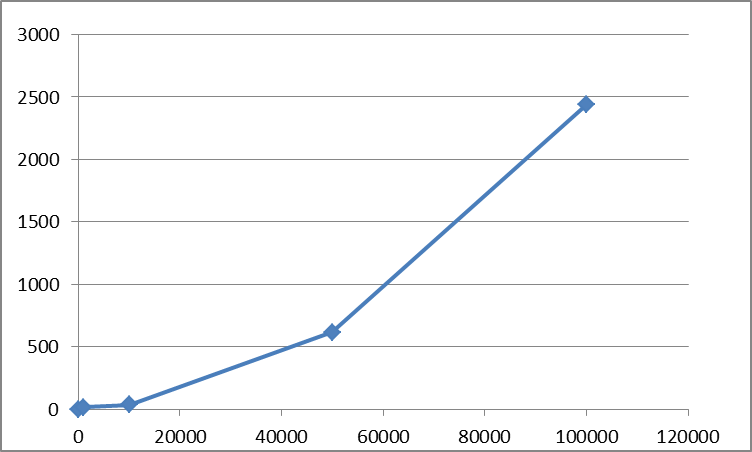
1. Prepare a table (for each sort) showing the runtime, number of comparisons, and number of movements for an array of size 128 and with separate results for each choice of how the elements are initially ordered. The full answer for this question is just one table per sorting implementation, no explanation required.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **Random** | **Inorder** | **ReverseOrder** | **Almost inorder** |
|  | **time** | 6.146306 | 1.602697 | 1.895852 | 1.567881 |
| **Radix** | **comparisons** | 512 | 512 | 512 | 512 |
|  | **copies** | 1024 | 1024 | 1024 | 1024 |
|  |  |  |  |  |  |
|  | **time** | 0.128145 | 0.662378 | 0.352839 | 0.260387 |
| **Quick Sort** | **comparisons** | 1228 | 6131 | 6179 | 1681 |
|  | **copies** | 1125 | 690 | 708 | 750 |
|  |  |  |  |  |  |
|  | **time** | 0.337041 | 0.052955 | 0.026917 | 0.238151 |
| **Insertion Sort** | **comparisons** | 4146 | 0 | 0 | 1190 |
|  | **copies** | 4400 | 254 | 254 | 1444 |
|  |  |  |  |  |  |
|  | **time** | 0.473671 | 1.36162 | 0.899067 | 0.924521 |
| **SelectionSort** | **comparisons** | 8128 | 8128 | 8128 | 8128 |
|  | **copies** | 381 | 381 | 381 | 381 |
|  |  |  |  |  |  |
|  | **time** | 0.160621 | 0.496199 | 0.173787 | 0.30515 |
| **MergeSort** | **comparisons** | 737 | 448 | 448 | 640 |
|  | **copies** | 1713 | 1344 | 1344 | 1560 |
|  |  |  |  |  |  |
|  | **time** | 0.170275 | 0.183441 | 0.236689 | 0.374196 |
| **Heap Sort** | **comparisons** | 1458 | 1454 | 1460 | 1458 |
|  | **copies** | 1110 | 1109 | 1110 | 1111 |

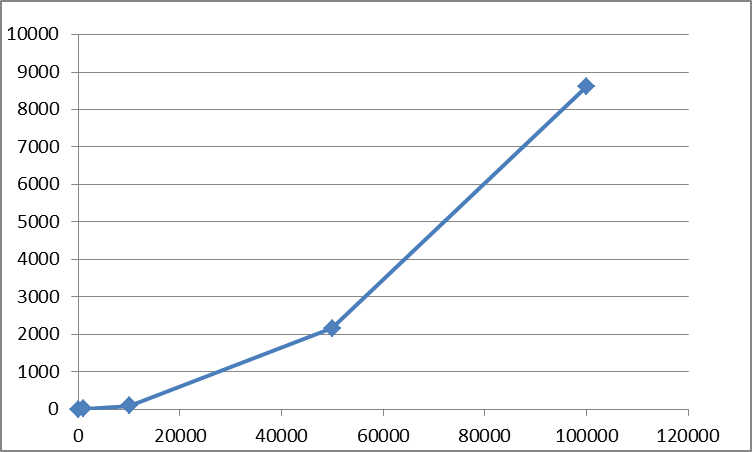
1. Prepare another table (for each sort) showing (just) the runtime of the sort for at least four different non-trivial array sizes. A non-trivial array size is one where the runtime is more than just a few milliseconds. When you can, increase the input size until the runtime takes at least one second. Input sizes will allow you to see enough variation in the statistics to give you strong evidence to support your answers to questions (3) below. You do not need to use the same input sizes for every algorithm, since this might not produce the most useful data. The full answer for this question is just one table per sorting implementation, no explanation required.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Size** |  |  |  |  |  |  |
|  | 100 | 1000 | 10000 | 50000 | 100000 | 500000 | 1000000 | 5000000 | 7000000 | 10000000 |
| **Radix Sort** | 1.386 | 9.50237 | 20.6212 | 39.4911 | 62.2457 | 397.58 | 1051.8 | 10415.2 | 17661 | >1sec |
|  |  |  |  |  |  |  |  |  |  |  |
| **Quick Sort** | 0.204 | 2.63401 | 3.37128 | 8.82478 | 16.1703 | 93.5787 | 102.785 | 521.167 | 762.546 | 1131.195 |
|  |  |  |  |  |  |  |  |  |  |  |
| **Insertion Sort** | 0.358 | 15.1577 | 35.1602 | 617.337 | 2436.46 | >1sec | >1sec | >1sec | >1sec | >1sec |
|  |  |  |  |  |  |  |  |  |  |  |
| **Selection Sort** | 0.762 | 11.103 | 88.4462 | 2155.8 | 8605.75 | >1sec | >1sec | >1sec | >1sec | >1sec |
|  |  |  |  |  |  |  |  |  |  |  |
| **Merge Sort** | 0.388 | 1.7955 | 2.33324 | 8.57347 | 13.9696 | 76.2223 | 124.383 | 660.462 | 977.778 | 1387.662 |
|  |  |  |  |  |  |  |  |  |  |  |
| **Heap Sort** | 0.311 | 1.66355 | 8.06732 | 23.4603 | 24.5203 | 56.5438 | 86.2235 | 440.79 | 594.893 | 853.0519 |

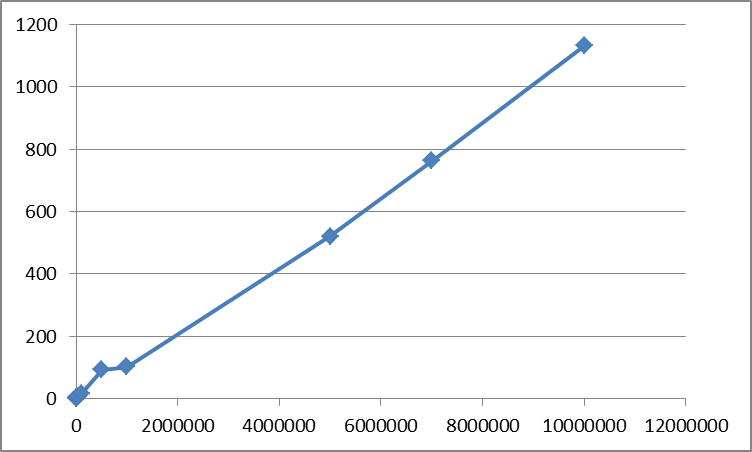
1. **Insertion sort graph O (n2)**



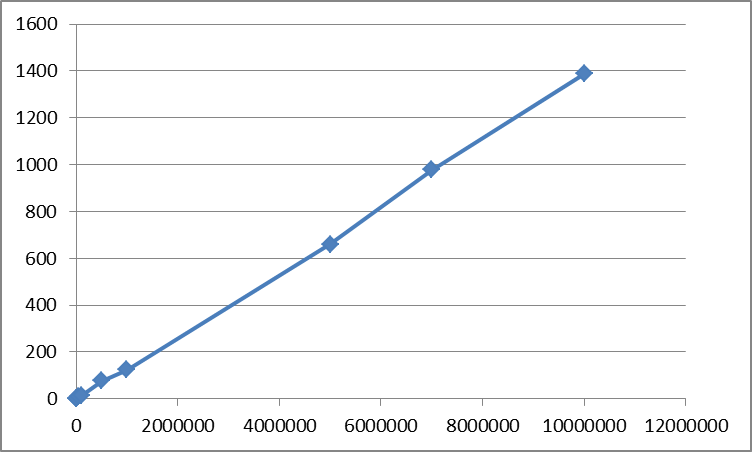
**Selection sort graph O (n2)**



**Quick sort n log n**



**Merge Sort n log n**



**Heap Sort n log n**

**Radix Sort O (nk)**