Computing Engine for Sorting Algorithms

Team achiyu
s1300231 Akari Moriya
s1300252 Yurika Yamao
s1300257 Chinatsu Yokoyama

Contents

- The Product
- Development details
- Contribution
- Demo
- Conclusion

The Product

- Requirements

  Create software that takes a 1-dimensional integer array as input and sorts its elements in ascending order. Design and implement the software to meet the following requirements (your team will receive a reasonable grade for meeting the requirements).

  - Not one, but several sorting algorithms are implemented.
  - Software validity is ensured by data generators and testing engine.
  - The software can be operated not only by CUI but also by GUI (no need to visualize the elements in this stage).
  - The GUI provides easy-to-learn steps of the algorithm by visualizing the movement of elements.
  - While meeting the above requirements, the software is designed to be flexible enough to expand when new sorting algorithms are added.
  - Other innovations are incorporated to make the software a better educational tool (e.g. additional effects for visualization, analysis of complexity of each algorithm, etc.)

Algorithms that we used

- Bubble Sort
- Selection Sort
- Heap Sort
- Quick Sort
- Insert Sort
- Merge Sort
  - These six were implemented because they were considered to be the basic sort.

Development details

- The environment
  - Front-end: vue.js
  - Back-end: node.js
  - Implement algorithm: java

- Why use
  - As we considered this to be a relatively small application, we used vue.js + node.js, which we felt was best suited to represent the GUI we use in other lectures.
  - Java was used to implement the algorithm because all members had used it before.

What can the user do

- Manual input of arrays, random generation
- Perform six different sorting algorithms.
- Analysis of sequences (what sort is suitable)
- Guide to URLs with detailed instructions for sorting.
- Temporal complexity display.

Ex) let numbers = [1, 3, 2, 4, 5];

- 1 → count +1
- 1 ≤ 3 → count +1
- 3 > 2
- 2 ≤ 4 → count +1
- 4 ≤ 5 → count +1

Total count: 4
sortedRatio = 4 / 5 = 0.8

Default: Quick Sort
sortedRatio = 0.9: Insert Sort
Array size less than 10: Selection Sort
Array size more than 30: Merge Sort

Contribution

name	analysis	design	coding	test	Total hour
Moriya Akari	2	0	1	0	3
Yamao Yurika	2	0	1	0	3
Yokoyama Chinatsu	2	2	15	2	21

- Moriya Akari
  - Implement 2 sort algorithm.
- Yamao Yurika
  - Implement 2 sort algorithm.
- Yokoyama Chinatsu
  - Build the development project.
  - Implement 2 sort algorithm, frontend & backend.
  - Design application & Test code. etc

Let’s demonstrate the system!

Conclusion

- Functions we wanted to implement
  - Coloured representation of sorted values.
  - Comparison of sort and another sort movements

- Through development
  - More detailed planning would have enabled the functionality that we wanted to implement.
  - There was too little communication between us.
  - The division of labour was not done well at all and there was a clear bias in the implementation.