READ ME Sort Manager Project

Project Background

First Java Project after learning Java core at Sparta Global. The Sort Manager utilises the Factory design pattern to produce its Sorter objects. All sorters implement a sorters interface that allows all sorters to be called as sorters as opposed to their derived classes.

The program also needed to follow the principles of SOLID and the pillars of OOP. The principles of SOLID are:

1. Single Responsibility Principal – each class should have only one reason for needing modification.
2. Open for extension closed for modification – to add additional functionality, the existing code should not need to be modified you should only have to extend existing code.
3. Liskov Substitution – an instance of the parent class should be interchangeable with instances of the child class.
4. Interface Separation – Interfaces should not force classes that implement them to have dummy methods that are not used.
5. Dependency injection – High level modules should not be affected by changes in low level modules.

The pillars of OOP are:

1. Encapsulation- Limiting what code can be accessed within a class so that each object controls its own state.
2. Abstraction- Hiding the details of an implementation within a class.
3. Polymorphism- being able to perform the same action in different ways.
4. Inheritance – creating child classes that take on the methods and state of a parent class/ interfaces.

Project Requirements

For the starting point of the project there was two interfaces an interface for Sorters and Binary Tree. The final program had to use both interfaces to take an array, output the unsorted array, tell the user the sorting algorithm to be used, the sorted array and then the time taken to complete the sort. The program needed implementations of a bubble sort, a merge sort, and a binary tree sort. The development process also needed to utilise the TDD (Test Driven Development) method.

Tools Utilised

To complete this project the following tools were utilised:

1. IntelliJ was used as the IDE.
2. Maven
3. Log4j
4. Jupiter Junit 5.7 for unit testing and performance testing.

Functionalities

The program can either take a user inputted array of a user assigned length or generate a random array between 3 and 20000 elements long and populate it with random values (maximum 50000 and minimum -50000).

The program can then take a user input for the type of sort desired and will run the sort algorithm and output the unsorted array, the algorithm used, the sorted array and the time taken to complete the sort in milliseconds. In bubble and merge sort the program produces a sorted array with the values in ascending order. The binary tree sorter has the functionality of creating an array in both ascending and descending order.

User Guide

Upon start up the program will ask for a user input that determines the type of sorter that the user wishes to implement by providing a list of options and the corresponding integer to enter for that option. If an integer is entered that does not correspond to one of the offered options, the program will throw an Illegal argument exception that is caught within the starter.

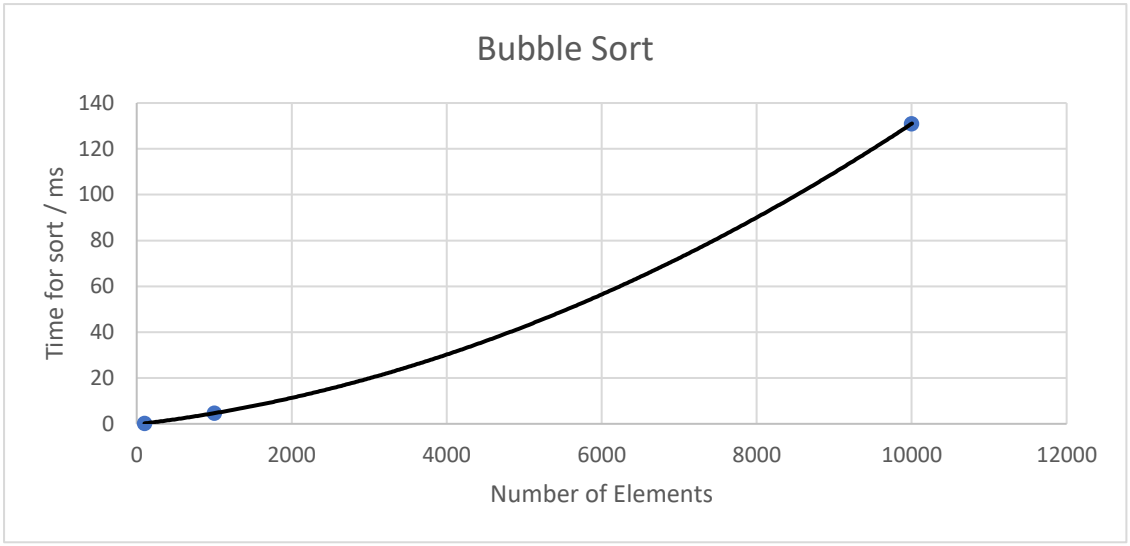
After the sorter type has been determined the program will ask the user to input a desired length of array any positive integer will produce an empty array that the user then populates by entering the elements into the array one at a time. If a value of zero is entered into the length the program will produce a random array to be sorted. If a negative value is entered the program will throw an Illegal argument exception that is caught within the starter.

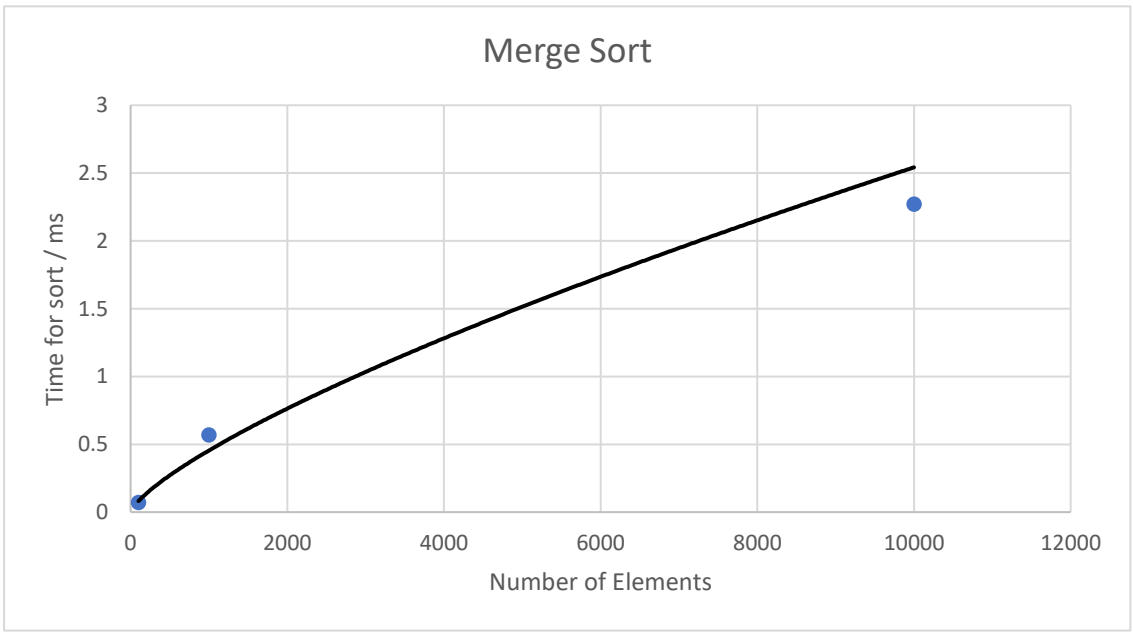
After this the program will check the array to ensure that the array is not already sorted in ascending order before using the bubble sorter or the merge sorter on the array. If the array is pre-sorted it will output, the original array with the message that the array is already sorted.

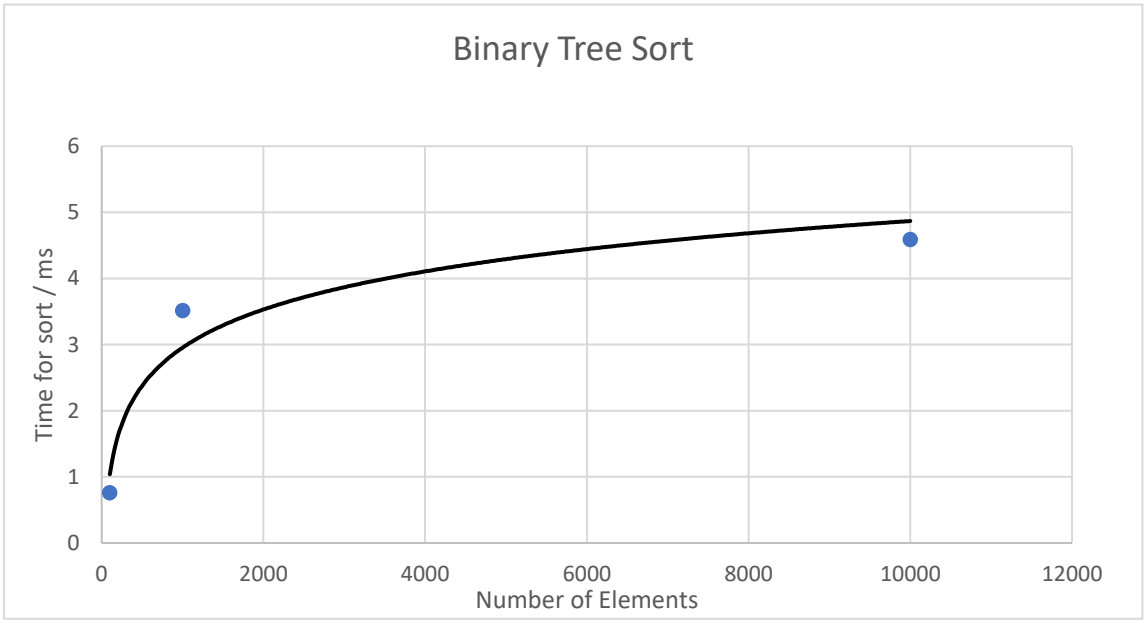
If the array is not pre-sorted the program will then output the sorted array with the time taken when using the bubble sort or merge sort. When using the binary tree sorter, the program will ask the user if they want the array in ascending or descending order using a list detailing the integer to enter for the desired option. Then the program will perform the operation and output the sorted array and the time taken.

Performance Testing

The program was performance tested by entering random arrays into the different sorting algorithms of length 10, 1000 and 10000 elements. The same random array was entered into each of the three different algorithms during a specific pass. This test was run five times and an average time to run produced for each of the algorithms. Because the binary tree cannot store duplicate elements each of the random arrays was made to have each of the elements be of distinct values. Below are some scatter graphs detail the results of the performance testing:


Bubble Sort
Time for sort / ms
140
120
100
80
60
40
20
0
0
2000
4000
6000
8000
10000
12000
Number of Elements


Merge Sort
Time for sort / ms
3
2.5
2
1.5
1
0.5
0
0
2000
4000
6000
8000
10000
12000
Number of Elements


Binary Tree Sort
Time for sort / ms
6
5
4
3
2
1
0
0
2000
4000
6000
8000
10000
12000
Number of Elements