DAA PROJECT REPORT

By Heer Patel and Rohan Shetty

**Introduction**

This is written report which is a summary based on our analysis of 5 sorting algorithm for data of various sizes, wherein we have compared the time taken by each algorithm to sort the array at various sizes and with different random values. In this project we have taken randomized values in array to compare runtimes of the 5 sorting algorithms, i.e.: Insertion sort, Quick sort, Merge Sort, Radix Sort and Heap Sort.

In the below sections we have summarized how each sorting algorithm performs under various conditions and what are the best and worst analysis cases for the same.

Comparisons of algorithm depends on various parameters. Here, we have mapped with respect to time and size of array.

Running time comparison based on data sizes

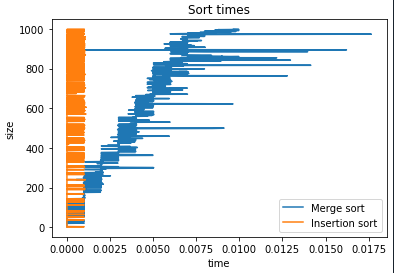
* Insertion sort: Insertion sort has the smallest runtime for very small datasets since it has the best time complexity of Ω(n) since it is more stable as compared to quicksort and requires lesser memory. Insertion sort does not have good running time for larger datasets, since it takes the most CPU time among the algorithms to compile data.
* Quick sort: Quick sort does not have good running time in our case for large datasets, since it is not a stable algorithm and the pivot keeps on switching, leading it to having its worst time complexity O(n^2) for bigger datasets. Since Quicksort is used at its worst time complexity it does not have good running time for smaller datasets.
* Heap sort: Heap sort has one of the best runtime when it comes to large datasets, since it has the best worst running time case O(nlog(n)). Heap sort does not have the best runtime as compared to radix sort and insertion sort when it comes to smaller datasets.
* Radix Sort: It works really well with large datasets with large numbers since it compares the integers by its base and it has low computational complexity as compared to comparison sorting algorithms when the length of keys is smaller as compared to database sizes. In our case Radix sort works well with smaller datasets as well
* Merge Sort: It is not good for smaller datasets, because it is not an in-place sorting algorithm and keeps creating spaces for storage. However, it is preferred for large datasets since it keeps breaking the array and has a good worst time complexity of O(n log(n)) but not better than radix sort when compared for time complexity.

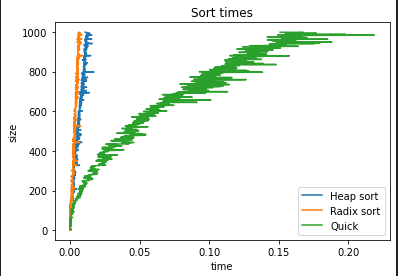
**Setup**

We have implemented this project in python. We have generated an array from 0 to 1000 with random values. Value will be the same in all the sorting algorithms for each array in an iteration and graphs will plotted accordingly.

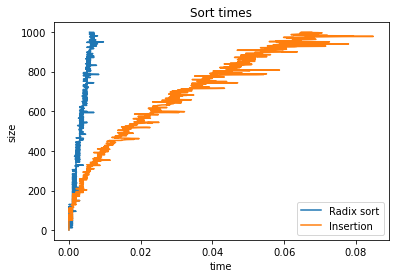
**Analysis**

* Best Case Analysis
  + In the below analysis we can see that Insertion sort and Radix sort both have the best time complexities hence, we compare it further.

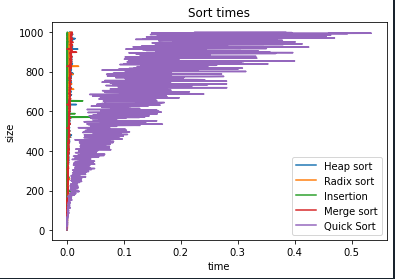




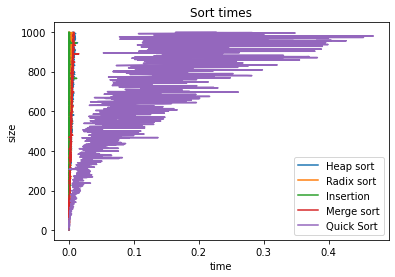
* In our comparison of Radix and Insertion sort, we found out that **Radix sort**  **performs better as compared to Insertion Sort.** For exceedingly small data size insertion sort performs better.



* Worst Case Analysis:
  + In this we are comparing graphs with 2 sets of values in order to determine worst case analysis.

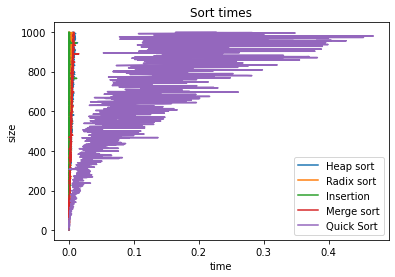


* **In both the cases we found out that quick sort has the worst time complexity for small and bigger datasets**



Speed comparison with respect to different data sizes

* Radix sort performs well with large datasets as well as average datasets. Insertion sort is favourable for smaller data sets. Merge sort and heap sort are working well with mainly larger datasets. Quicksort in our case has the worst time complexity for smaller and larger datasets.



* Ways to improve running time using our discovery
  + Quick sort must be avoided in most of the cases
  + Radix Sort must be preferred for larger and smaller dataset
* Which algorithm is preferred under which condition
  + Algorithms like Merge Sort, Heap Sort and Radix Sort are more favourable for larger datasets.
  + Algorithms like Insertion Sort and Radix Sort are favourable for smaller datasets.

**Conclusion**

* The results that we have fetched are in accordance to our theory. The properties of the 5 sorting algorithms (i.e.: Insertion sort, Quick sort, Merge Sort, Radix Sort and Heap Sort) are quantitatively confirmed.
* According to our analysis, we found out Radix sort to be super-efficient, since it requires the least time to execute
* Even in the case of smaller dataset, Radix sort turned out to be the efficient along with Heap sort
* Quick sort on the other hand had the worst time complexity in all the cases.

**Reference:** <https://www.cse.unr.edu/~chen_q/sorta.html> , https://www.geeksforgeeks.org/fundamentals-of-algorithms

**Work Done:**

Heer Patel: Insertion sort, Radix Sort, Heap Sort, Merge Sort

Rohan Shetty: Quick Sort, PPT and Report.