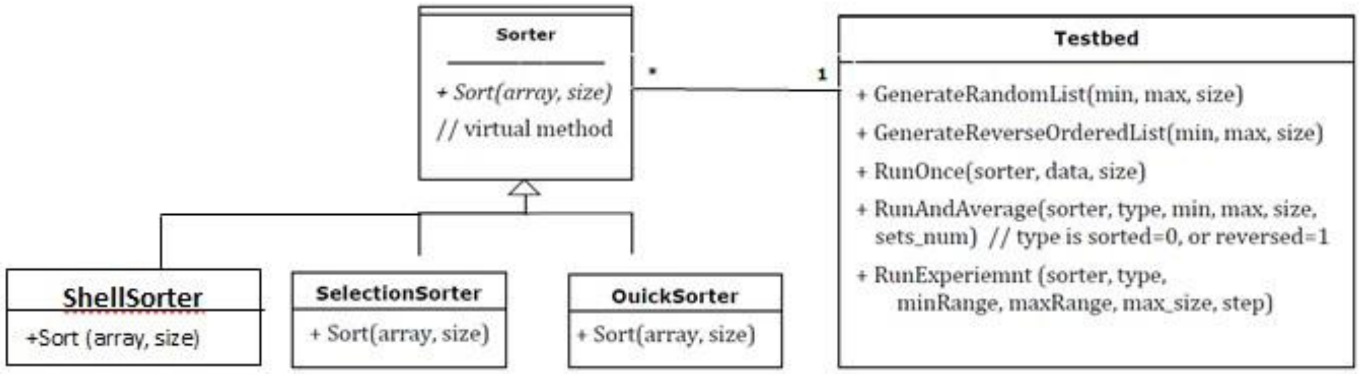
Yusuf Fawzy Elnady

Comparing Sorting Algorithms

Summary:

This program has initially **three classes** for testing three sorting algorithms (**Selection**, **Quick** and **Shell** sort). **Sorter Class** can sort any data type (**Generic**), with the shown methods in the following high-level UML diagram, to support experimenting and analyzing sorting algorithms performance.


Sorter
+ Sort(array, size)
// virtual method
*
1
Testbed
+ GenerateRandomList(min, max, size)
+ GenerateReverseOrderedList(min, max, size)
+ RunOnce(sorter, data, size)
+ RunAndAverage(sorter, type, min, max, size, sets_num) // type is sorted=0, or reversed=1
+ RunExperiemnt (sorter, type, minRange, maxRange, max_size, step)
ShellSorter
+Sort (array, size)
SelectionSorter
+ Sort(array, size)
QuickSorter
+ Sort(array, size)

Testbed Class:

The class is consisted of the following functions:

1. **GenerateRandomList(min, max, size):** Generates a given number of random integer data from a certain range.
2. **GenerateReverseOrderedList(min, max, size):** Generates a given number of reverse ordered integer data from a certain range.
3. **RunOnce(sorter, data, size)**: Runs a given sorting algorithm on a given set of data and calculates the time taken to sort the data.
4. **RunAndAverage(sorter, type, min, max, size, sets_num):** Runs a given sorting algorithm on several sets of data of the same length and same attributes (from the same range and equally sorted; e.g., random or reversed) and calculates the average time.
5. **RunExperient (sorter, type, min, max, min_val, max_val, sets_num, step):** Runs a given sorting algorithm and calculates its performance on sets of different sizes (e.g., data of size 10000, 20000, etc.) as follows: **All sets** are generated with values **between min and max**. First, we generate **sets_num** sets with size **min_val**, using **RunAndAverage** () we record average time to process the sets. Then, we **repeat** the previous step but **with sets** whose **size** increases by step till reaching **max_val**.

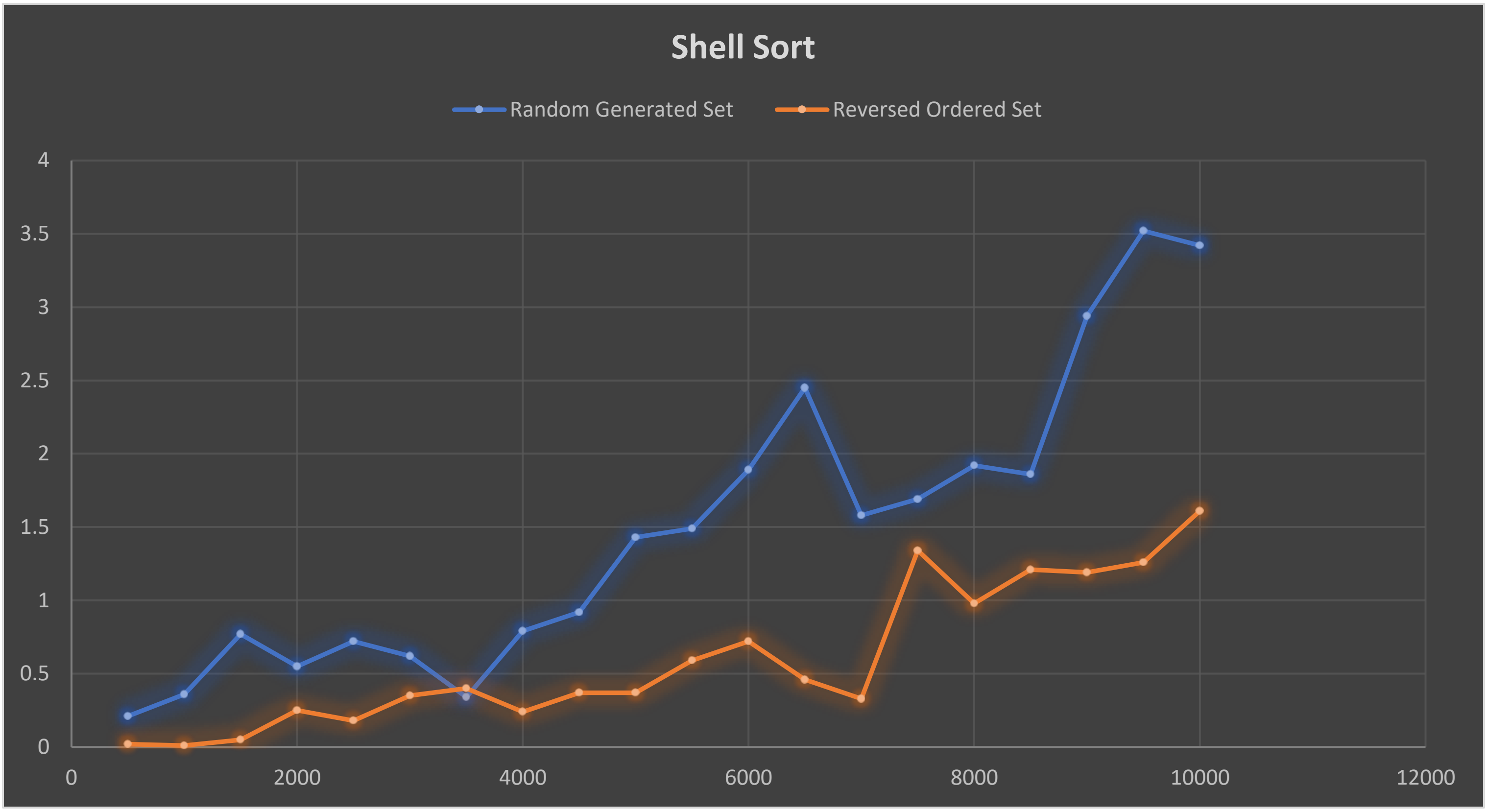
Yusuf Fawzy Elnady

For example, the program can run Quick sort algorithm on **randomly sorted integers** data taken from the **range (1 to 1,100,000)** and with **input value** (data size) from **0 to 100000**, with **step 5000**. This means we will **run the algorithms** on **data sets** of **5000**, **10000**, **15000**, …, **100000** randomly sorted integers, with each step generates **sets_num** different sets and always take the average of their runs.

The results are plotted in the attached **excel file.**

The project is built using **C++, Code::Blocks**.

Results:


Shell Sort
Random Generated Set
Reversed Ordered Set