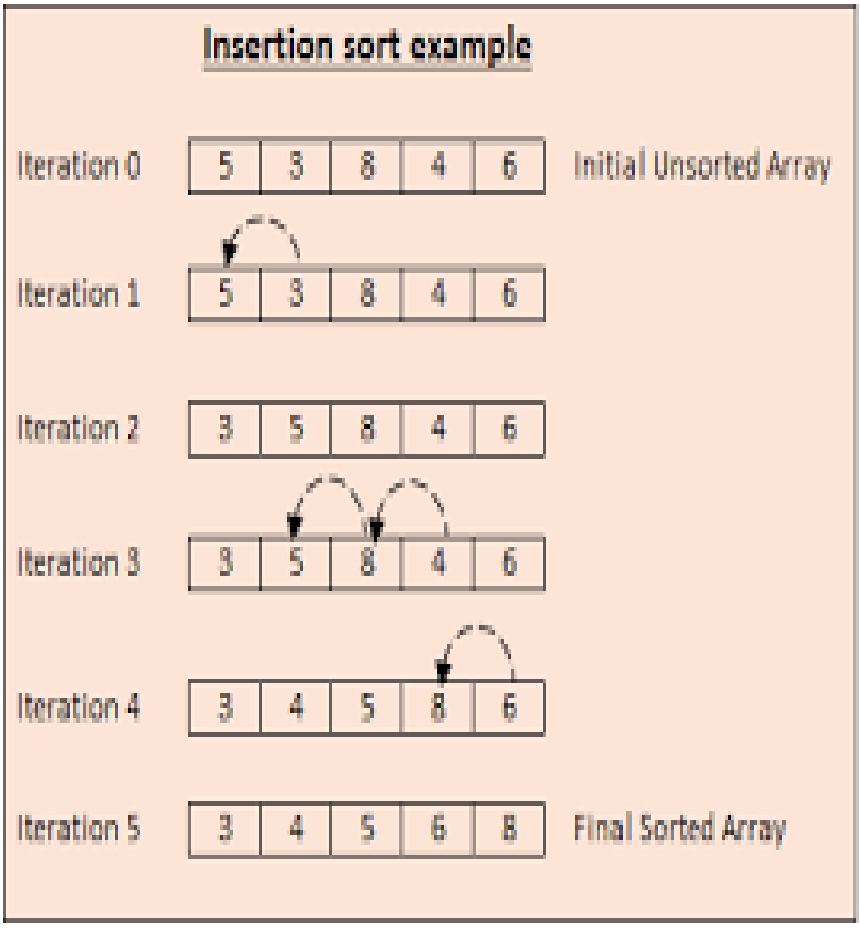
# PROJECT REPORT:

**INSERTION SORT:**

•It is a straightforward Sorting calculation which sorts the cluster by moving components one by one.
•It dependably keeps up a sorted sublist in the lower places of the rundown

•Each new thing is then "embedded" once again into the past sublist to such an extent that the sorted sublist is one thing substantial.




REF: http://www.programiz.com/sites/tutorial2program/files/Insertion-Sort-AlgorithmProgramming.jpg

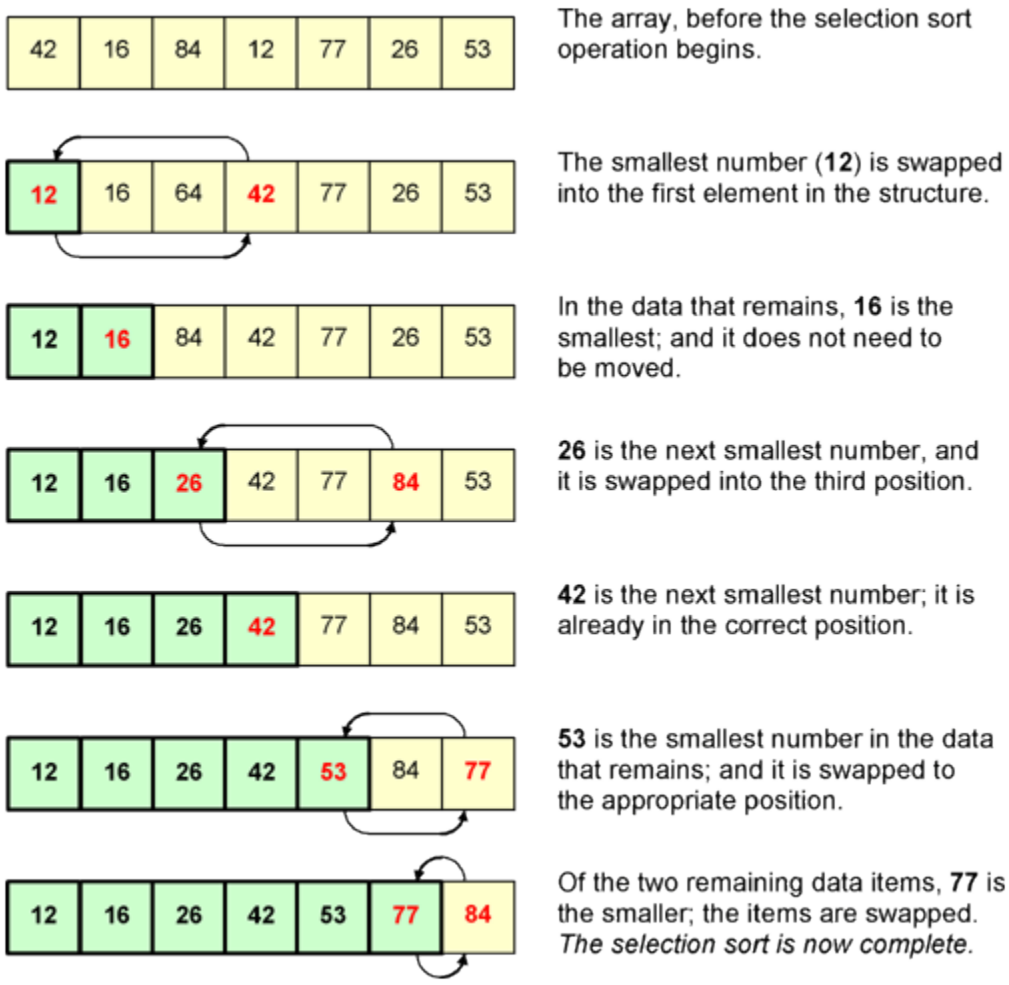
Insertion sorting is a set up sorting calculation. After n cycles in insertion sort, we would have figured out how to sort(n+1) passages. In every emphasis the main outstanding information is evacuated and embedded into its right position.

WORST CASE ANALYSIS:

The worst case and average case time complexity for insertion sort is O(n2 ).

SELECTION SORT: The possibility of determination sort is basic: we over and again locate the following biggest (or littlest) component in the exhibit and move it to its last position in the sorted cluster. Choice sort calculation begins by contrasting the initial two components of the exhibit and swapping if essential.

Give us a chance to consider that we are sorting in rising request, then choice sort analyzes the first and second component, if the primary component is more noteworthy than the second component it swaps the first and second component, else its left undisturbed. This procedure goes ahead until the end of the exhibit (i.e. Looking at the first and the last component). In the event that there are an aggregate of n components, then the above procedure is rehashed n-1 times so as to get a totally sorted cluster.

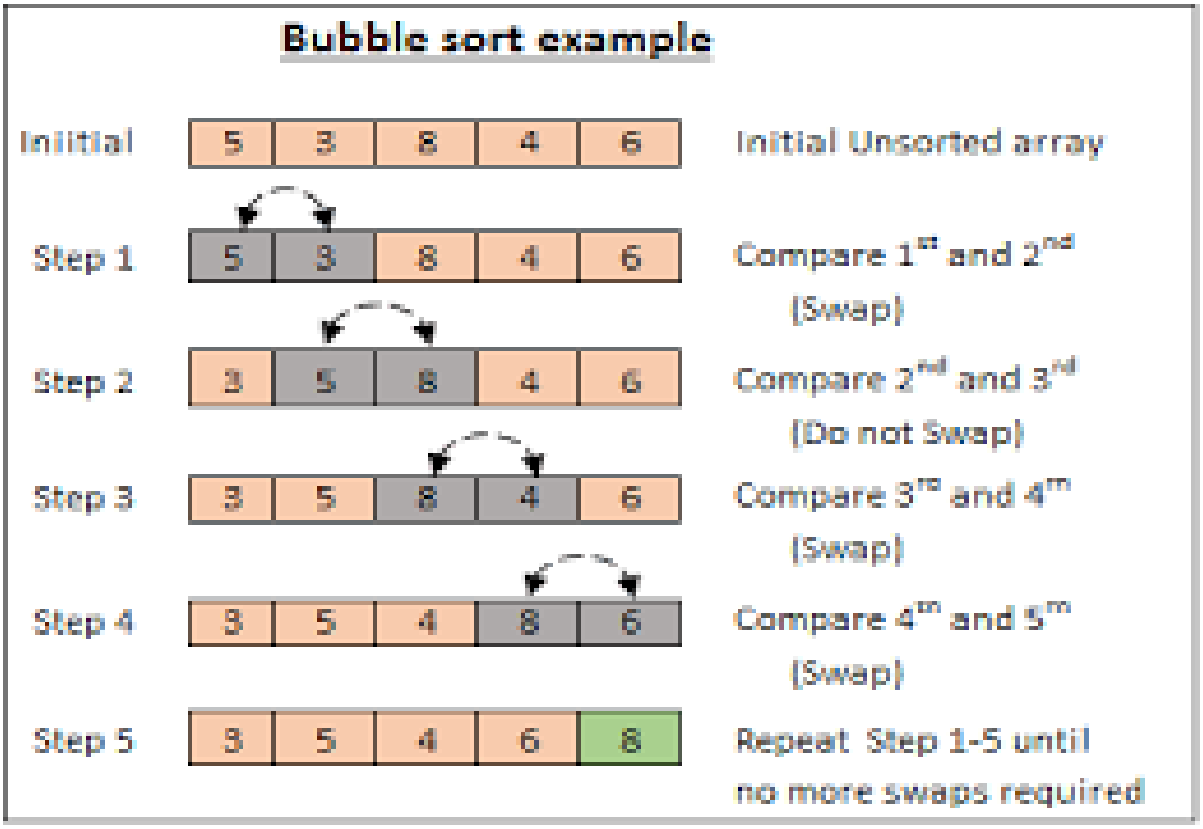


BUBBLE SORT:

•Bubble deal with calculations cycle a rundown, investigating sets of components from left to right, or start to finish.

•If the furthest left component in the match is not exactly the furthest right component, the combine will stay in a specific order.

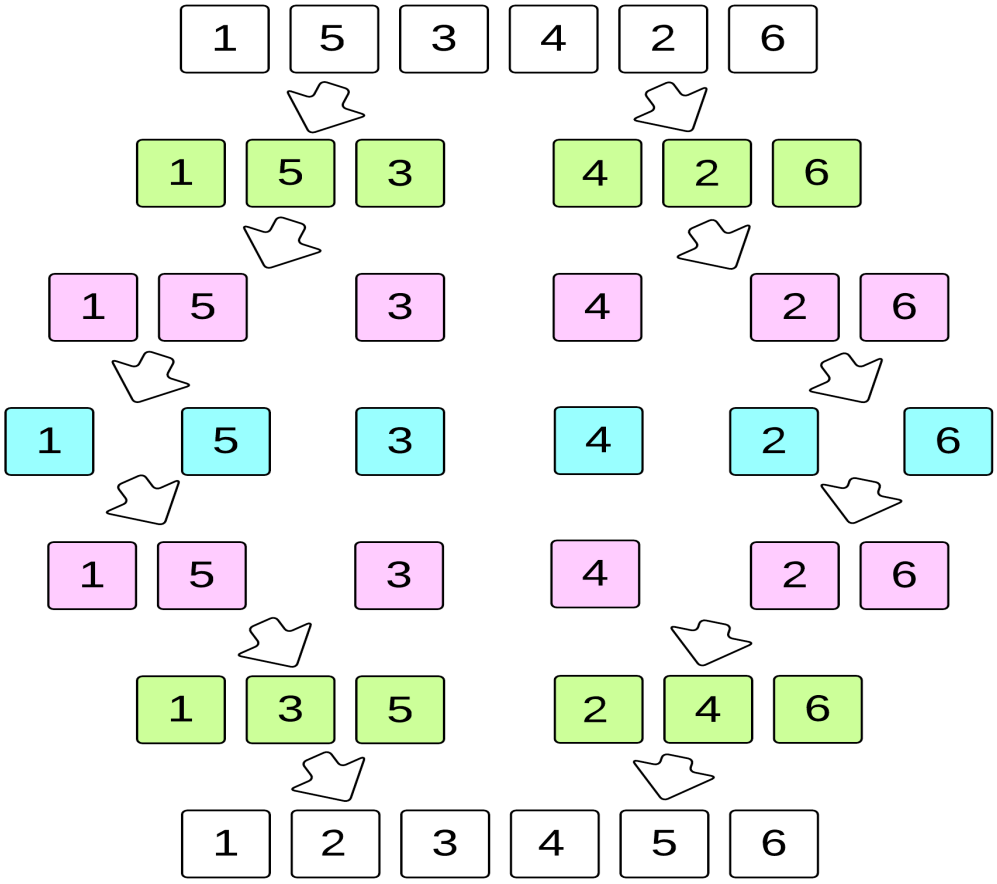
•If the furthest right component is not exactly the furthest left component, then the two components will be exchanged. This cycle rehashes from start to finish until a go in which no switch happens.




WORST CASE ANALYSIS:

The worst case and the average case complexity for bubble sort is O(n2). The space complexity for the algorithm is O(1).

MERGE SORT: Merge Sort is a Divide and Conquer calculation. It separates input cluster in two parts, calls itself for the two parts and afterward combines the two sorted parts. The Merge() capacity is utilized for combining two parts.




WORST CASE ANALYSIS:

The recurrence relation for merge sort is:

$T(n) = 2T(n/2) + O(n)$ if $n>1$

$T(n) = O(1)$ if $n=1$

Applying the master's theorem on the above recursive relation, it can be shown that the worst case complexity of merge sort is O(nlogn).

QUICK SORT: The contrast between Quick sort and Merge sort is that consolidation sort does almost no calculation before the recursive calls are made and performs work simply after the recursive calls are registered, whereas Merge sort plays out the greater part of its additional work preceding making the recursive calls.

Quicksort utilizes these means:

1. Pick any component of the exhibit to be the turn.

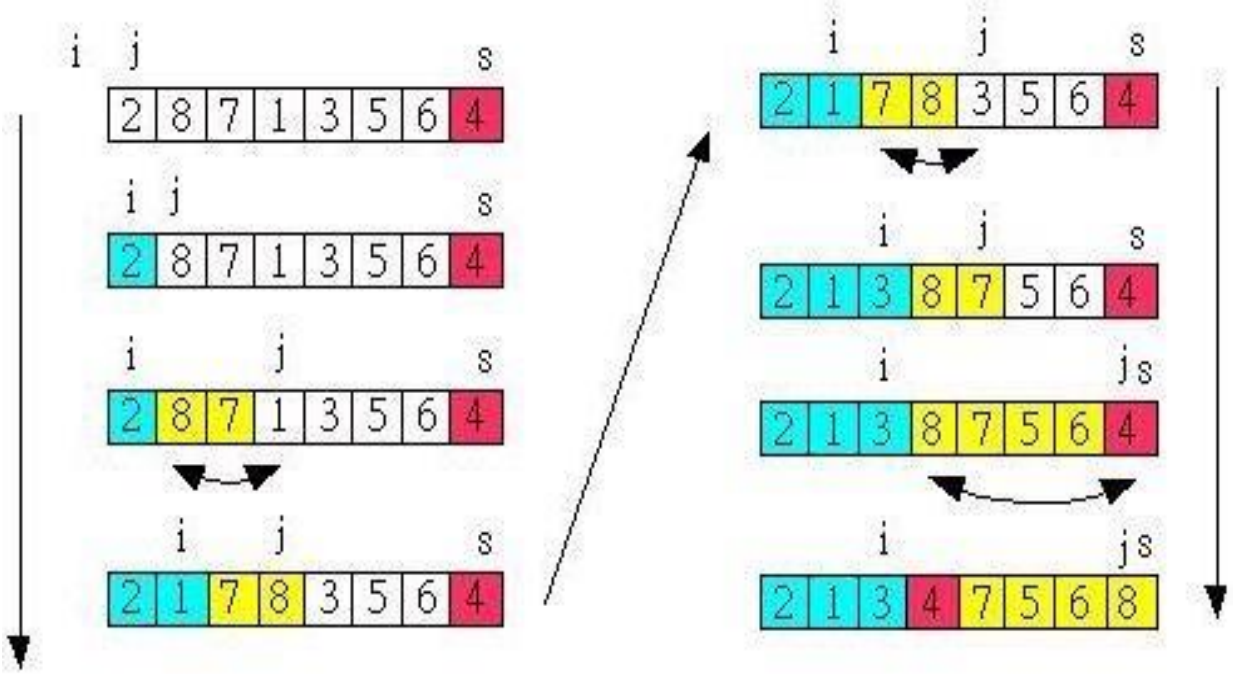
2. Isolate every single other component (aside from the turn) into two allotments.

- All components not exactly the rotate must be in the main parcel.
- All components more noteworthy than the turn must be in the second segment.

3. Utilize recursion to sort both allotments.

4. Join the initially sorted segment, the rotate, and the second sorted parcel.

The best rotate makes segments of equivalent length. The most noticeably awful rotate makes an unfilled segment. The run-time of Quicksort extents from O(n log n) with the best turns, to O(n2) with the most noticeably awful rotates, where n is the quantity of components in the cluster.



**DATA SETS**:

I have used 4 different datasets in order to analyze the time and space complexity of the above mentioned techniques, varying my input size from 1 to 1000 for analyzing time and space complexity.

Of the 4 used datasets, two datasets are Synthetic datasets and the other two datasets are real-time datasets.

REAL DATA SETS:

1.Random numbers generated with Java random functions.

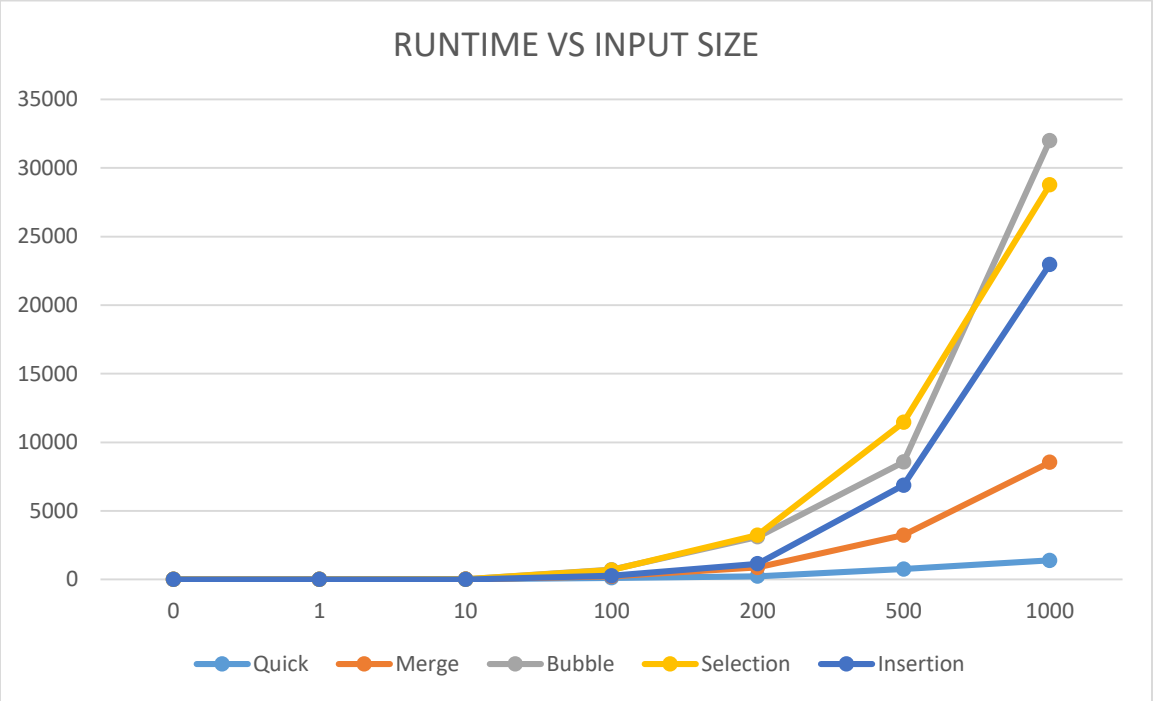
2.Generated a **Normal distribution** dataset with mean of 12000 and **Standard deviation** of 1000.

SYNTHETIC DATA SETS REFERENCE:

http://perso.telecom-paristech.fr/.eagan/class/igr204/datasets

## GRAPHS:

### RUNTIME VS ARRAY SIZE: (RANDOM NUMBERS DATASETS)




QUICK SORT:

| Input | T1 | T2 | T3 | T4 | T5 | T(avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 13 | 17 | 18 | 14 | 15.2 |
| 10 | 16 | 15 | 13 | 20 | 13 | 15.4 |
| 100 | 104 | 176 | 106 | 113 | 103 | 120.4 |
| 200 | 227 | 245 | 232 | 239 | 240 | 236.6 |
| 500 | 729 | 823 | 731 | 721 | 789 | 758.6 |
| 1000 | 1322 | 1466 | 1317 | 1448 | 1424 | 1395.4 |

MERGE SORT:

| Input | T1 | T2 | T3 | T4 | T5 | T(avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 3 | 2 | 1 | 3 | 2 |
| 10 | 26 | 32 | 24 | 31 | 29 | 28.4 |
| 100 | 206 | 253 | 196 | 244 | 267 | 233.2 |
| 200 | 861 | 842 | 881 | 893 | 920 | 879.4 |
| 500 | 3119 | 3362 | 3261 | 2924 | 3496 | 3232.4 |
| 1000 | 8372 | 8217 | 8416 | 9007 | 8721 | 8546.6 |

BUBBLE SORT:

| Input | T1 | T2 | T3 | T4 | T5 | T(avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 4 | 5 | 4 | 6 | 4.4 |
| 10 | 18 | 12 | 17 | 16 | 17 | 16 |
| 100 | 708 | 672 | 829 | 652 | 701 | 712.4 |
| 200 | 2381 | 3142 | 3619 | 2816 | 3599 | 3111.4 |
| 500 | 8659 | 8266 | 8913 | 8765 | 8216 | 8563.8 |
| 1000 | 34100 | 28945 | 33633 | 31153 | 32113 | 31988.8 |

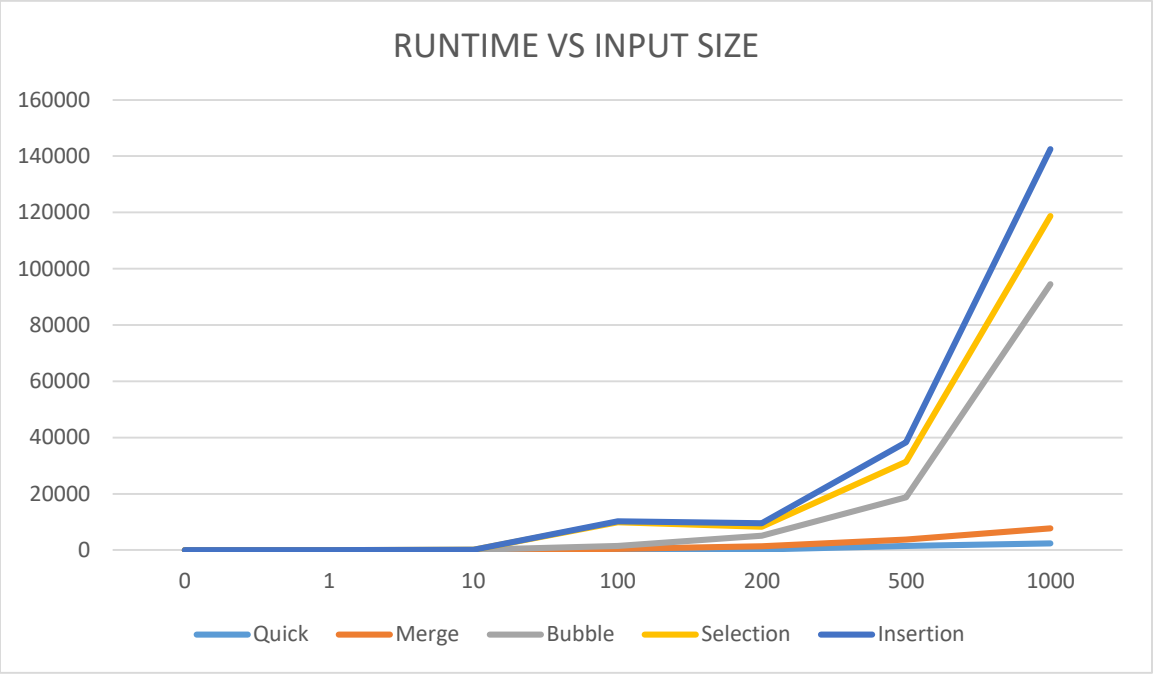
SELECTION SORT:

| Input | T1 | T2 | T3 | T4 | T5 | T(avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 2 | 3 | 2 | 3 | 2.4 |
| 10 | 14 | 17 | 17 | 15 | 14 | 15.4 |
| 100 | 678 | 691 | 742 | 572 | 769 | 690.4 |
| 200 | 3129 | 3254 | 3310 | 2879 | 3589 | 3232.3 |
| 500 | 11129 | 10986 | 11679 | 12789 | 10673 | 11451.2 |
| 1000 | 28674 | 26842 | 28427 | 28780 | 30863 | 28761.4 |

INSERTION SORT:

| Input | T1 | T2 | T3 | T4 | T5 | T(avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 2 | 2 | 1 | 1.6 |
| 10 | 8 | 7 | 11 | 7 | 9 | 8.4 |
| 100 | 262 | 247 | 295 | 287 | 303 | 278.8 |
| 200 | 1043 | 1361 | 985 | 1193 | 1172 | 1150.8 |
| 500 | 6723 | 7709 | 7021 | 6727 | 6192 | 6874.4 |
| 1000 | 23699 | 22943 | 23274 | 21245 | 23679 | 22968 |

**RUNTIME VS ARRAY SIZE: (NORMAL DISTRIBUTION)**




**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 3 | 2 | 2 | 2.4 |
| 10 | 10 | 7 | 8 | 11 | 8 | 8.8 |
| 100 | 346 | 305 | 366 | 354 | 351 | 344.4 |
| 200 | 1262 | 1055 | 1501 | 1395 | 1044 | 1251.4 |
| 500 | 6119 | 6982 | 7064 | 7695 | 6986 | 6969.2 |
| 1000 | 22815 | 22474 | 20696 | 23675 | 29098 | 23751.6 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 2 | 1 | 1.8 |
| 10 | 13 | 11 | 10 | 12 | 14 | 12 |
| 100 | 816 | 832 | 805 | 876 | 849 | 835.6 |
| 200 | 3125 | 3096 | 3167 | 3255 | 3297 | 3188 |
| 500 | 12745 | 12626 | 12071 | 12918 | 12613 | 12594.6 |
| 1000 | 25552 | 23318 | 22962 | 25910 | 23361 | 24220.6 |

**BUBBLE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 2 | 3 | 4 | 3 | 3 |
| 10 | 12 | 15 | 14 | 12 | 16 | 13.8 |
| 100 | 976 | 1005 | 1097 | 1018 | 1047 | 1028.6 |
| 200 | 3731 | 3946 | 3826 | 3731 | 3598 | 3766.4 |
| 500 | 15305 | 14625 | 14749 | 15316 | 14894 | 14977.8 |
| 1000 | 87146 | 87693 | 81497 | 88611 | 88789 | 86750.8 |

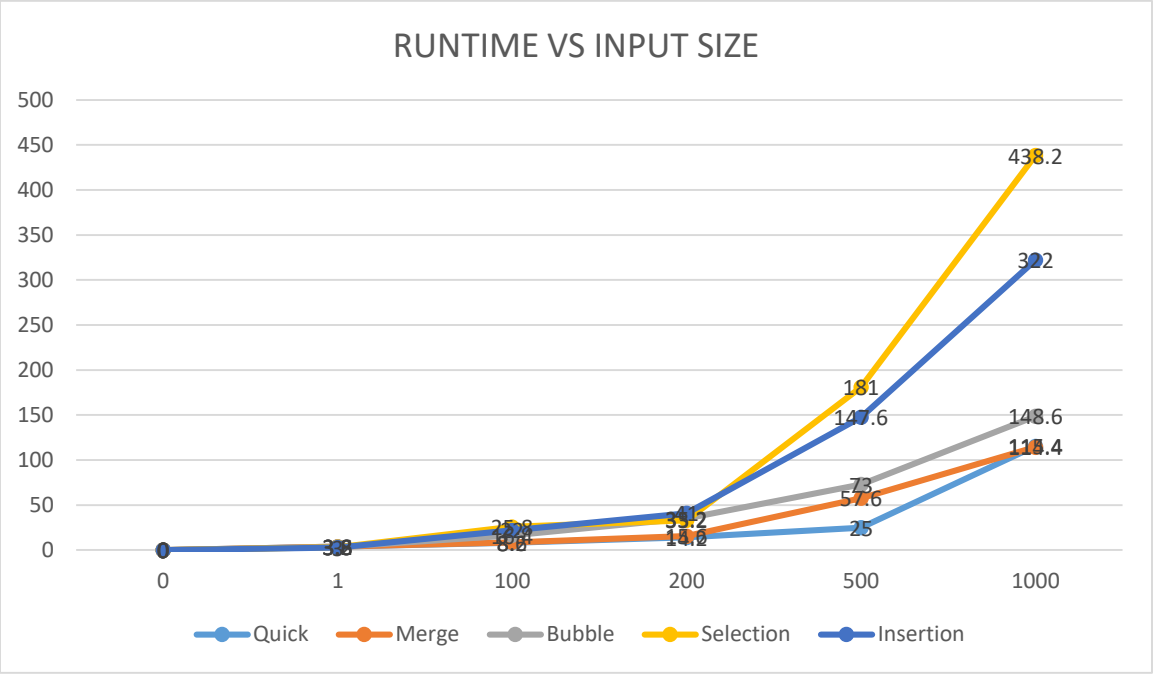
**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 3 | 3 | 4 | 3 | 3.4 |
| 10 | 23 | 31 | 22 | 23 | 32 | 26.2 |
| 100 | 321 | 345 | 319 | 327 | 344 | 331.2 |
| 200 | 1179 | 1045 | 1292 | 1186 | 1177 | 1175.8 |
| 500 | 2365 | 2492 | 2292 | 2254 | 2292 | 2339 |
| 1000 | 5286 | 5164 | 5244 | 5286 | 5760 | 5348 |

**QUICK SORT**:

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 12 | 16 | 17 | 19 | 16.6 |
| 10 | 20 | 19 | 17 | 20 | 19 | 15.4 |
| 100 | 123 | 156 | 167 | 179 | 63 | 157.6 |
| 200 | 224 | 227 | 221 | 223 | 225 | 224 |
| 500 | 1367 | 1467 | 1492 | 1517 | 1478 | 1464.2 |
| 1000 | 2459 | 2366 | 2519 | 2319 | 2502 | 2432.4 |

**RUNTIME VS ARRAY SIZE: (CAMERA-DATA SETS)**




**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 2 | 3 | 2 | 5 | 3 |
| 100 | 21 | 23 | 22 | 21 | 23 | 22 |
| 200 | 43 | 38 | 41 | 40 | 43 | 41 |
| 500 | 149 | 142 | 147 | 144 | 156 | 147.6 |
| 1000 | 324 | 339 | 321 | 298 | 332 | 322.8 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 3 | 4 | 3 | 3 | 3.2 |
| 100 | 24 | 19 | 25 | 33 | 28 | 25.8 |
| 200 | 29 | 42 | 31 | 35 | 29 | 33.2 |
| 500 | 177 | 169 | 167 | 176 | 167 | 181 |
| 1000 | 462 | 456 | 424 | 438 | 411 | 438.2 |

**BUBBLE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 4 | 3 | 5 | 3 | 3.6 |
| 100 | 16 | 15 | 18 | 16 | 17 | 16.4 |
| 200 | 32 | 34 | 33 | 39 | 38 | 35.2 |
| 500 | 70 | 69 | 72 | 82 | 72 | 73 |
| 1000 | 141 | 147 | 146 | 159 | 150 | 148.6 |

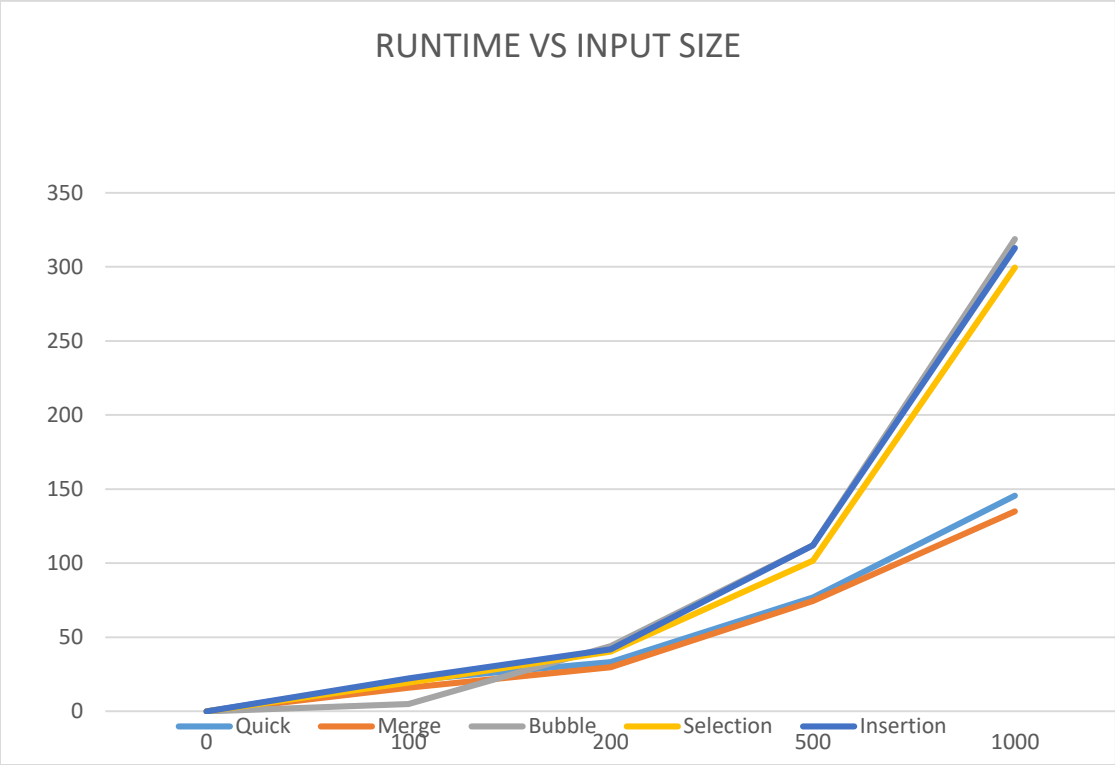
**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 4 | 5 | 3 | 4 | 3.8 |
| 100 | 8 | 9 | 11 | 7 | 8 | 8.6 |
| 200 | 16 | 11 | 18 | 17 | 16 | 15.6 |
| 500 | 61 | 59 | 62 | 54 | 52 | 57.6 |
| 1000 | 112 | 119 | 119 | 102 | 121 | 114.4 |

**QUICK SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 5 | 4 | 3 | 4 | 3.8 |
| 100 | 8 | 11 | 9 | 6 | 7 | 8.2 |
| 200 | 15 | 16 | 11 | 18 | 11 | 14.2 |
| 500 | 24 | 28 | 25 | 26 | 22 | 25 |
| 1000 | 112 | 119 | 123 | 102 | 121 | 115.4 |

**RUTIME VS ARRAY SIZE: (PRICE DATA-SET)**




**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 21 | 23 | 22 | 24 | 21 | 22.2 |
| 200 | 40 | 45 | 44 | 42 | 38 | 41.2 |
| 500 | 98 | 117 | 114 | 115 | 116 | 112 |
| 1000 | 310 | 322 | 312 | 296 | 324 | 312.8 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 22 | 16 | 25 | 18 | 16 | 19.4 |
| 200 | 32 | 46 | 39 | 48 | 36 | 40.2 |
| 500 | 98 | 110 | 102 | 103 | 95 | 101.6 |
| 1000 | 303 | 318 | 293 | 285 | 299 | 299.6 |

**BUBBLE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 4 | 6 | 3 | 4 | 7 | 4.8 |
| 200 | 42 | 46 | 49 | 40 | 43 | 44 |
| 500 | 123 | 109 | 108 | 114 | 106 | 112 |
| 1000 | 326 | 319 | 325 | 312 | 302 | 318.8 |

**MERGE SORT:**

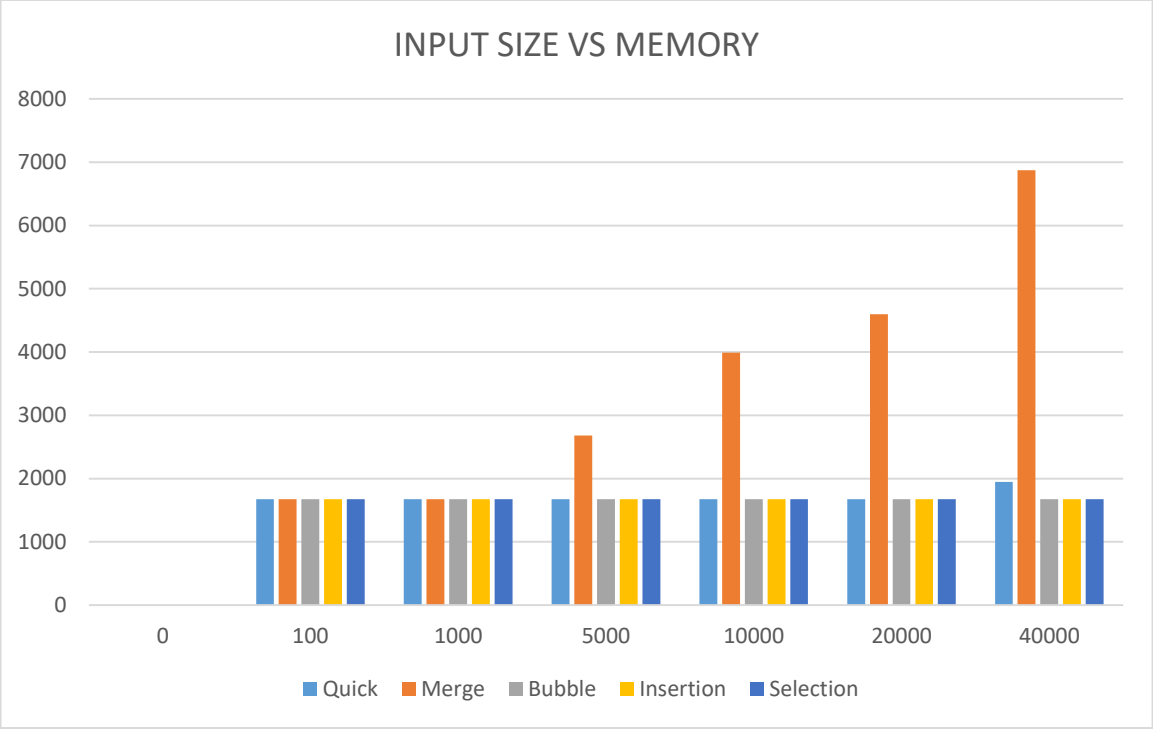
| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 16 | 15 | 17 | 17 | 14 | 15.8 |
| 200 | 29 | 30 | 28 | 31 | 30 | 29.6 |
| 500 | 68 | 72 | 86 | 79 | 67 | 74.4 |
| 1000 | 147 | 135 | 132 | 131 | 130 | 135 |

**QUICK SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 19 | 25 | 17 | 21 | 22 | 20.8 |
| 200 | 32 | 31 | 36 | 35 | 32 | 33.2 |
| 500 | 82 | 78 | 81 | 72 | 71 | 76.8 |
| 1000 | 151 | 146 | 147 | 142 | 141 | 145.4 |

From graphs we conclude that:

It is clear from the bends portrayed over that the three Sorting calculation to be specific Insertion sort, bubble sort, Selection sort have a bend that is expanding at a quadratic rate. Thus the most pessimistic scenario many-sided quality of these three calculations are O(n2 ). The Merge sort is sub-direct with the pattern of logn. Quick sort is again sub-direct with the pattern of logn. The most pessimistic scenario of quicksort is O(n2 ) which is the same as that of Insertion, bubble , selection sorting calculations. Thus from the above graphs we can conclude that quick sort and merge sort perform better than Insertion, bubble, selection sorting algorithms for large input sizes.




**INPUT SIZE VS MEMORY USAGE: (RANDOM DATA SETS)**

**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**BUBBLE SORT:**

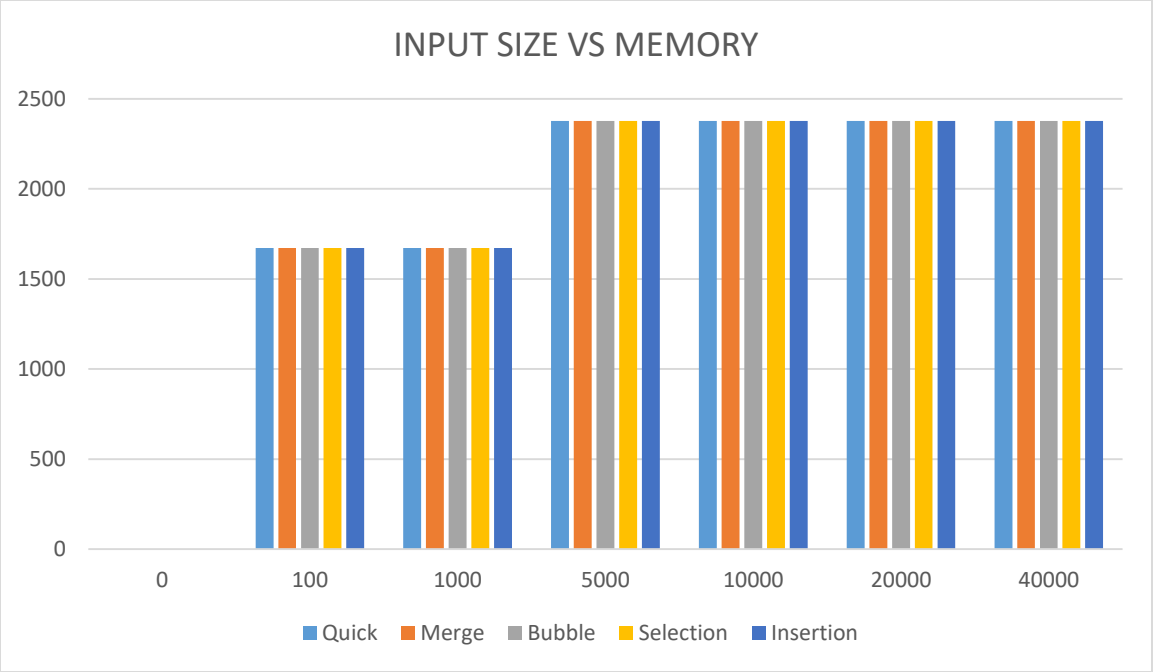
| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 2678 | 2678 | 2678 | 2678 | 2678 | 2678 |
| 10000 | 3989 | 3989 | 3989 | 3989 | 3989 | 3989 |
| 20000 | 4598 | 4598 | 4598 | 4598 | 4598 | 4598 |
| 40000 | 6827 | 6827 | 6827 | 6827 | 6827 | 6827 |

**QUICK SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1948 | 1948 | 1948 | 1948 | 1948 | 1948 |




**INPUT SIZE VS MEMORY USAGE: (NORMAL DISTRIBUTION)**

**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**BUBBLE SORT:**

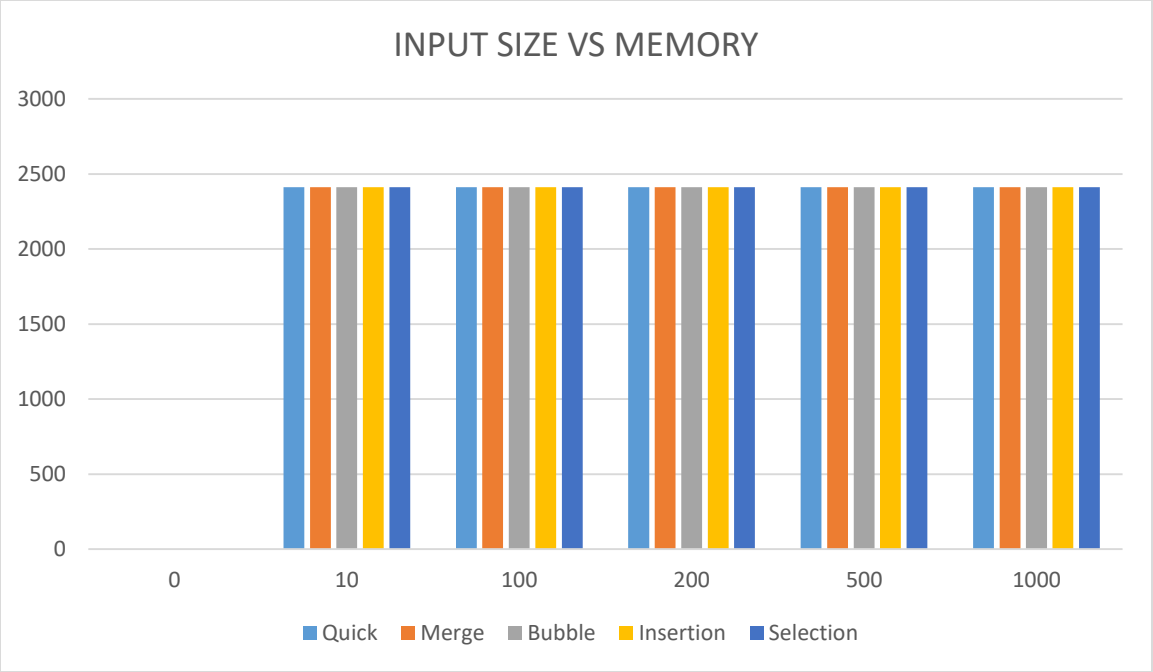
| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**QUICK SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 10000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 20000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 40000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |

**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 1000 | 1672 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 5000 | 2378 | 2378 | 2378 | 2378 | 2378 | 2378 |
| 10000 | 2378 | 2378 | 2378 | 2378 | 2378 | 2378 |
| 20000 | 2378 | 2378 | 2378 | 2378 | 2378 | 2378 |
| 40000 | 2378 | 2378 | 2378 | 2378 | 2378 | 2378 |




**INPUT SIZE VS MEMORY USAGE: (CAMERA DATA SETS)**

**INSERTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 5000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 10000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 20000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 40000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 5000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 10000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 20000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 40000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**BUBBLE SORT:**

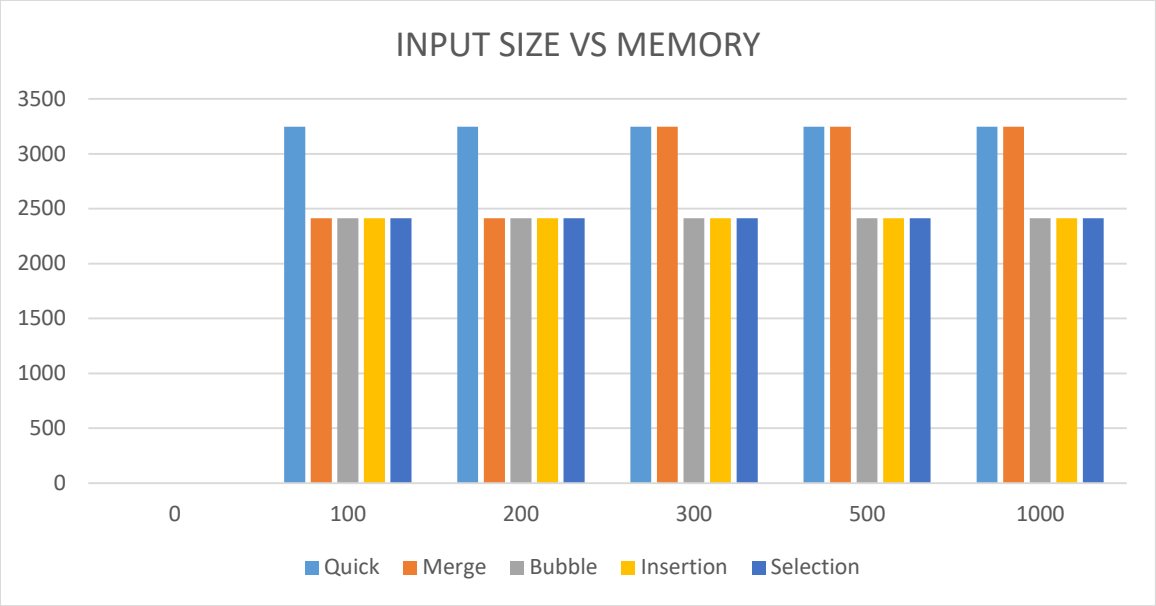
| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 5000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 10000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 20000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 40000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 5000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 10000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 20000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 40000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**QUICK SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 5000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 10000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 20000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 40000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |




**INPUT SIZE VS MEMORY USAGE: (PRICE DATA SETS)**

**INSERTION SORT**:

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 200 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 300 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 500 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**SELECTION SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 200 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 300 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 500 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**BUBBLE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 200 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 300 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 500 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 1000 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |

**QUICK SORT:**

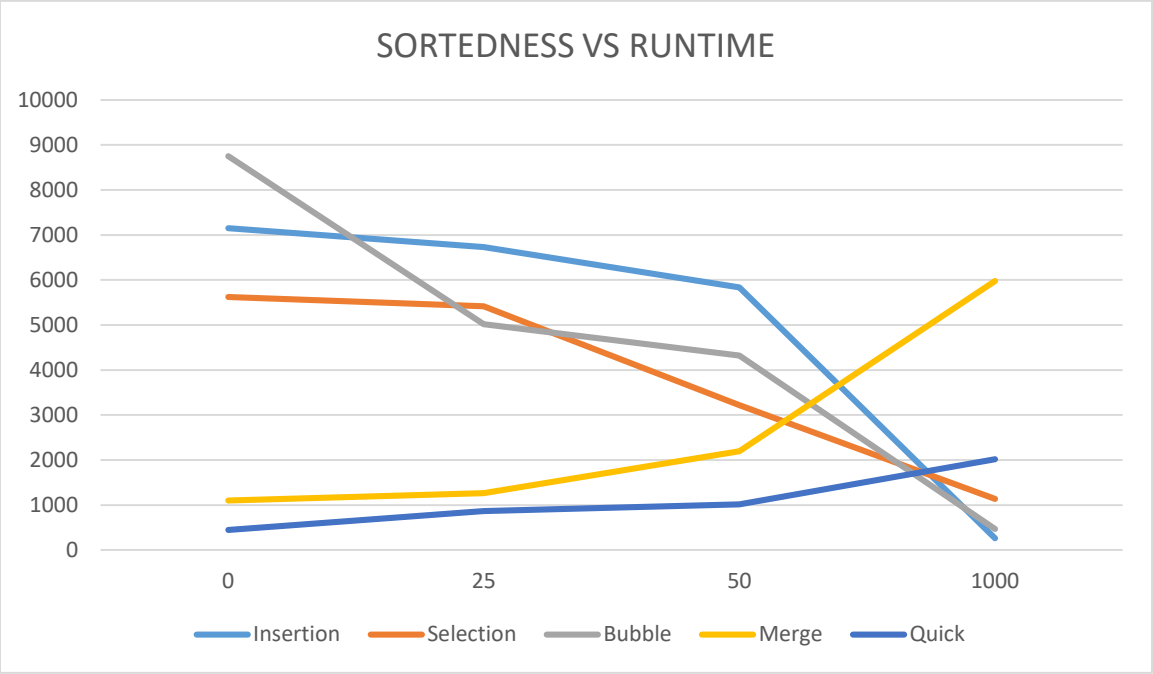
| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 100 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 200 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 300 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 500 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 1000 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |

**MERGE SORT:**

| Input | T1 | T2 | T3 | T4 | T5 | T(Avg) |
|---|---|---|---|---|---|---|
| 0 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 100 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 200 | 2412 | 2412 | 2412 | 2412 | 2412 | 2412 |
| 300 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 500 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |
| 1000 | 3248 | 3248 | 3248 | 3248 | 3248 | 3248 |

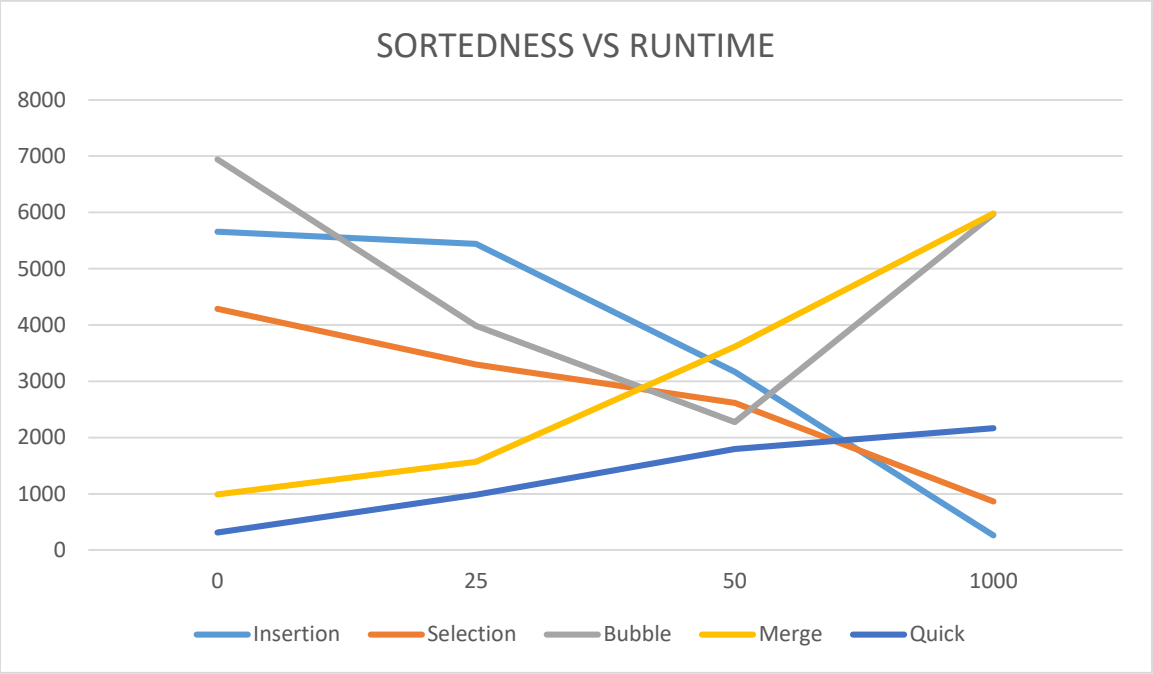
From graphs we conclude that:

The three sorting calculations Insertion sort, selection and bubble sort devour consistent memory, concerning the information estimate. Merge sort and quicksort just possess abundance memory when the information size is expansive. Merge sort performs more terrible than quicksort regarding memory, as it takes in an additional spaceduring the separation step.



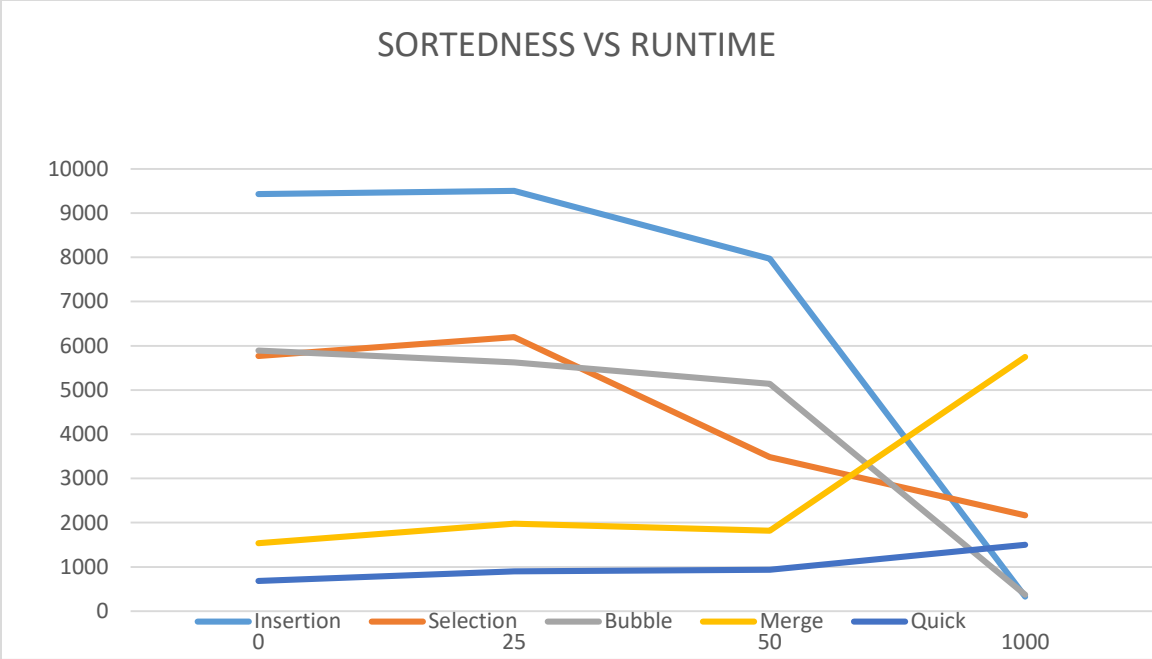

**SORTEDNESS RANDOM VS RUNTIME: (NORMAL)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 7149.2 | 5618.75 | 8751.2 | 1103.27 | 445.3 |
| 25 | 6734 | 5416 | 5018.9 | 1267 | 867.5 |
| 50 | 5832 | 3218.25 | 4318 | 21986 | 1018.9 |
| 1000 | 264 | 1137 | 467 | 5978 | 2019.5 |



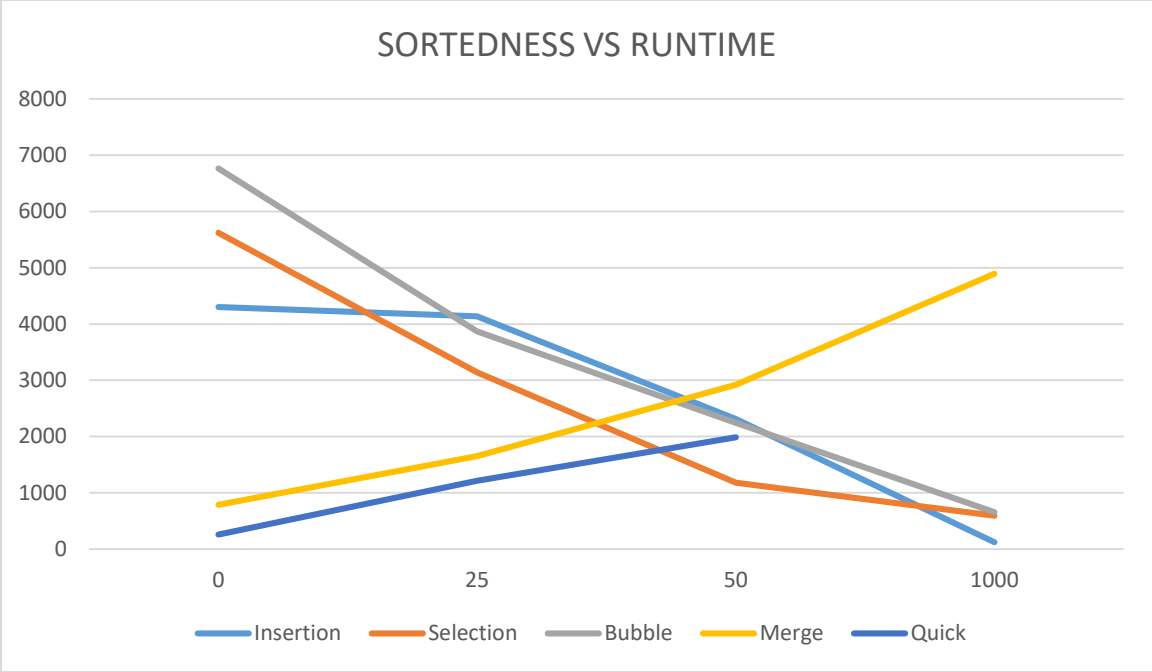

**SORTEDNESS RANDOM VS RUNTIME: (CAMERA -DATASETS)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 5658 | 4289.7 | 6943 | 987 | 312 |
| 25 | 5438.9 | 3296.8 | 3987.95 | 1568.25 | 986.25 |
| 50 | 3168.2 | 2617 | 2274.25 | 3618.75 | 1793.8 |
| 1000 | 179 | 866 | 5968 | 5982 | 2168.9 |




**SORTEDNESS RANDOM VS RUNTIME: (RANDOM DATASETS)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 9429 | 5796.25 | 5893 | 1534.25 | 683.25 |
| 25 | 9501.25 | 6195.75 | 5627.25 | 1976 | 896.75 |
| 50 | 7959 | 3482.65 | 5138.75 | 1817.65 | 937 |
| 1000 | 327 | 2167.2 | 373 | 5750 | 1498 |

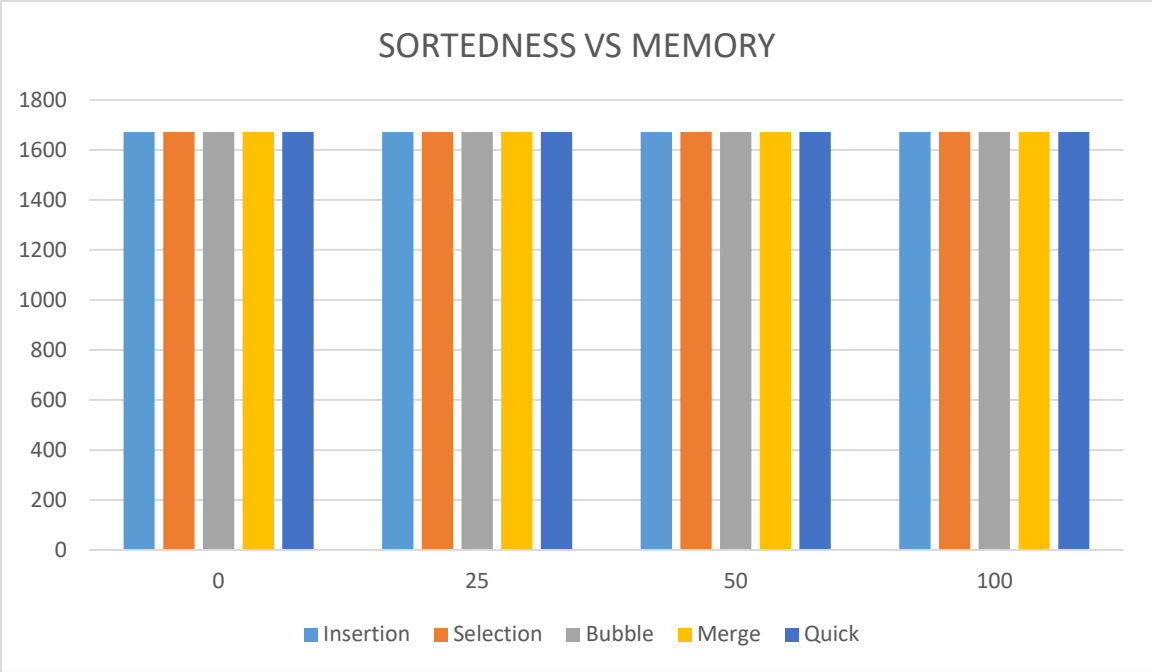



**SORTEDNESS RANDOM VS RUNTIME: (PRICE DATASETS)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 4301 | 5618.7 | 6761 | 789.2 | 259 |
| 25 | 4134.2 | 3139.8 | 3872.1 | 1653.8 | 1213.8 |
| 50 | 2312.5 | 1179.25 | 2243.2 | 2918.5 | 1988.5 |
| 1000 | 123 | 595 | 656 | 4891.2 | 2894.6 |

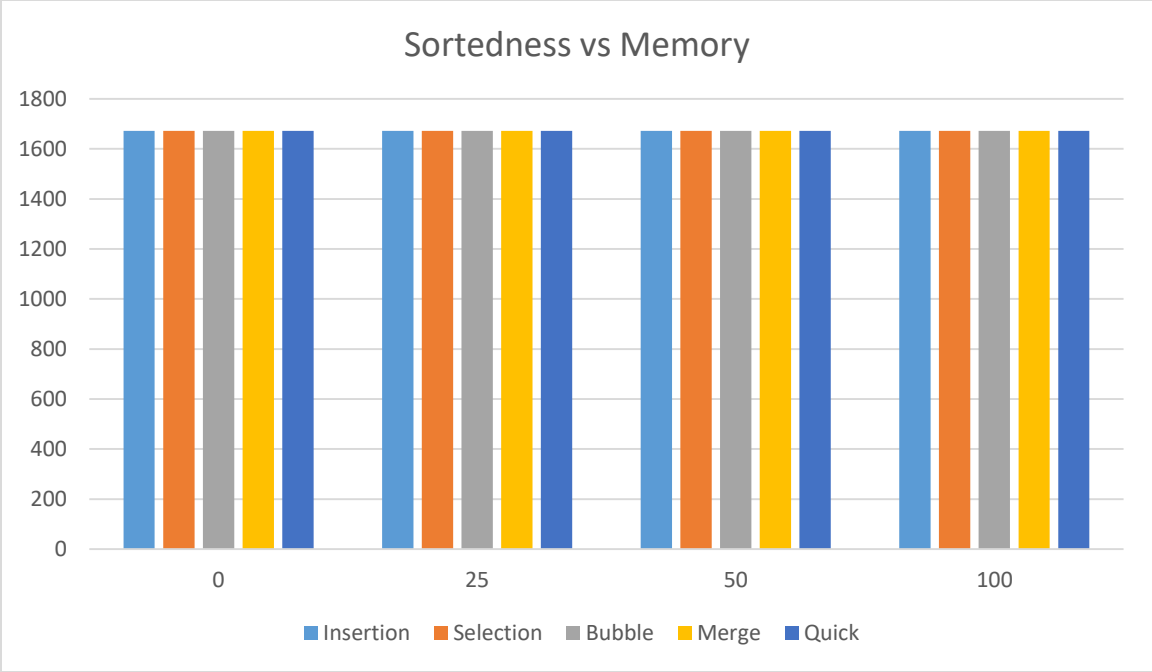
From the graphs we conclude that:

If an exhibit is not sorted, the three sorting calculations insertion sort, selection sort, bubble sort have higher runtime when contrasted with merge sort and quick sort on differing the information estimate. As the level of sortedness increments (25%,50%) the runtime of the three calculations insertion sort, bubble sort, selection sort diminishes, whereas the runtime of union sort and quicksort keep on increasing. When the cluster is totally sorted, merge sort and quick sort have the greatest runtime at the point when contrasted with bubble sort, insertion sort and selection sort. Runtime of merge sort increments radically when the exhibit is totally sorted.



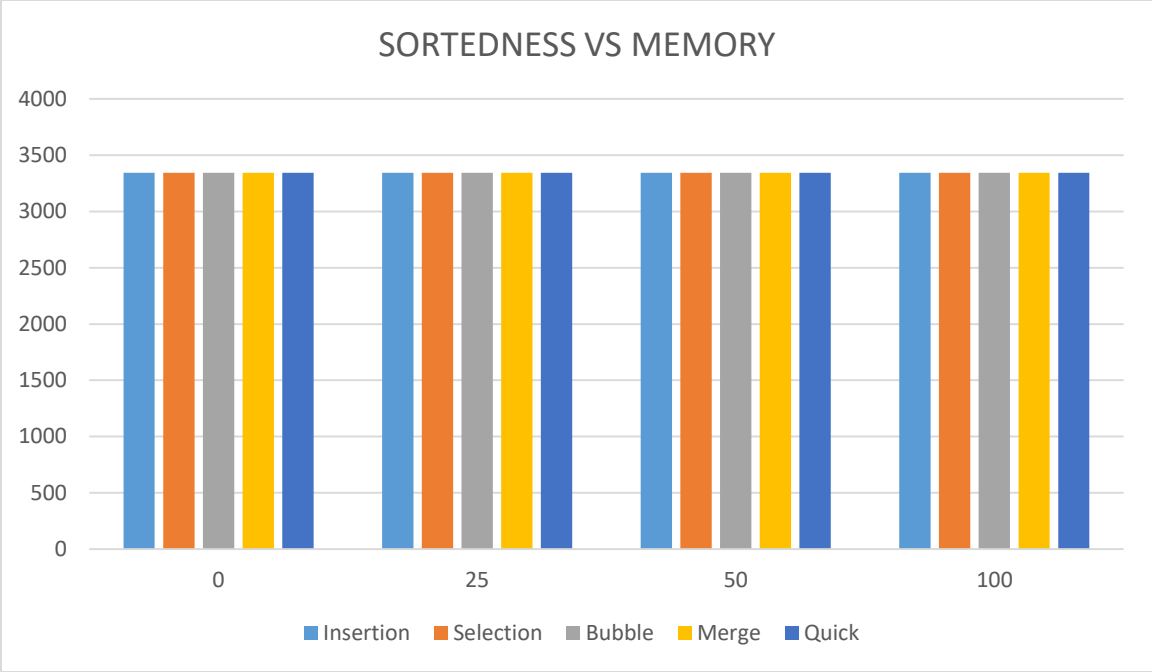

**SORTEDNESS VS MEMORY: (RANDOM)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 25 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 50 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 |



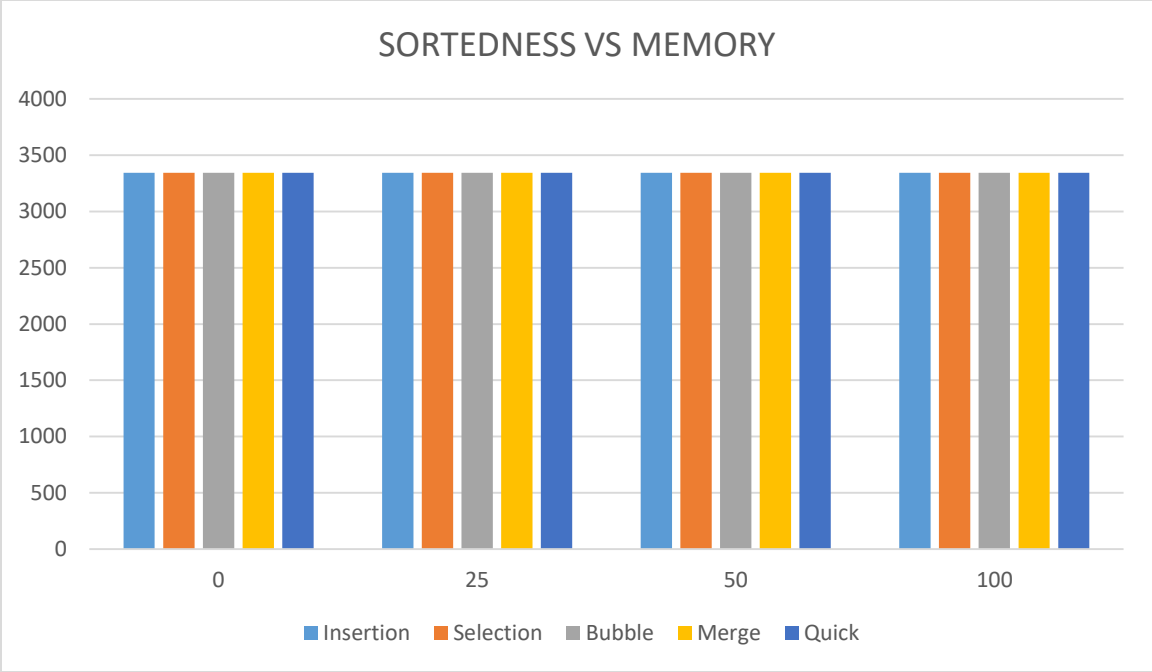

**SORTEDNESS VS MEMORY: (NORMAL)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 25 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 50 | 1672 | 1672 | 1672 | 1672 | 1672 |
| 100 | 1672 | 1672 | 1672 | 1672 | 1672 |




**SORTEDNESS VS MEMORY: (CAMERA -DATASETS)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 25 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 50 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 100 | 3344 | 3344 | 3344 | 3344 | 3344 |




**SORTEDNESS VS MEMORY: (PRICE DATASETS)**

| Input | Insertion | Selection | Bubble | Merge | Quick |
|---|---|---|---|---|---|
| 0 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 25 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 50 | 3344 | 3344 | 3344 | 3344 | 3344 |
| 100 | 3344 | 3344 | 3344 | 3344 | 3344 |

From the graphs we conclude that:

The memory of the 5 specified calculations does not rely on the level of sortedness .For all estimations of level of sortedness the calculation is as yet expending a similar steady measure of memory. The memory depends just on the info size of the algorithm.

## SUMMARY OF SORTEDNESS MEASURE:

- With high degree of sortedness, the merge sort and quick sort perform worse than bubble sort, insertion sort and selection sort.

- The space complexity of Merge sort and quick sort are greater than any other sorting algorithm as they are not in-place sorting algorithms. Other sorting algorithms such as insertion sort, bubble sort and selection sort are in-place sorting algorithms and their space complexity is O(1).

- The runtime of quick sort highly depends on choosing the pivot element. Consistent poor choices of pivot element can result in drastically slow O(n^2) complexity for quick sort. Choosing median as the pivot results in O(nlogn) complexity.

- When we have large amount of data the merge sort and quick sort perform a lot better than insertion sort, bubble sort and selections sort.

- Quick sort and merge sort work on the fundamental principle of divide and conquer. This makes them highly scalable sorting algorithms