|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Copies/Comparisons/Swaps** | | | | | | |
| **ArrSize** | **Merge Sort** | | | **Shell Sort** | | |
| **Copies** | **Comparisons** | **Swaps** | **Copies** | **Comparisons** | **Swaps** |
| 10000 | 133616 | 120449 | 133616 | 781227 | 781227 | 75243 |
| 15000 | 208616 | 189324 | 208616 | 1693428 | 1693428 | 120243 |
| 20000 | 287232 | 268096 | 287232 | 2961177 | 2961177 | 165243 |
| 25000 | 367232 | 334048 | 367232 | 4551618 | 4551618 | 210243 |
| 30000 | 447232 | 408582 | 447232 | 6480621 | 6480621 | 255719 |
| 35000 | 529464 | 484554 | 529464 | 8807609 | 8807609 | 305719 |
| 40000 | 614464 | 561869 | 614464 | 11431033 | 11431033 | 355719 |
| 45000 | 699464 | 639713 | 699464 | 14409625 | 14409625 | 405719 |
| 50000 | 784464 | 717987 | 784464 | 17825878 | 17825878 | 455719 |

|  |  |  |  |
| --- | --- | --- | --- |
| **ArrSize** | **Quick Sort** | | |
| 10000 | **Copies** | **Comparisons** | **Swaps** |
| 15000 | 30414 | 69486 | 69486 |
| 20000 | 45214 | 102278 | 56826 |
| 25000 | 62651 | 136822 | 79056 |
| 30000 | 80660 | 189282 | 101828 |
| 35000 | 91546 | 214736 | 127196 |
| 40000 | 105702 | 233614 | 153160 |
| 45000 | 124901 | 281231 | 177608 |
| 50000 | 141484 | 307073 | 203696 |
| 10000 | 162520 | 337177 | 229766 |

**Trend analysis:**

**Merge Sort:**

- The number of copies and swaps increase roughly linearly with the array’s size, and stay the same with each other as they increase.

- The number of comparisons also rises up roughly linearly with the array’s size, but with a slower rate comparing to the number of copies and swaps.

**Shell Sort:**

- The number of copies and swaps: same with Merge Sort.

- The number of comparisons also has the same way of increasing, but with a faster rate.

**Quick Sort:**

- The number of copies are less than those of the Merge Sort and Shell Sort, and they increase at a slower rate with the array’s size.

- The number of comparisons made increases linearly with the size of the array, but it's notably smaller compared to the other algorithms.

- Quick Sort requires a significant number of swaps, especially for larger arrays. However, the number of swaps increases at a slower rate compared to the number of comparisons.

This table is in Lab2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **COPIES / COMPARISONS / SWAPS** | | | | | | |
|  | **Bubble Sort** | | **Selection Sort** | | **Insertion Sort** | |
|  | **Comparisons** | **Swaps** | **Comparisons** | **Swaps** | **Comparisons** | **Swaps** |
| 10000 | 49994999 | 24898040 | 49995000 | 9999 | 24991732 | 24991732 |
| 15000 | 112492499 | 55671085 | 112492500 | 14999 | 56046176 | 56046176 |
| 20000 | 199989999 | 100424266 | 199990000 | 19999 | 99681057 | 99681057 |
| 25000 | 312487499 | 156301931 | 312487500 | 24999 | 156054211 | 156054211 |
| 30000 | 449984999 | 225363046 | 449985000 | 29999 | 225707534 | 225707534 |
| 35000 | 612482499 | 306053598 | 612482500 | 34999 | 306381568 | 306381568 |
| 40000 | 799979999 | 395696673 | 799980000 | 39999 | 398883105 | 398883105 |
| 45000 | 1012477499 | 506491922 | 1012477500 | 44999 | 504186538 | 504186538 |
| 50000 | 1249974999 | 625825839 | 1249975000 | 49999 | 627644514 | 627644514 |

Comparison between 2 tables:

Overall, Merge Sort, Shell Sort and Quick Sort seems to have better performance than Bubble Sort, Selection Sort and Insertion Sort.

Specifically:

**Merge Sort, Shell Sort, and Quick Sort:**

* **Copies:** Merge Sort generally requires a moderate to high number of copies, Shell Sort tends to require more copies, while Quick Sort requires fewer copies compared to both Merge Sort and Shell Sort. Quick Sort demonstrates the most efficient performance in terms of copies.
* **Comparisons:** Merge Sort and Quick Sort show relatively similar numbers of comparisons, with a linear increase in comparisons as the array size grows. Shell Sort requires significantly more comparisons compared to Merge Sort and Quick Sort for all array sizes.
* **Swaps:** Merge Sort doesn't involve swaps, Shell Sort requires a moderate number of swaps that increase roughly linearly with the array size, and Quick Sort requires a relatively small number of swaps compared to Shell Sort, increasing at a slower rate.

**Bubble Sort, Selection Sort, and Insertion Sort:**

* **Copies:** These sorting algorithms exhibit significantly higher numbers of copies compared to Merge Sort, Shell Sort, and Quick Sort. Bubble Sort, Selection Sort, and Insertion Sort require a quadratic increase in copies with the array size.
* **Comparisons:** Similar to copies, Bubble Sort, Selection Sort, and Insertion Sort require a quadratic increase in comparisons with the array size. They generally perform many more comparisons compared to Merge Sort, Shell Sort, and Quick Sort.
* **Swaps:** Bubble Sort, Selection Sort, and Insertion Sort involve a significant number of swaps, increasing quadratically with the array size. They require considerably more swaps compared to Merge Sort, Shell Sort, and Quick Sort.

=> **Quick Sort offers a good balance between comparisons, copies, and swaps, making it efficient for a wide range of datasets.** Merge Sort, while requiring more copies, is stable and works well with linked lists. Shell Sort provides a compromise between insertion and merge sorts, with an emphasis on reducing the number of swaps. Bubble Sort, Selection Sort, and Insertion Sort, while simple to implement, are inefficient for larger datasets due to their quadratic time complexity.