Performance Measurement of Popular Sorting Algorithms Implemented using Java and Python

Conference Paper · November 2022

DOI: 10.1109/ICECCME55909.2022.9988424

2 authors:

Omar Khan Durrani
Ghousia college of Engineering
11 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Sayed Abdulhayan
P.A.College of Engineering
5 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Performance Measurement of Popular Sorting Algorithm Implemented using Java and Python

Omar Khan Durrani, Dr. Sayed AbdulHayan

Department of Computer Science & Engineering, Ghousia College E,ngineering, affiliated to VTU Ramanagara,BangaloreRegion

omardurrani2003@yahoo.com

sabdulhayan.cs@pace.edu.in

***Abstract*- In modern technology where disciplines like data sciences, data Analytics, and machine learning are emerging and infrastructure being set for Internet of things, important operations like sorting and searching plays a vital role. Hence efficient sorting along with searching is still important for the efficiency of other algorithms. These emerging areas are using 4GL and 5GL languages for its implementation. Also it is seen fever analysis ismade on algorithms using 4GL languages though there exist sufficient analysis and results using languages like C ,C++, and Java. In this paper we have made a thorough asymptotic analysis and performance measures using Python and Java languages. We conducted our experiments on Pseudo random data and sorted data. The research shows that Mergesort doing well for Python implementation instead of Quicksort which very much lacks in its Performance. Quicksort remains excellent in performance for Java implementation.**

Keywords—sorting algorithms, asymptotic analysis, python, java, test bed, performance measurement,compiler design;

I. INTRODUCTION

In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly; it is also often useful for canonicalizing data and for producing human-readable output.

More formally, the output must satisfy two conditions:
1. The output is in non-decreasing order
2. The output is a permutation, or reordering, of the input.

Since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. For example, bubble sort was analyzed as early as 1956. later other algorithms like selection sort, insertion sort merge sort, quick sort and many others methods were introduced which we term as popular sorting algorithms. Due to fast evolution in computer technology, although many consider it a solved problem, useful new sorting algorithms are still being invented. The evolution in technologies like Data sciences, Data Analytics , Machine learning etc also gives the researchers an opportunity to come up with still better solutions.

II. ASYMPTOTIC ANALYSIS

The word asymptotic means that it is the study of function of a parameter **n**, as **n** become larger and larger without any bounds. Here we are concerned with how the runtime of an algorithm increases with the size of the input. We can do analysis of algorithms which will accurately reflect the way in which the computation time will increase with the size, but ignores the other details with little effect on total. It permits us to provide upper and lower bounds on the value of a function **f** for suitably large values. The efficiency analysis frame work concentrates on order of growth of an algorithms' basic operation count as the principal indicator of the algorithms' efficiency. To compare and rank such orders of growth, computer scientist use three notations:

- O(big-oh)
- Ω(big-omega)
- Θ(big-theta)

O(big-oh) bounds from above ,Ω (big-omega) from bottom and Θ(big-theta) does from both above and below, details study is given in [1][2].

In the following section we have briefly explained the theoretical aspects of the popular sorting algorithm namely **merge sort, quick sort and others**. Further we have conducted the experiments on these sorting algorithms using two test beds to realize their behavior and further investigate in their performance with respect to the fourth generation (4GL) popular programming languages.

III. SORTING ALGORITHMS

A. *MERGE SORT*

The merge sort splits the list to be sorted into two equal halves, and places them in separate arrays. This sorting method is an example of the DIVIDE-AND-CONQUER paradigm i.e. it breaks the data into two halves and then sorts the two half data sets recursively, and finally merges them to obtain the complete sorted list. The merge sort is a comparison sort and has an algorithmic complexity of $O(n \log n)$. Elementary implementations of the merge sort make

use of two arrays - one for each half of the data set. The following image depicts the complete procedure of merge sort.

Pros: 1) marginally faster than the heap sort for larger sets. 2) Merge Sort always does lesser number of comparisons than Quick Sort. Worst case for merge sort does about 39% less comparisons against quick sort's average case. 3) Merge sort is often the best choice for sorting a linked list because the slow random-access performance of a linked list makes some other algorithms (such as quick sort) perform poorly, and others (such as heap sort) completely impossible.

Cons: 1) At least twice the memory requirements of the other sorts because it is recursive. This is the BIGGEST cause for concern as its space complexity is very high. It requires about a Θ (n) auxiliary space for its working. 2) Function overhead calls (2n-1) are much more than those for quick sort (n). This causes it to take more time marginally to sort the input data.

B. *Quick Sort*

Quick Sort is an algorithm based on the DIVIDE-AND-CONQUER paradigm that selects a pivot element and reorders the given list in such a way that all elements smaller to it are on one side and those bigger than it are on the other. Then the sub lists are recursively sorted until the list gets completely sorted. The time complexity of this algorithm is O (n log n).

Pros: 1) One advantage of parallel quick sort over other parallel sort algorithms is that no synchronization is required. Sufficient work on asymptotic analysis and performance measurement of the above popular sorting algorithms have been done in [1][4]. Also my paper [8] has made asymptotic analysis and performance measurement of popular sorting algorithms by implementing them in C language and has exhibited the true behaviors of the popular sorting algorithms.the paper also shows the point or the time taken for that sample from which onwards the sorting algorithm enter into the Asymtotic beaviour range (ABR). Further When we found less work of analysis of popular sorting algorithms using Java and Pyhton programming languages which are being popularly used in the fields of machine learning , data sciences and data analytics. Therefore, in order to find how well these perform when they are used to implement popular sorting algorithms and observe their performance and behaviors. A new thread is started as soon as a sub list is available for it to work on and it does not communicate with other threads. When all threads complete, the sort is done. 2) All comparisons are being done with a single pivot value, which can be stored in a register. 3) The list is being traversed sequentially, which produces very good locality of reference and cache behavior for arrays.

Cons: 1) Auxiliary space used in the average case for implementing recursive function calls is O (log n) and hence proves to be a bit space costly, especially when it comes to large data sets. 2) Its worst case has a time complexity of O (n^2) which can prove very fatal for large data sets. Competitive sorting algorithms

C. *Insertion Sort, Selection Sort and Bubble Sort*

These are classified as n^2 class of sorting algorithms,these very slow than the nlogn class discussed in the above section. Insertion sort collects and places the first element in the first index of empty array and subsequent elements are placed by comparing it with the elements from one side and after insertion in its place the algorithm shifts the elements of other side, this is repeated for every new insertion.

Bubble sort does the sorting by comparing n elements and swaps the the unsorted into the order, this process is repeated for similar **n** passes.

Selection sort, which also does the sorting in **n** passes, at every pass it picks the min number and puts it in its order. In selection sort few comparison are avoided in comparison to bubble sort making it little better.

These basic algorithms, the Bubble sort or Selection sort are inefficient in nature but very easy to code. They are of importance in academic studies but are not in concern at practices. On the other side Insertion sort though being in the class of order $\mathbf{n^2}$ performs well to sort small sample sizes along with nlogn class algorithms. Insertion sort is faster among the n^2 class of sorting algorithms. As our intention of this paper is to make analysis and performance measurement of sorting algorithms implemented using the most popular programming languages Java and Python. The performance measurement and observations are only possibleby considering asymptotic analysis which is explained briefly in the next section.

IV. Related Work

Sufficient work on asymptotic analysis and performance measurement of the above popular sorting algorithms have been done in [1][4]. Also my paper [8] has made asymptotic analysis and performance measurement of popular sorting algorithms by implementing them in C language and has exhibited the true behaviors of the popular sorting algorithms. The paper also shows the **Asymptotic Point(AP)** or the time taken for that sample from which onwards the sorting algorithm enter into the Asymtotic beaviour range (ABR), here Range refers to data samples or set greater than that sample size we achieved the **AP**. Further, when we found less work of analysis of popular sorting algorithms implemented using Java and Pyhton programming languages which are being popularly used in the fields of machine learning , data sciences and data analytics encouraged us to consider the 4GLs. Therefore, in order to find how well Python and Java perform for popular sorting algorithms and observe their performance and behaviors we took up this research.

V. Performance Measurement

A. *Experiment Set-up*

Performance measurement is concerned with obtaining the actual time requirements of a program. To obtain the execution time of a program, we need a clocking mechanism. We have used the Python method time() by importing time class from its library, which measures time in seconds. Similarly, nanoTime() which clocks in nano seconds later the same was converted to seconds.

As we expect the asymptotic behavior to begin for larger values of n as it is shown in [8]. for safer side we have our sample size n=1000,2000,.. ,10000,20000,..100000..500000. we have considered two class of data samples for conducting our experiment to testify the average and the worst case of algorithms. First (samples for average case analysis), random samples which we generated by using random numbers generator methods in both python and java languages by importing their respective classes given below:

Import random and the method random.randint()

Import java.util.Random; and the method Rantdom() called through its object,ie., Random generator = new random().

Second(sample for worst case analysis), array index assigned as the array element, a[i]=i or by using the reverse() method in python language. As there is not much consideration in practice for Best case analysis we neglect it.

The two driver codes, which initiates the experiment execution for both python and Java respectively are given below:

 # Driver code of Python

```
a=[]
for n in range(10000,100001,10000):
   #generating random numbers in range 1..50
   #append method adds the generated number into the list a
   #for i in range(n):
         #a.append(random.randint(1, 50))
   #generating the sample data for worst case analysis
   for i in range(n):
         a.append(i)
   a.reverse()
   start=time.time()
   #bubbleSort(a,n)
   #selectionSort(a,n)
   #insertionSort(a,n)
   #mergeSort(a,n)
   #heapSort(a)
   #quickSort(a, 0, n - 1)
   quickSortIterative(a, 0, n-1)
   elapsed=time.time()-start
   print(elapsed)
     #elapsed is the time taken by the algorithm to sort the
         #sample
```

// Driver code of Java program (calling selection sort)

```
public static void main(String[] args){
int[] a;
int i,step=1000;
 for(int n=1000;n<=max;n=n+step){
          if (n >= 10000) step=10000;
          a = new int[max];
          Random generator = new Random();
          for(i=0;i<n;i++)
                a[i]=generator.nextInt(100));
          long startTime = System.nanoTime();
          SelectionSort ss = new SelectionSort();
          ss.selectionSort(a);
          long stopTime = System.nanoTime();
          long elapseTime = (stopTime - startTime);
          System.out.println("Time taken to Sort n elements
          ="+n + " is :" +elapseTime + " nanoseconds\n");
           } //sample loop
} // main body end
```

Note: we enable each call of sorting method by removing the # (comment symbol) and disabling the others by adding '# ' symbol for the other sorting methods calls in case of python. Similarly we do it in case of java with the help of '// 'symbols. The sample sizes are also as per the range needed in case of python with the help of range() function and using for loop with suitable step size in case of Java to form the range of data and random method used in both python and java to generate pseudo random numbers.
We have used two system configurations for conducting our experiments , test bed 1 and test bed 2. The system configuration of each of test bed is specified below.

Test bed 1:Online compiler like Programiz which is in[11] used to execute our python and Java codes. The configurations are not mentioned on their websites which is found to be litle faster then test bed 2. We have used this bed 1 as a cross reference to our Test bed 2 system.

Test bed 2: Memory of 3.7 GiB, Intel® Core™ i3-6006U CPU @ 2.00GHz × 4 ,64-bit, 970.9 GB

As mentioned earlier we have used two types of data set, the random data set and ordered data set (for worst case analysis, generation of these data is mentioned above) over which we conduct our experiments.

B. Experiment results and Observations

We have conducted **two different class of experiments**, in the first one we conducted experiments on different samples of pseudo random numbers using both Python and Java languages and the second one conducting experiments on different samples of decreasing order numbers using both Python and Java languages. Also for cross reference purpose we executed for each class of data on a online compiler available for both Python and Java languages. Following

paragraphs illustrates the observations made from the two perspective, of implementations with respect to two different programming languages java and Python for two class of data (random data and decreasing order data) .

i. Experimenting with Random data set

Table 1,Table 2 ,Table 3 and Table 4 along with their respective graph plots in Figure 1, Figure 2, Figure 3 and Figure 4 shows the results of experiments done on Pseudo random numbers using Python and Java languages. By making observation from Table 1, Table 2, Table 3 and Table 4 and their respective Graphs shows clearly that sorting algorithms Merge sort, Insertion sort, Bubble sort and Selection sort have performed as per their asymptotic behavior and order of growths as explained in[1][2][3]. Quick sort and Merge sort being the fast as it belong to the nlogn class among the order of growth classes as listed in [2]. Insertion sort,selection sort and bubble sort are slow in execution since they belong to quadratic ($\mathbf{n^2}$) class among the order of growth classes. Further it is seen that Quick sort though faster than $\mathbf{n^2}$ class has performed slow in its python implementations and remained as fastest in the Java implementation. The reasons of being slow in python implementation can be due to compiler design for Python language which is the basis of research study**.** Python implementation on test bed 1 and test bed 2 shows time clocking difference, which could be of reasons like difference in compiler versions used and the processor speed. The on-line compiler which we have used has not provided its system configurations and also our purpose of usage is for cross reference our system ie. the test bed 2.

We notice missing of time values in Table 1 for online compiler(Test bed 1) which could be due to network delays and since the sample size is large which needed few seconds for bubble sort code to give the results the same happened with insertion sort for n=100000. Also we happen to come across a runtime error while executing the quick sort implementation using python on both test beds which was due to the recursion stack overflow. The exception message is shown below:

Traceback (most recent call last):

File "QMHISBSORT_time.py", line 328, in <module>

quickSort(a, 0, n - 1)

pi = partition(array, low, high)

RuntimeError: maximum recursion depth exceeded

The exception was removed by increasing the recursion stack limit by importing the sys(import sys) class's method sys.setrecursionlimit(size) and executing it with size=5000 (default size=1000)in the code hence the solution was found which we got it from [6]. The runtime stack overflow was not encountered with java implementation for random data samples indicating the strength and robustness of java language. This observation indicates that java could be preferred over python for sorting larger samples. Also experiments done in [7] shows java better than C++ programming language.

Also an important observation seen is that Quick sort is slowest among the nlogn class (of growth orders) with respect to random data set when implemented using Python language, as we see time taken for sample size 100000 is 13.47 in table 1 (test bed 1) and 17.463 in Table 2 (test bed 2) where as time taken for the same sample size for Merge sort is 4.437 and 2.689 indicating that merge sort is fastest with Python Implementation than Quick sort. The same case with respect to java implementation as seen in Table 3 and Table 4 we observe that quick sort excels with time taken 0.114 and 0.1122 (in two test beds respectively) for sample size 100000. Also we observe that Insertion sort performing better than Merge sort for very large values of **n(samples),which is of our future study** as it is related to the design of compiler. The Bubble sort and selection sort remains unchanged with respect to the proportion of behaviors with respect to Python and Java langua**ges.**

TABLE 1
ASYMPTOTIC TIME TAKEN ON TEST BED 1 USING PYTHON CODE FOR RANDOM DATA SAMPLES

Sample size	Quick Sort	Merge Sort	Insertion Sort	Selection Sort	Bubble Sort
	Time in sec	Time in sec	Time in sec	Time in sec	Time in sec
1000	**0.003**	**0.003**	0.032	0.106	0.079
2000	0.010	0.065	0.129	0.630	0.286
3000	0.020	0.031	0.147	1.266	0.598
4000	0.034	0.038	0.171	2.312	1.030
5000	0.064	0.066	0.220	1.223	1.523
6000	0.086	0.085	0.260	1.354	2.137
7000	0.146	0.117	0.317	1.833	3.126
8000	0.149	0.162	0.351	3.051	3.700
9000	0.180	0.197	0.402	3.100	4.561
10000	0.313	**0.040**	5.754	2.347	5.634
20000	0.715	0.133	14.328	9.754	66.485
30000	1.995	0.266	24.935	34.538	79.302
40000	3.131	0.470	31.890	41.387	190.600
50000	3.410	0.730	39.994	118.772	212.082
60000	4.892	1.058	54.902	126.825	257.040
70000	6.625	1.795	61.307	132.009	-
80000	8.637	1.923	69.691	279.583	-
90000	10.94	2.448	99.619	242.325	-
100000	13.47	**4.437**	-	390.749	-

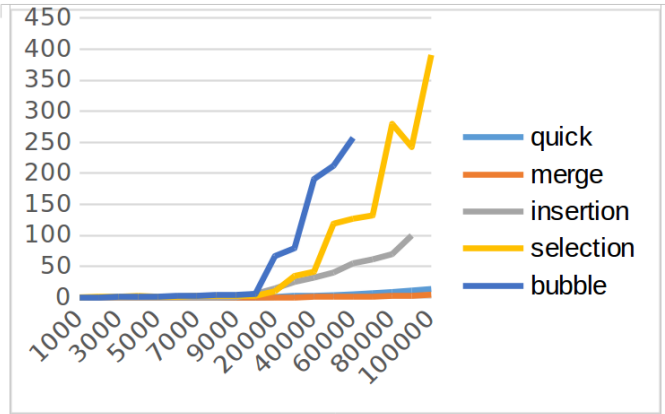

FIG 1 :GRAPH PLOT FOR TABLE 1 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

TABLE 2

ASYMPTOTIC TIME TAKEN ON TEST BED2 USING PYTHON CODE FOR RANDOM DATA SAMPLES

Sample size	Quick Sort Time in sec	Merge sort Time in **sec**	Insertion Sort Time in sec	Selection sort Time in sec	Bubble sort Time in sec
1000	**0.004**	**0.006**	0.034	0.032	0.070
2000	0.016	0.014	0.098	0.126	0.254
3000	0.025	0.032	0.171	0.287	0.515
4000	0.037	0.073	0.236	0.514	0.863
5000	0.059	0.118	0.296	0.809	1.283
6000	0.076	0.163	0.362	1.175	1.832
7000	0.105	0.162	0.422	1.603	2.415
8000	0.135	0.224	0.519	2.114	3.034
9000	0.165	0.299	0.571	2.702	3.785
10000	**0.193**	0.354	0.615	3.389	4.615
20000	0.700	0.160	10.191	13.166	25.718
30000	1.543	0.328	16.443	32.573	55.480
40000	2.725	0.556	23.398	61.848	90.736
50000	4.510	0.848	29.950	101.888	130.150
60000	6.142	0.639	36.043	156.848	183.345
70000	8.777	1.209	42.907	219.166	248.706
80000	10.786	1.635	49.679	301.854	322.293
90000	13.948	2.130	57.422	271.083	397.109
100000	17.463	**2.689**	63.824	348.351	748.495

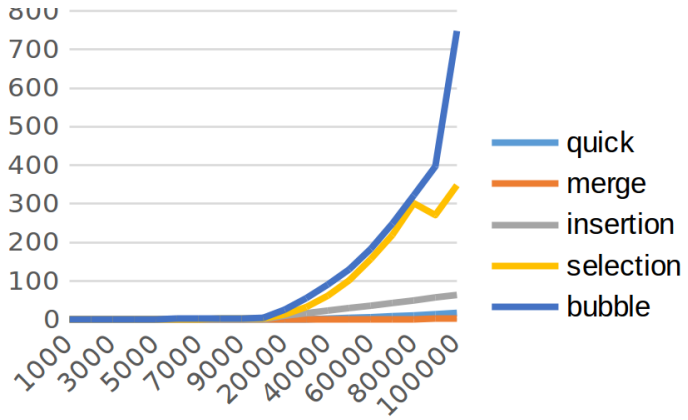

FIG 2: GRAPH PLOT FOR TABLE 1 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

TABLE 3

ASYMPTOTIC TIME TAKEN IN SECONDS ON TEST BED 1 USING JAVA CODE FOR RANDOM DATA SAMPLES

Sample size	**Quick Sort** Time in sec	**Merge Sort** Time in sec	**Insertion Sort** Time in sec	**Selection Sort** Time in sec	**Bubble Sort** Time in sec
10000	**0.015**	2.615	**0.015**	0.053	0.132
20000	0.007	4.24	0.045	0.22	0.446
30000	0.062	6.103	0.098	0.167	0.889
40000	0.018	7.76	0.168	0.784	1.834
50000	0.017	9.491	0.212	1.28	3.62
60000	0.074	10.93	0.314	1.879	4.964
70000	0.056	12.504	0.386	2.73	6.109
80000	0.82	18.809	0.472	3.243	8.767
90000	0.098	16.757	0.644	4.771	11.004
100000	**0.114**	19.849	0.954	5.176	13.467
200000	0.454	38.055	3.68	7.611	50.698
300000	1.092	59.27	8.836	18.34	116.454
400000	**1.648**	84.644	21.473	33.496	225.051

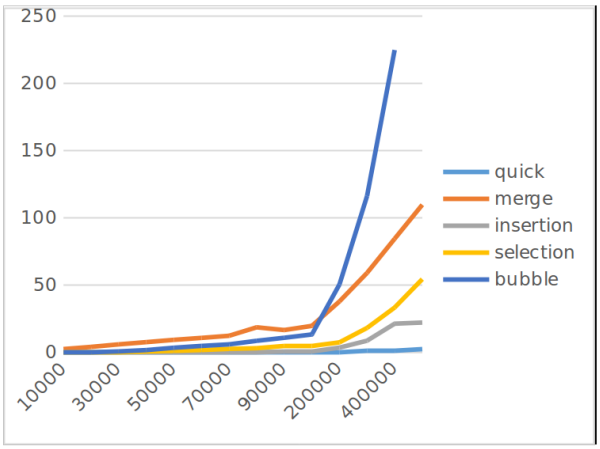

FIG 3: GRAPH PLOT FOR TABLE 3 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

TABLE 4

ASYMPTOTIC TIME TAKEN ON TEST BED 2 USING JAVA CODE FOR RANDOM DATA SAMPLES

Sample size	**Quick Sort Time taken**	**Merge sort Time in sec**	**Insertion Sort Time in sec**	**Selection sort Time in sec**	**Bubble sort Timein sed**
10000	**0.008**	1.968	0.033	0.120	.0.273
20000	0.007	3.411	0.089	0.428	1.199
30000	0.013	5.029	0.209	6.47	2.364
40000	0.019	6.789	0.355	1.124	5.216
50000	0.029	8.425	0.556	1.859	6,572
60000	0.045	10.032	0.831	2.548	9.482
70000	0.061	11.817	1.133	3.511	12.800
80000	0.079	13.584	1.806	4.621	16.842
90000	0.100	15.080	1.829	5.985	21.250
100000	**0.1122**	17.651	2.297	7.299	26.347
200000	0.471	35.600	9.221	32.103	110.425
300000	1.075	52.400	20.789	65.924	241.174
400000	**1.881**	69.616	37.971	115.139	427.947

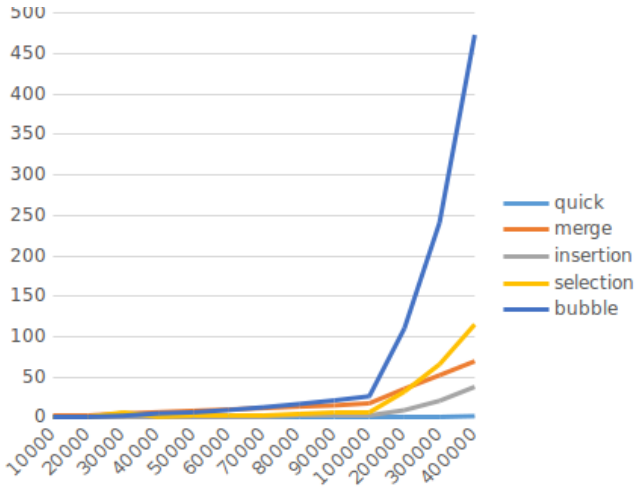

Fig 4 GRAPH PLOT FOR TABLE 4 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

ii. Experimenting with Sorted data set

As mentioned in the experiment set-up section, we have mentioned how to generate sorted data set. Sorted data set is considered for analyzing the worst case behaviors especially for the nlogn class of sorting algorithms which are very much faster ie., the Quick sort and Merge sort in case of random data samples. Table 5 and Table 6 along with their graph plots in Figure 5 and Figure 6 respectively shows the results of experiments done on sorted numbers using Python and Java languages The time clocked nano and millis later converted to second(sec).

TABLE 5
ASYMPTOTIC TIME TAKEN ON TEST BED 2 FOR PYTHON CODE ON SORTED DATA SAMPLES

Sample size	Quick Sort Time in sec	Merge Sort Time in sec	Insertion sort Time in sec	Selection sort Time in sec	Bubble sort Time in sec
10000	5.47	**0.04**	6.83	3.38	4.10
20000	21.97	0.15	27.00	13.78	24.36
30000	49.89	0.33	63.06	30.75	39.98
40000	89.03	0.53	129.82	54.94	99.15
50000	141.23	0.82	182.84	87.71	114.12
60000	209.51	1.16	259.94	130.88	220.95
70000	293.68	1.57	378.23	178.36	286.96
80000	429.04	2.06	461.70	238.01	337.61
90000	536.65	2.61	582.57	300.06	405.04
100000	598.79	**1.505**	684.93	375.28	460.40
200000	2590.36	1.668	2109.76	1447.73	1751.17 (29 minutes)
300000	5858.01	3.531	6167.30	3777.81	5862.24 (97 minutes)
400000	10255.04	6.083	10991.7	6090.13	8104.421 (135 minutes)
500000	15713.08	**9.410**	20052.8	9125.70	**11748.8** (195 minutes)

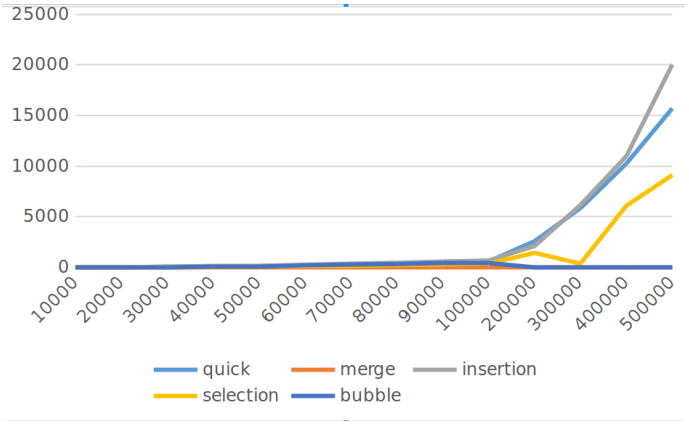

FIG 5: GRAPH PLOT FOR TABLE 5 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

TABLE 6
ASYMPTOTIC TIME TAKEN ON TEST BED 2 FOR JAVA CODE ON SORTED DATA SAMPLES

Sample Size	Quick sort Time in sec	Merge sort Time in sec	Insertion sort Time in sec	Selection sort Time in sec	Bubble sort Time in sec
10000	0.87	25.25	**0.54**	**0.32**	**0.47**
20000	3.09	33.48	1.84	0.92	1.56
30000	7.07	51.26	3.98	2.37	2.93
40000	12.54	67.51	7.14	3.67	6.19
50000	16.26	84.46	11.24	5.70	9.49
60000	25.56	102.03	16.02	8.25	13.85
70000	34.56	119.61	21.99	10.96	18.87
80000	45.05	137.71	28.88	14.88	24.54
90000	57.04	153.69	36.37	18.40	31.16
100000	70.44	169.90	44.90	**25.86**	32.58
200000	283.43	337.16	181.17	104.96	131.53
300000	650.18	530.29	415.99	236.41	295.66
400000	1142.17	696.76	749.28	429.95	529.66
500000	1812.31	**875.73**	1167.27	670.42	848.90

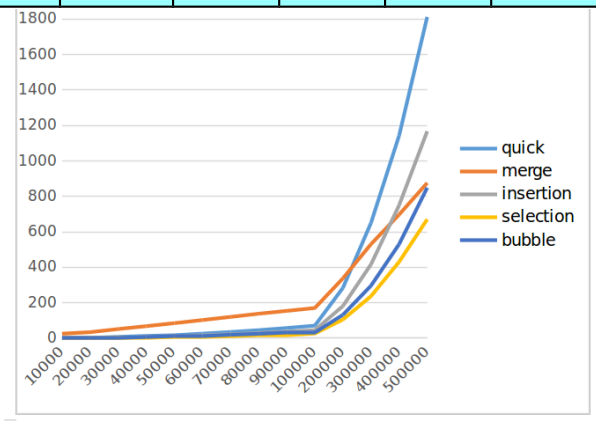

FIG 6: GRAPH PLOT FOR TABLE 6 VALUES, X-AXIS REPRESENTS SAMPLE SIZE AND Y-AXIS SHOWS THE TIME TAKEN IN SECONDS

VI. Conclusion

Our research shows that **Merge sort is fastest among all the popular sorting algorithm considered for both**

Random and Sorted data sets samples for Python language. In Java, Merge sort gets slower in case of large sorted data sets than the n^2 class of sorting algorithms but still performs better than Quick sort. In Python, Quick sort performs better than n^2 class and is very much slower than merge sort for random data set. **In Java, Quick sort tops among others and shows its excellent performance for random large data sets**. Quick sort get worst in large samples of sorted data sets becoming the slowest among all popular sorting algorithms in both Python and Java which is natural because of it worst case complexity. Merge sort gets slower for very large sorted data sample(400000) in Java implementation. As we know that the sorted large data is fever in practice and we normally come across random data set in the real world context, hence we conclude that Merge sort out performs for Python and Quick sort out performs for Java. Also this observation opens are area of research with respect to compiler design for sorting and searching data as we see that the performance degrade of Quick sort seems to be because of the Python Compiler.

ACKNOWLEDGEMENT

We are thankful to almighty Allah who made this successful, also we are thankful to Mr Ejaz Ahmed sir our promoter, the department staff for encouragement and our families who provided opportunity to get devoted by keeping us undisturbed at least for few weeks which is inexpressible. Also, we want to thanks students of fourth semester 2022 batch of our department for java language experiments supports also the students of Ghousia womens polytechnic who helped in Python programming executions, finally our department head Dr. Dilshad Begum and our beloved Principal for their supports. Hence made this practically implemented and get published.

REFERENCE

[1] Sartaj Sahni, “Data structures and Algorithms in C++”, university press publication 2004, chapter 4: performance measurement, pages 123-136.

[2] Levitin, A. Introducgtion to the Design and Analysis of Algorithms. Addison-Wesley, Boston MA, 2007.

[3] Thomas H. Cormen,Charles E. Leiserson,Ronald L. Rivest,Clifford Stein, “Introduction to Algorithms”,Second Edition,Prentice-Hall New Delhi,2004.

[4] Vandana Sharma, Parvinder S. Sandhu, Satwinder Singh, and Baljit Saini,” Analysis of Modified Heap Sort Algorithm on Different Environment", World Academy of Science, Engineering and Technology 42 2008.

[5]Gina Soileau, Muhammad Younus, Suresh Nandlall,Tamiko Jenkins, Thierry Ngouolali, Tom Rivers,”Sorting Algorithm Analysis”, (SMT-274304-01-08FA1), Professor James Iannibelli, December 21, 2008.

[6]james Gallagher, “Python maximum hrecursion depth exceeded in comparison solution”. Available: http://www.careerkarma.com

[7]Mike Mc Millan, “comparing Programming language efficiency in 4 programming languages”, Available:http//www.gitconnected.com/levelup

[8]Omar Khan Durrani, Shreelakshmi V, Sushma Shetty & Vinutha D C “ Analysis and Determination of Asymptotic Behavior Range For Popular Sorting Algorithms”Special Issue of International Journal of Computer Science & Informatics , ISSN (PRINT) :2231– 5292, Vol.- II, Issue-1, 2.

[9]C.Canaan, M.S Garai, M Daya,”Popular Sorting Algorithms”,World Applied Programming, Vol 1,No. 1,April 2011,pages 42-50.

[10]Weiss, M. A., “Data Structures and Algorithm Analysis in C”,Addison-Wesley,Second Edition, 1997 ISBN: 0-201-49840-5.

[11]Online Python/Java compiler(interpreter), Available at : https://www.programiz.com/

[12]You Yang, Ping Yu and Yan Gan, "Experimental study on the five sort algorithms," 2011 Second International Conference on Mechanic Automation and Control Engineering, 2011, pp. 1314-1317, doi: 10.1109/MACE.2011.5987184.

[13]M. Marcellino, D. W. Pratama, S. S. Suntiarko and K. Margi, "Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage," 2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI), 2021, pp. 154-160, doi: 10.1109/ICCSAI53272.2021.9609715.

[14]D. Mittermair and P. Puschner, "Which sorting algorithms to choose for hard real-time applications," Proceedings Ninth Euromicro Workshop on Real Time Systems, 1997, pp. 250-257, doi: 10.1109/EMWRTS.1997.613792.