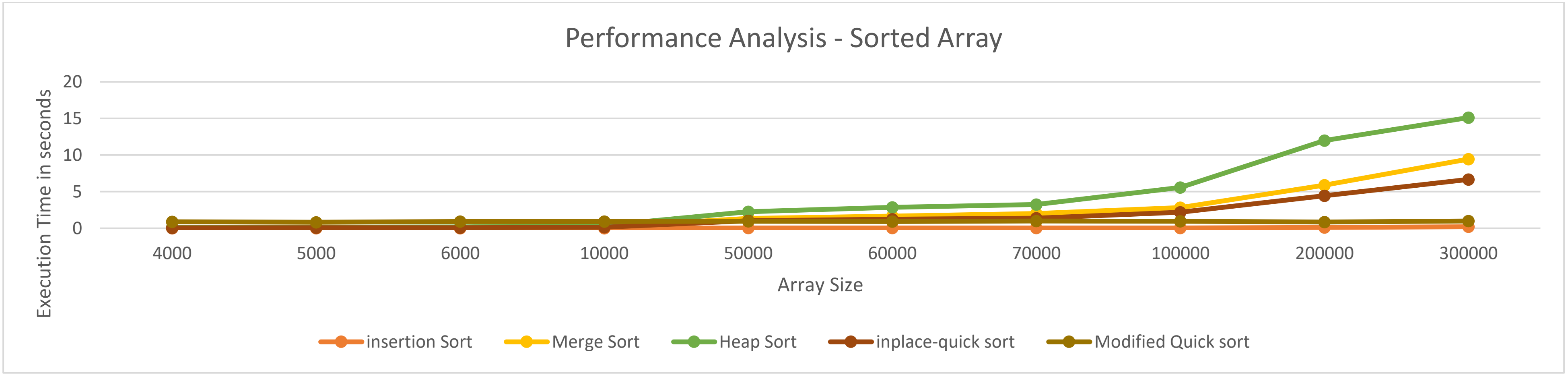
Submitted By Saranyaa Thirumoorthy (801223117)

Performance Analysis of Sorting Algorithms

Sorted Array

Sorting Algorithm's Performance for a Sorted Array with random numbers as elements and array sizes: 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000

	insertion Sort	Merge Sort	Heap Sort	inplace-quick sort	Modified Quick sort
4000	0.00366459	0.09328287	0.11807003	0.06183586	0.89778339
5000	0.00383757	0.10269601	0.16439453	0.0808334	0.82827348
6000	0.00413978	0.12365684	0.18988295	0.08264412	0.92585909
10000	0.00579898	0.20924938	0.38203336	0.14349782	0.913451
50000	0.02922686	1.34072766	2.28146827	1.03141708	0.95051329
60000	0.04441665	1.66669412	2.88077735	1.23515797	0.94007536
70000	0.04965664	2.02431648	3.23580574	1.38785582	0.98023469
100000	0.06046661	2.83170994	5.56365983	2.18940906	0.96055221
200000	0.13268139	5.90910824	11.99468216	4.4393947	0.86652528
300000	0.21037052	9.45184043	15.12183571	6.6741164	0.99701871

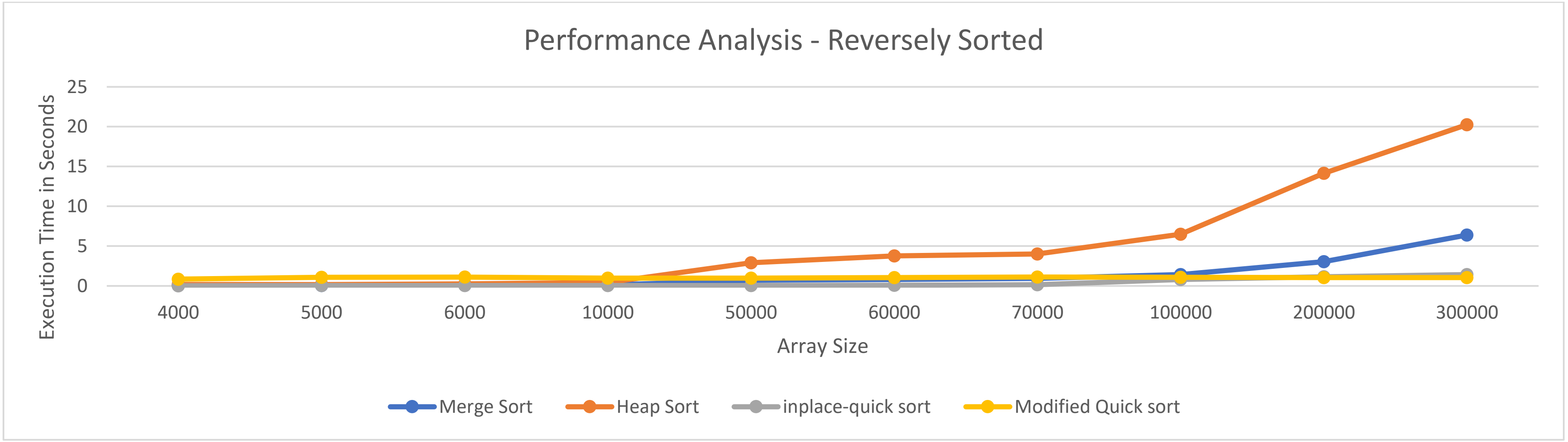


Submitted By Saranyaa Thirumoorthy (801223117)

Reversely Sorted Array

Sorting Algorithm's Performance for a reversely Sorted Array with random numbers as elements and array sizes: 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000

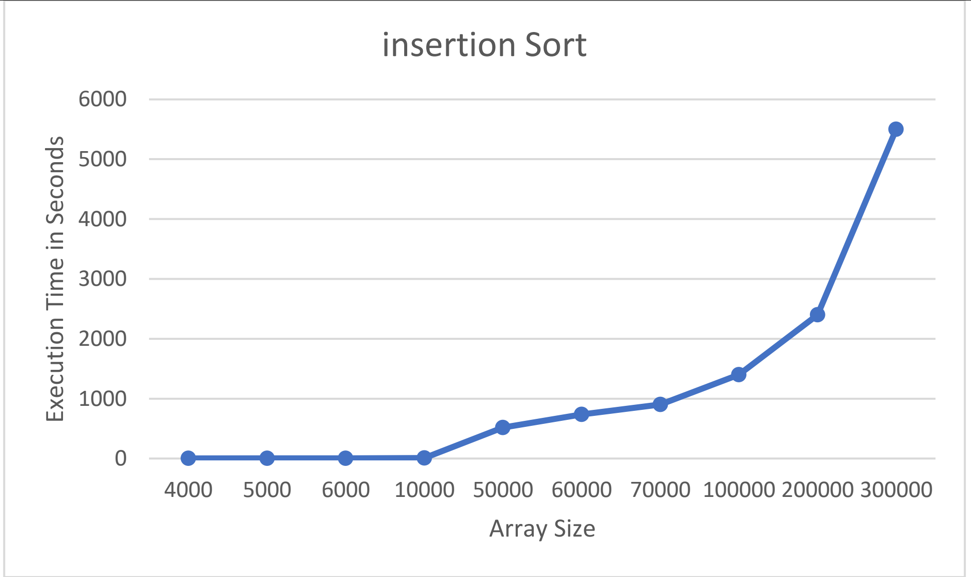
Array Size	Merge Sort	Heap Sort	inplace-quick sort	Modified Quick sort
4000	0.04525415	0.1759122	0.02775932	0.85743044
5000	0.05811771	0.17049933	0.03383155	1.07365054
6000	0.07132208	0.24822783	0.05556643	1.10782762
10000	0.12833512	0.43946442	0.06610995	0.98791339
50000	0.69995135	2.92226679	0.06469055	0.99578586
60000	0.80118559	3.76649096	0.06860892	1.03774978
70000	0.94620773	4.02487854	0.14096749	1.1071953
100000	1.42643979	6.50686198	0.82025505	1.07384202
200000	3.03489209	14.1321752	1.14338846	1.0406678
300000	6.39111711	20.25521756	1.42498293	1.0453801



Submitted By Saranyaa Thirumoorthy (801223117)

Insertion Sort reversely sorted array

Array Size	insertion Sort
4000	1.4497871
5000	2.1197134
6000	3.0966656
10000	8.5131185
50000	514.4549
60000	734.9816783
70000	900.3366975
100000	1400.900883
200000	2400.839335
300000	5500.221119



Submitted By Saranyaa Thirumoorthy (801223117)

Time Complexity & Data structure Used

Algorithm	Time Complexity	Data Structure Used
Insertion Sort	O(n)	array
Merge Sort	O(n log n)	array
Heap Sort	O(n log n)	array
In-place Quick Sort	O(n log n) (As I have implemented random pivot)	array
Modified Quick Sort	O(n^2)	array

About Code

Language Used: Python Version: 3.8 (To execute the code please use Python version 3.6 and Above)

Where to execute the code?

- Anaconda – Jupiter lab
- Visual Studio
- With Python and all the required packages in code installed normal command prompt should be fine
  Packages: matplotlib, numpy, time, random, array

How to execute the code?

If Visual Studio or Terminal of computer: use python <filename.py>

If Anaconda's Jupiter Lab: use shift + enter to execute (For Windows)

Concept [Sorted Array & Unsorted Array]

Array values used: Random value from range 1 to 80000

Array sizes used: 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000

Number of executions: 10

Final value considered: Average of 10 executions for each and every array size

Submitted By Saranyaa Thirumoorthy (801223117)

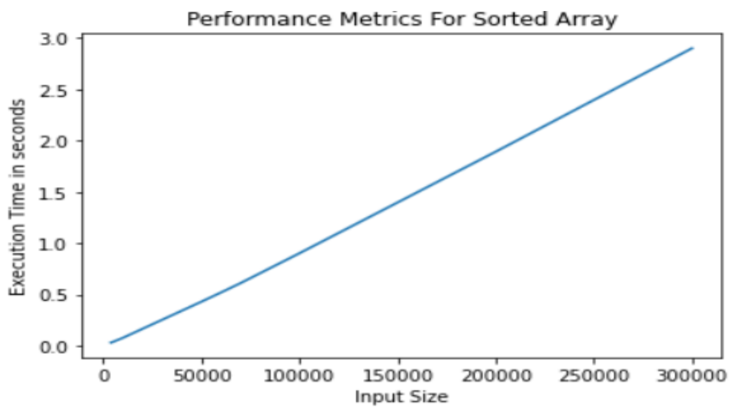
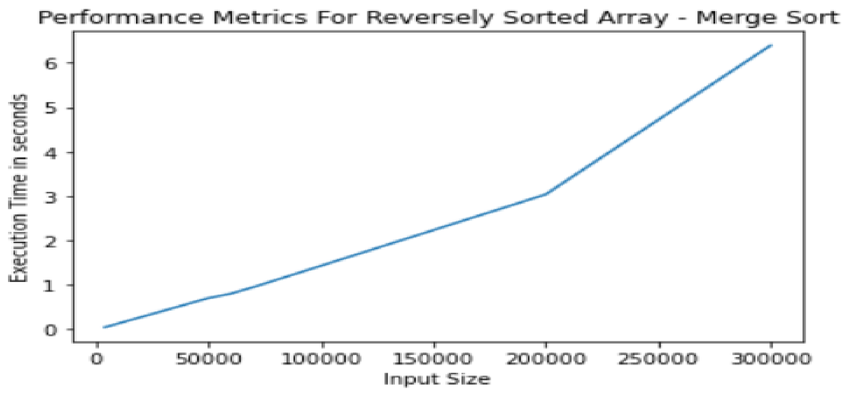
Individual Execution Results for Sorts

Insertion Sort

Sorted Array	Reversely sorted Array
Result Average of Execution Time for 10 rounds, for each of the used array sizes [0.003664589999925738, 0.003837569999996049, 0.004139780000059545, 0.005798980000008669, 0.02922685999992609, 0.0444166499999028, 0.0496566399999665, 0.060466610000048604, 0.1326813900000161, 0.21037051999987852] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.	**Result** Average of Execution Time for 10 rounds, for each of the used array sizes [1.4497871,2.1197134,3.0966656,8.5131185,514.4549,734.9816783,900.3366975,1400.900883,2400.839335,5500.221119] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.

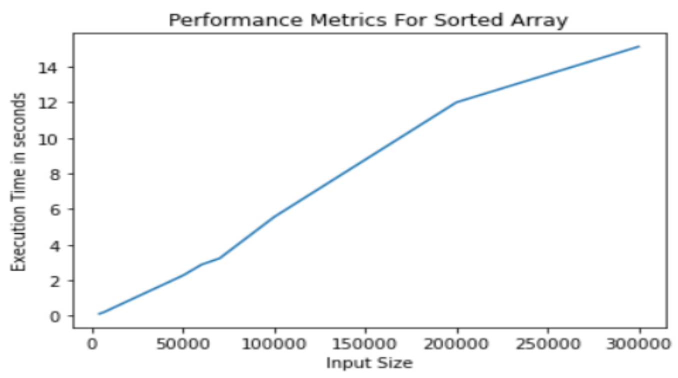
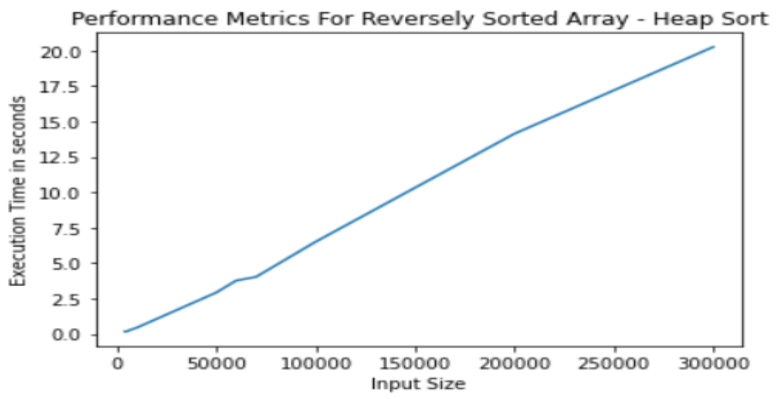
Submitted By Saranyaa Thirumoorthy (801223117)

Merge Sort

Sorted Array	Reversely sorted Array
<u>Result</u> Average of Execution Time for 10 rounds, for each of the used array sizes 0.09328286999999377, 0.10269601000000535, 0.12365684000000102, 0.2092493799999957, 1.3407276600000002, 1.6666941199999983, 2.0243164800000057, 2.8317099399999988, 5.909108240000002, 9.451840430000003] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.	**<u>Result</u>** Average of Execution Time for 10 rounds, for each of the used array sizes [0.04525415000002795, 0.05811770999998771, 0.07132207999993626, 0.12833512000001973, 0.6999513500000376, 0.8011855900000228, 0.9462077299999692, 1.4264397899999495, 3.0348920899999485, 6.391117109999914] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.
	

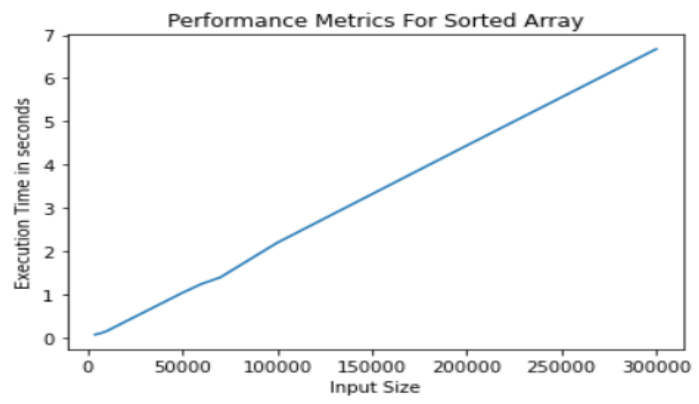
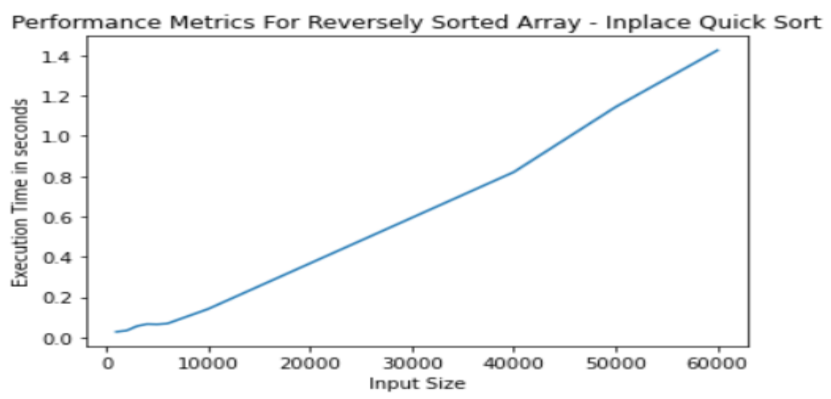
Submitted By Saranyaa Thirumoorthy (801223117)

Heap Sort

Sorted Array	Reversely sorted Array
<u>**Result**</u> Average of Execution Time for 10 rounds, for each of the used array sizes [0.11807002999999554, 0.1643945299999814, 0.18988295000000904, 0.38203336000000265, 2.2814682700000164, 2.880777350000011, 3.2358057399999893, 5.563659829999994, 11.994682159999996, 15.12183570999997] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.	<u>**Result**</u> Average of Execution Time for 10 rounds, for each of the used array sizes [0.17591220000003888, 0.1704993300001661, 0.24822782999999618, 0.4394644199999675, 2.9222667900000032, 3.7664909599999192, 4.0248785400001, 6.50686198000003, 14.132175199999939, 20.255217560000027] Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.
	

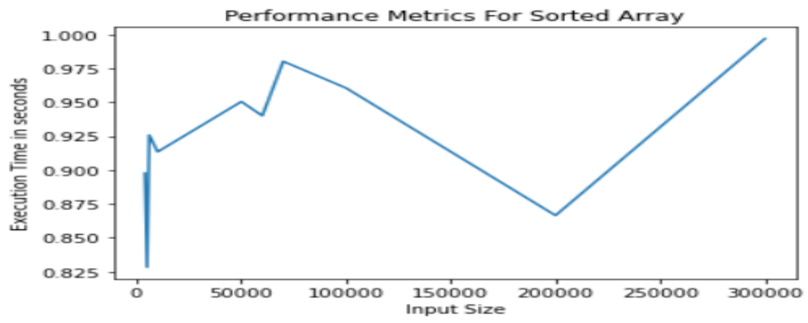
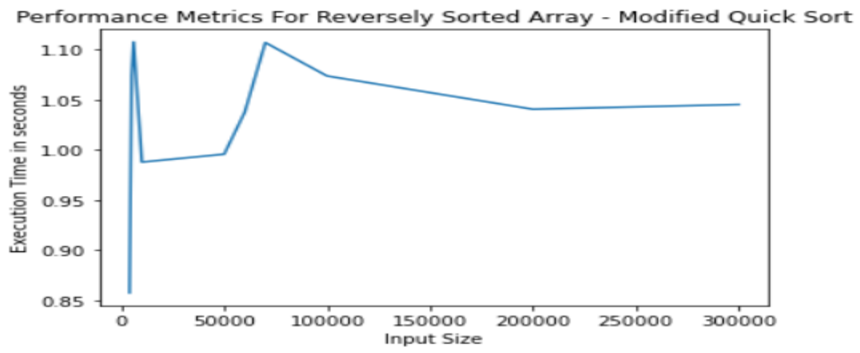
Submitted By Saranyaa Thirumoorthy (801223117)

In-Place Quick Sort

Sorted Array	**Reversely sorted Array**
Result	**Result**
Average of Execution Time for 10 rounds, for each of the used array sizes	Average of Execution Time for 10 rounds, for each of the used array sizes
[0.06183586000008745, 0.08083340000002863, 0.08264412000000902, 0.14349782000001596, 1.0314170799999602, 1.2351579699999775, 1.3878558199999815, 2.1894090599999343, 4.43939469999998, 6.674116400000071]	[0.027759320000041044, 0.033831550000058996, 0.05556642999995347, 0.06610994999996364, 0.06469054999993204, 0.068608919999906, 0.14096749000000272, 0.8202550499999234, 1.1433884599999147, 1.424982929999942]
Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on	Where the 1st value is the average value of 10 rounds for array size: 4000 Where the 2nd value is the average value of 10 rounds for array size: 5000 and so on
Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000	Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000
Note: The Execution time might slightly vary when newly run every time.	Note: The Execution time might slightly vary when newly run every time.
	

Submitted By Saranyaa Thirumoorthy (801223117)

Modified Quick Sort

Sorted Array	Reversely sorted Array
<u>**Result**</u> Average of Execution Time for 10 rounds, for each of the used array sizes [0.8977833900000632, 0.8282734800000299, 0.9258590900000172, 0.9134509999999864, 0.9505132900000035, 0.9400753600000143, 0.9802346900000203, 0.9605522099999917, 0.8665252800000417, 0.9970187099999748] Where the 1^{st} value is the average value of 10 rounds for array size: 4000 Where the 2^{nd} value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.	<u>**Result**</u> Average of Execution Time for 10 rounds, for each of the used array sizes [0.8574304400000073, 1.0736505400000624, 1.10782762000008, 0.9879133899999488, 0.9957858599999782, 1.0377497800001039, 1.1071952999999666, 1.0738420199999836, 1.040667799999983, 1.0453801000000567] Where the 1^{st} value is the average value of 10 rounds for array size: 4000 Where the 2^{nd} value is the average value of 10 rounds for array size: 5000 and so on Array size order 4000,5000,6000,10000,50000,60000,70000,100000,200000,300000 Note: The Execution time might slightly vary when newly run every time.
	

Submitted By Saranyaa Thirumoorthy (801223117)

Observation:

Based on the execution results and the average execution time taken from 10 rounds of executions, Quick sort works efficiently. Merge sort is the second choice of go to execution. Insertion sort works efficiently for an already sorted array. However, insertion sort works poorly for a reversely sorted array and I feel it is not advisable to use insertion sort in this case. Heap sort is time consuming too.