Program Structures and Alogorithms

Spring 2023

Assignment 6

Name: Srikanth Nandikonda

NUID: 002737724

Task:

Task is to determine--for sorting algorithms--what is the best predictor of total execution time: comparisons, swaps/copies, hits (array accesses), or something else.

Conclusion:

The relationship between the number of Hits, Copies and Compares and the number of elements is linear on a log-log graph.

For Merge Sort, the time complexity is O (n log n), which means that the number of Compares is the dominant factor in determining the total execution time of the algorithm. The number of Hits, Copies, and Swaps also increase with the number of elements, but they have a smaller impact on the overall time complexity compared to the number of Compares.

Evidence:

Sort algorithm is run for String array with length from 10,000 to 2,56,000 with doubling the size,

10,000 – 20,000 – 40,000 – 80,000 – 1,60,000 – 2,56,000

Average Time For Merge Sort Algo without instrumentation for different combinations like merge sort, merge sort with no copy, merge sort with Insurance and merge sort with both insurance and no copy

Graphical user interface, application

Description automatically generated

Based on the given data, it appears that Merge Sort with Insurance has the best performance in terms of total execution time for sorting arrays of 10,000 and 20,000 elements. However, for larger array sizes, Merge Sort with No Copy and Insurance has the best performance.

It is interesting to note that Merge Sort with No Copy has worse performance than regular Merge Sort for larger array sizes, but when combined with the Insurance optimization, it shows a significant improvement in performance. This suggests that the Insurance optimization is more effective when used in combination with the No Copy optimization.

Overall, the data suggests that the choice of optimization strategy depends on the size of the array being sorted. For smaller arrays, Merge Sort with Insurance appears to be the best choice, while for larger arrays, Merge Sort with No Copy and Insurance is the better option.

Merge sort with instrumentation on for the plain merge sort.

Table

Description automatically generated with low confidence

|  |  |  |  |
| --- | --- | --- | --- |
| Log No Of Elements | Log of Hits | Log of Copies | Log Compares |
| 13.28771238 | 18.90185387 | 17.747144 | 16.8907274 |
| 14.28771238 | 20.01500254 | 18.87267488 | 18.00473619 |
| 15.28771238 | 21.12000684 | 19.9881521 | 19.11041449 |
| 16.28771238 | 22.21798698 | 21.0950673 | 20.20903802 |
| 17.28771238 | 23.30960402 | 22.19460298 | 21.30121763 |
| 17.96578428 | 24.04988535 | 22.9505559 | 22.03699187 |

The relationship between the number of Hits, Copies and Compares and the number of elements is linear on a log-log graph.

For Merge Sort, the time complexity is O (n log n), which means that the number of Compares is the dominant factor in determining the total execution time of the algorithm. The number of Hits, Copies, and Swaps also increase with the number of elements, but they have a smaller impact on the overall time complexity compared to the number of Compares.

Merge Sort with insurance and no copy with instrumentation

Graphical user interface, text, application

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| Log No of Elements | Log of Hits | Log of Copies | Log Compares |
| 13.28771238 | 18.14456365 | 16.87267488 | 16.91447857 |
| 14.28771238 | 19.24094269 | 17.9881521 | 18.02681519 |
| 15.28771238 | 20.33116188 | 19.0950673 | 19.13095761 |
| 16.28771238 | 21.41607126 | 20.19460298 | 20.22812113 |
| 17.28771238 | 22.49628719 | 21.28771238 | 21.31915279 |
| 17.96578428 | 23.22042069 | 22.03891899 | 22.05627553 |

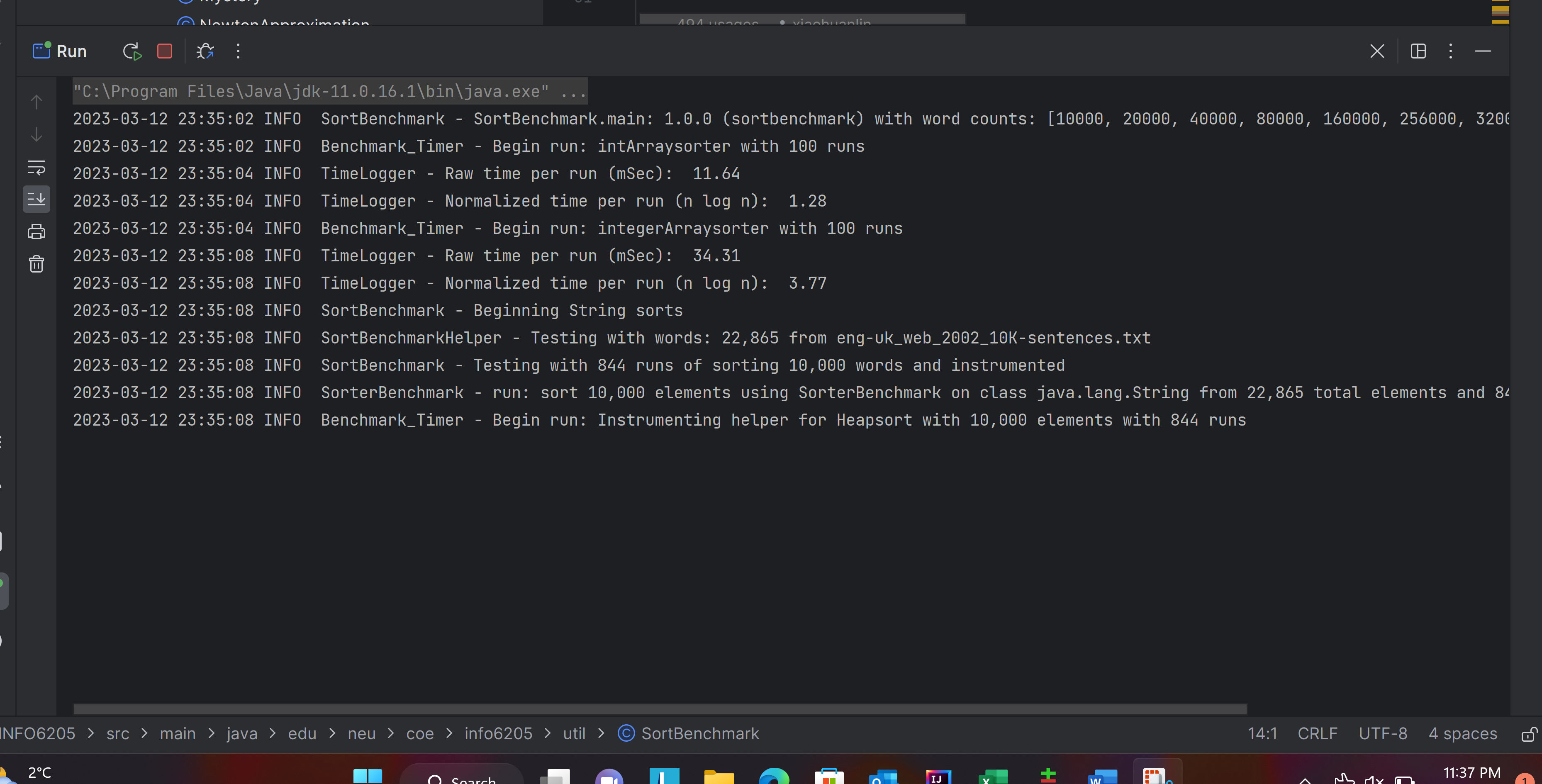
Heap sort and Quick Sort :

Graphical user interface, text, application

Description automatically generated

Based on the data it can be concluded that Merge Sort is the most efficient algorithm for sorting large datasets, followed by Heap Sort. Quick Sort may be a good choice for smaller datasets, but its performance degrades significantly for larger datasets. Merge sort can be further optimized for large data sets by implementing no insurance and no copy optimizations.

Tried running instrumentation for quick sort and heap sort but it took and lot of time and I was not able to procced.



Output :

Merge Sort , Merge Sort with no copy, Merge Sort with Insurance and Merge sort with insurance and no copy – with no Instrumentation

Text

Description automatically generated

A picture containing text

Description automatically generated

Merge Sort and Merge sort with insurance and no copy – Instrumentation

A picture containing background pattern

Description automatically generated

Background pattern

Description automatically generated

A picture containing text, outdoor

Description automatically generated

**Testcase report :**

**Merge sort**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Heap sort**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Quick sort dual pivot test case**

**Text

Description automatically generated**