|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Size | Insertion Sort  (Time in seconds) | Bubble Sort  (Time in seconds) | Merge Sort  (Time in seconds) | Quick Sort  (Time in seconds) | Radix Sort  (Time in seconds) |
| 10,000 | 0.0328569 | 0.1203266 | 0.0027974 | 0.0022874 | 0.0052436 |
|  | 0.033455 | 0.119988 | 0.0028463 | 0.0024143 | 0.0052094 |
|  | 0.0358711 | 0.1177524 | 0.0034836 | 0.0021496 | 0.0054379 |
|  | 0.0322585 | 0.1210976 | 0.0031882 | 0.0025837 | 0.0057163 |
|  | 0.0322618 | 0.1231613 | 0.0033102 | 0.0024426 | 0.005337 |
| Average Time | 0.0328579 | 0.120470733 | 0.0031149 | 0.002381433 | 0.0053395 |
|  |  |  |  |  |  |
| 50,000 | 0.7298306 | 3.5899824 | 0.0097063 | 0.0080252 | 0.0147311 |
|  | 0.7282887 | 3.5536703 | 0.0100561 | 0.0168278 | 0.0147959 |
|  | 0.6981836 | 3.5637395 | 0.0096208 | 0.0163819 | 0.0144842 |
|  | 0.7305575 | 3.5666213 | 0.0096832 | 0.0078213 | 0.0148591 |
|  | 0.7279022 | 3.5521401 | 0.0095867 | 0.0164114 | 0.0147702 |
| Average Time | 0.728673833 | 3.5613437 | 0.0096701 | 0.013606167 | 0.014765733 |
|  |  |  |  |  |  |
| 100,000 | 2.8567309 | 14.4057549 | 0.0176875 | 0.0151099 | 0.0237495 |
|  | 2.8611471 | 14.4156418 | 0.0184332 | 0.0162702 | 0.0240094 |
|  | 2.8583011 | 14.4082673 | 0.0176616 | 0.0198676 | 0.0239415 |
|  | 2.8374936 | 14.3677997 | 0.0179223 | 0.0162023 | 0.0239632 |
|  | 2.8476235 | 14.3542801 | 0.0177871 | 0.0150387 | 0.0241843 |
| Average Time | 2.8542185 | 14.39394063 | 0.017798967 | 0.0158608 | 0.023971367 |
|  |  |  |  |  |  |
| 500,000 | 69.4968942 | 351.4185111 | 0.0930707 | 0.0567325 | 0.0891255 |
|  | 69.6159754 | 352.3399469 | 0.0942607 | 0.0624209 | 0.0884414 |
|  | 69.4265965 | 351.770747 | 0.0934921 | 0.0564626 | 0.0891892 |
|  | 70.2532064 | 351.1506802 | 0.0931639 | 0.0666425 | 0.0884566 |
|  | 69.7821333 | 350.9506921 | 0.0924669 | 0.0674175 | 0.0886775 |
| Average Time | 69.63166763 | 351.4466461 | 0.093242233 | 0.061931967 | 0.0887532 |
|  |  |  |  |  |  |
| 1,000,000 | 274.5388862 | <10 minutes | 0.2070853 | 0.0949256 | 0.1700831 |
|  | 271.6106498 | <10 minutes | 0.1791179 | 0.1031674 | 0.1703486 |
|  | 275.6121668 | <10 minutes | 0.1776121 | 0.112762 | 0.1703243 |
|  | 274.3116183 | <10 minutes | 0.1771895 | 0.1026902 | 0.1705993 |
|  | 276.5080227 | <10 minutes | 0.1785969 | 0.1172272 | 0.1717611 |
| Average Time | 274.8208887 | <10 minutes | 0.1784423 | 0.106206533 | .170424067 |
|  |  |  |  |  |  |
| 5,000,000 | <10 minutes | <10 minutes | 0.9138452 | 0.5183647 | 0.7786851 |
|  | <10 minutes | <10 minutes | 0.9079864 | 0.5052695 | 0.7729648 |
|  | <10 minutes | <10 minutes | 0.9084815 | 0.4897149 | 0.7682839 |
|  | <10 minutes | <10 minutes | 0.9199403 | 0.4893939 | 0.7761937 |
|  | <10 minutes | <10 minutes | 0.9505255 | 0.4960375 | 0.7713068 |
| Average Time | <10 minutes | <10 minutes | 0.914089 | 0.4970073 | 0.773488433 |

(Table of running times for sorting algorithms. All average times exclude the highest and lowest times from each respective sort. NOTE: >10 minutes indicates that the run time was longer than ten minutes.)

Chart showing a visual representation of running times for all the sorting algorithms.

(Time for insertion sort of data set size 5,000,000 not included as it exceeded ten minutes. Time for bubble sort of data set size 1,000,000 and 5,000,000 not included as it exceeded ten minutes).

Summarization of the findings:

Looking at the time it takes for each sorting algorithm to run we find the following results. First, we can group the algorithms broadly into slow algorithms and fast algorithms.

The slow algorithms are very clearly insertion sort and bubble sort. For very small data sets or sorting that does not need to be done multiple times these two algorithms have execution times that may seem negligible to the end user. But as we move to larger data sets insertion sort’s and bubble sort’s running time grow rapidly. These two algorithms run time becomes a burden and are not viable to use where lots of sorting or sorting on large data sets is required. This is very clear when we get to large data sets (such as 1,000,000 and 5,000,000) where sorting can take several hours to complete.

The next group of algorithms, that can be considered the fast algorithms, are merge sort, quick sort, and radix sort. All these algorithms maintain a lower execution time and do not grow dramatically when the amount of data is increased greatly. But we can compare these three algorithms to one another. The pattern we see emerge is as follows. Merge sort and quick sort stay very close in comparable running time. With quick sort generally outperforming the other with the varying data set sizes and a sort of battle going on for best run time. However, once the data set has reached a very large size (such as the 5,000,000 entries size) we see that quicksort is almost twice as fast. This is likely due to where the sorting is taking place. I.e. quicksort is sorting in place where-as merge sort is not. While merge sort is more stable than quicksort. Quicksort can be adjusted to be more stable and operate quicker from sorting in place. Where-as merge sort requires more overhead, with having to create arrays for its divide and conquer strategy causing a slower runtime. We see this happening in the machine environment (personal computers) we are using. Radix sort compared to these two is consistently slower and can be attributed to the data set values themselves we are sorting as larger values will require more computation due to the nature of radix sort. However, radix sort can still be considered viable when comparing to the use of insertion sort or merge sort.

So, what we have found is that insertion sort and bubble sort operate tremendously slower than that of the other three sorting algorithms. Even on smaller data sets (such as the 10,000 and 50,000 data sets) the runtime is extremely higher compared to the other sorting algorithms. We can also gather that merge sort, quick sort, and radix sort are all viable options. Ultimately our choice comes down to the factors affecting the data set we are sorting and how many times we are sorting. We see that larger data sets can grossly affect the run time of the sorting algorithms and if we are sorting several times, even if the runtime is negligible, the slower performance will compound and can create an efficiency issue.