TOOLS FOR COMPARING THE RESULTS OF THE WORK OF SORTING ALGORITHMS

Larysa Gumieniuk, Vladimir Lotysh, Pavlo Gumieniuk
Lutsk National Technical University, Department of Automation and Computer – Integrated Technologies

Abstract. The program implementation of sorting algorithms is obtained. The program realization of complex for comparison of sorting algorithms is obtained. Using the obtained tools, an analysis of algorithms for sorting by speed was performed depending on the number of members of the data array.

Keywords: sorting algorithms, program realization, a software package

NARZĘDZIA DO POROWNANIA WYNIKÓW PRACY ALGORYTMÓW SORTOWANIA

Streszczenie. Wykonano program realizujący algorytmy sortowania. Otrzymano programowy układ do porównania algorytmów sortowania. Wykrywając otrzymane narzędzia, wykonano analizę algorytmów sortowania według prędkości, w zależności od liczby elementów tablicy danych.

Słowa kluczowe: algorytmy sortowania, realizacja programu, pakiet oprogramowania

Introduction

Often, there is a need to arrange objects based on a single quality: to record number’s data in ascending order, arrange people by their height, arrange words in alphabetical order. If you are able to compare any two items from the given set, then this set can always be arranged. The process of organizing information is called “sorting”.

The volumes of data arrays reach the sizes that decades ago seemed almost unbelievable. The need to organize large amounts of information that is used to effectively implement a real-time search and processing procedure is increasing. The larger the amount of processed data, the more important is the task of optimizing the algorithms used, including sorting.

Thus, the development and research of methods for sorting data arrays, presenting them in a more convenient and formalized form with subsequent implementation is an urgent task at the present stage of development of high-performance computing instruments.

The purpose of this work is to develop a software package for comparing the results of the algorithms of sorting. The creation of a complex includes the development of algorithms and software for comparing the results of the algorithms of sorting.

1. Problem statement

In the development of tools (software complex) the most common algorithms for data sorting have been analysed. Due to the analysis conducted for the program implementation, the following sorting algorithms were selected:

- Built-in sorting algorithm (Python),
- Quicksort (Hoare sorting),
- Merge sort,
- Heapsort (pyramid sort),
- Binary insertion sort,
- Sorting by using simple (linear) inserts,
- Shell sort,
- Sort by choice,
- Bubble sort,
- Threaded sort,
- Bin sort (Bucket sort),
- Integer sort (Radix sort).

For program realization of selected algorithms scripting programming language is being used.

The scripting languages are used by themselves as complete base tool platforms more frequently. For example, many large commercial applications are now programmed mainly in Perl, PHP or Python. Python belongs to a dynamic typing language class, provides the programmer with an automatic "garbage collection" and convenient high-level data structures, such as dictionaries, lists, tuples, etc. Python combines striking power with a simple and understandable syntax, thought-out modularity and scalability.

The Python language interpreter is freely distributed under the Python Software Foundation (PSF) License, which is to some extent even more democratic than the GNU General Public License.

For Python there are libraries for access to the DBMS (on the Windows platform, access to the DB is possible through ADO). There are extension modules for Python under Windows and Unix/Linux for access to Oracle, Sybase, Informix, MySQL and SQLite.

The implementation of tools for comparing results is done in the Delphi programming language.

For temporarily storing data about the speed of program implementation of sorting algorithms, SQLite database is selected. The program is lined up with a library and the engine becomes an integral part of the program.

SQLite stores the entire database (including definitions, tables, indexes, and data) in a single standard file on the computer that is running the program.

Database Management System "SQLite" is a program that is provided under "open source" terms.

The SQLite library itself is written in C and is included in the Python installation application. A number of wrappers and components have been developed to work with Delphi. To implement the Delphi-SQLite connection, the ZeosLIB components have been selected.

ZeosLib is an open source project that supports multiple database management systems for Delphi, FreePascal, Kylix and BCBBuilder: MySQL, PostgreSQL, Interbase, Firebird, MS SQL, Sybase, Oracle, SQLite. ZeosLib uses native DBMS libraries, but can also use its own modified libraries. Usually it's used for configuring and linking components to each other and the host.

The software implementation of the selected algorithms has been carried out in the programming language Python version 3.4.3. SQLite3 database version 3.7.0.1 was used to save data.

To save the data of calculation of the sorting algorithm time to obtain information about the average, median, and fashion, we use the box_plot database table box created using the SQLite3 DBMS.

The table structure is designed to store the data of ten runs of each sorting algorithm with a fixed value of the number of members of the Nb data array.

To analyse the obtained data a software package was developed, which includes: application for calculation (average, median, mode) and visualization of the obtained results; Application for analysis of the received data (regression equation, time) of the sorting process and their 2D and 3D visualization; an application for comparing graphs of sorting time dependence on the number of sorting elements.

Let's take a closer look at each of the applications.
2. Main results

Data Mining application – has the ability to download data from the box’s database, namely the box_plot table, display it in tabular form, calculate the average, median, mode, maximum value and minimum value, and visualize this data in the form of a graph (Figure 1).

The application Chart_m is intended to calculate the total sorting time, creating the sorting time graphical dependence on the number of array elements for this sorting method (2D and 3D), maintaining the obtained dependence into the BMP file, and printing the received results (Figure 2).

The Charts application is designed to construct sorting time dependencies on the number of members of the sorting array for different sorting methods (Figure 3).

For each testing algorithm, a preliminary analysis of how much time algorithms work, depending on the size of the input data, was carried out.

It has been found by the research that all sorting algorithms, except for threaded sort, sorting by choice, sorting by simple inserts, and “bubble” sorting, work fairly quickly. For fast algorithms (built-in sort, integer sort, bucket sort, etc.) testing with incoming data up to 1,500,000 entries was performed; for others (that work slowly) – this limit was up to 110000 entries.

From the analysed data the results of the complexity of each algorithm are obtained.

Table 1 shows the dependence of the working time (sec.) on the number of elements sorted by different algorithms (complexity of the algorithm). The language of implementation is Python.
The following table shows the following sorting algorithms: threaded sort, sort by choice, simple inserts, ‘bubble’ sort, work very long in comparison with others.

The graph of the time dependence of these algorithms on the number of elements sorted is as follows (Figure 4).

From the graph it is noticeable that the algorithm of "threaded sort" is considerably inferior to others (more than 2 times).

Let's consider the running time of other algorithms.

We will start with the algorithm of binary inserts. The graph of the time dependence of this algorithm looks like this (Figure 5).

We describe the resulting curve by the equation of the form \( y = ax^b \). We get \( y = 1.736E-8 \times 1.77365 \). We give the similarly calculated dependencies in the four previous algorithms and in the binary insertion algorithm in Table 2.

![Image of a graph showing the speed of the binary insertion algorithm](image1.png)

**Figure 5. The speed of the binary insertion algorithm**

The binary insertion algorithm works faster than the previous four also because the constant b in this algorithm is smaller.

Let's consider a series of fast algorithms: quicksort, merge sort, Shell sort, pyramid sort (Figure 6).

![Image of a graph showing the speed of various sorting algorithms](image2.png)

**Figure 6. The speed of algorithms: 1 – Quicksort, 2 – Merge sort, 3 – Shell sort, 4 – Pyramid sort**

### Table 1. Dependence of the working time (sec.) on the number of elements sorted by different algorithms

<table>
<thead>
<tr>
<th>Number of sorted elements</th>
<th>Algoalgorithmms name</th>
<th>Build-in sorting</th>
<th>Quicksort</th>
<th>Binary inserts</th>
<th>Merge sort</th>
<th>Shellsort</th>
<th>Pyramid sort</th>
<th>Sort by choice</th>
<th>Simple inserts</th>
<th>Bubble sort</th>
<th>Bacci sort</th>
<th>Sort by grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>0.0083</td>
<td>0.0923</td>
<td>0.277</td>
<td>0.1781</td>
<td>4.471</td>
<td>2.486</td>
<td>0.1166</td>
<td>0.241</td>
<td>13.26</td>
<td>4.317</td>
<td>0.0111</td>
<td>0.0686</td>
</tr>
<tr>
<td>20000</td>
<td>0.0175</td>
<td>0.1987</td>
<td>0.3774</td>
<td>0.3781</td>
<td>25.862</td>
<td>9.901</td>
<td>0.2143</td>
<td>0.461</td>
<td>51.52</td>
<td>4.115</td>
<td>0.0218</td>
<td>0.1325</td>
</tr>
<tr>
<td>30000</td>
<td>0.0235</td>
<td>0.3111</td>
<td>0.5867</td>
<td>0.3871</td>
<td>73.461</td>
<td>22.26</td>
<td>0.3504</td>
<td>0.798</td>
<td>116.8</td>
<td>16.019</td>
<td>0.0324</td>
<td>0.1981</td>
</tr>
<tr>
<td>40000</td>
<td>0.0311</td>
<td>0.4311</td>
<td>0.8901</td>
<td>0.5871</td>
<td>158.671</td>
<td>40.14</td>
<td>0.4584</td>
<td>1.077</td>
<td>213.3</td>
<td>15.415</td>
<td>0.0446</td>
<td>0.2664</td>
</tr>
<tr>
<td>50000</td>
<td>0.0412</td>
<td>0.5623</td>
<td>1.0216</td>
<td>0.7989</td>
<td>292.989</td>
<td>65.65</td>
<td>0.578</td>
<td>1.242</td>
<td>343.9</td>
<td>63.397</td>
<td>0.0571</td>
<td>0.3355</td>
</tr>
<tr>
<td>60000</td>
<td>0.0551</td>
<td>0.6911</td>
<td>1.2466</td>
<td>0.9231</td>
<td>484.829</td>
<td>92.31</td>
<td>0.7266</td>
<td>1.510</td>
<td>512.3</td>
<td>101.59</td>
<td>0.0687</td>
<td>0.4042</td>
</tr>
<tr>
<td>70000</td>
<td>0.0657</td>
<td>0.8177</td>
<td>1.4707</td>
<td>1.7503</td>
<td>750.883</td>
<td>127.39</td>
<td>0.8605</td>
<td>1.794</td>
<td>722.4</td>
<td>146.83</td>
<td>0.0796</td>
<td>0.4759</td>
</tr>
<tr>
<td>80000</td>
<td>0.0661</td>
<td>0.9599</td>
<td>1.7018</td>
<td>1.9075</td>
<td>1087.57</td>
<td>169.20</td>
<td>0.9939</td>
<td>2.073</td>
<td>959.9</td>
<td>206.16</td>
<td>0.0924</td>
<td>0.5482</td>
</tr>
<tr>
<td>90000</td>
<td>0.0769</td>
<td>1.0872</td>
<td>2.0834</td>
<td>2.1939</td>
<td>1502.1</td>
<td>217.09</td>
<td>1.1345</td>
<td>2.356</td>
<td>1231.7</td>
<td>268.33</td>
<td>0.1041</td>
<td>0.6113</td>
</tr>
<tr>
<td>100000</td>
<td>0.0861</td>
<td>1.2197</td>
<td>3.1603</td>
<td>2.6171</td>
<td>2037.53</td>
<td>263.05</td>
<td>1.6266</td>
<td>3.64</td>
<td>1539.4</td>
<td>339.57</td>
<td>0.1164</td>
<td>0.6812</td>
</tr>
<tr>
<td>200000</td>
<td>0.2051</td>
<td>2.6143</td>
<td>4.5241</td>
<td>4.6329</td>
<td>26.27</td>
<td>5.619</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.2509</td>
<td>1.3716</td>
</tr>
<tr>
<td>300000</td>
<td>0.3316</td>
<td>4.0677</td>
<td>9.2377</td>
<td>7.1507</td>
<td>-</td>
<td>-</td>
<td>4.1002</td>
<td>8.710</td>
<td>-</td>
<td>-</td>
<td>0.3860</td>
<td>2.6093</td>
</tr>
<tr>
<td>400000</td>
<td>0.4679</td>
<td>5.5883</td>
<td>12.097</td>
<td>9.0957</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.4515</td>
<td>11.898</td>
<td>-</td>
<td>0.5220</td>
</tr>
<tr>
<td>500000</td>
<td>0.6084</td>
<td>7.1111</td>
<td>13.223</td>
<td>12.223</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.0134</td>
<td>15.255</td>
<td>-</td>
<td>0.6609</td>
</tr>
<tr>
<td>600000</td>
<td>0.7548</td>
<td>8.7311</td>
<td>14.918</td>
<td>14.918</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.4102</td>
<td>18.398</td>
<td>-</td>
<td>0.7958</td>
</tr>
<tr>
<td>700000</td>
<td>0.9002</td>
<td>10.2311</td>
<td>17.786</td>
<td>17.786</td>
<td>-</td>
<td>-</td>
<td>10.236</td>
<td>21.748</td>
<td>-</td>
<td>-</td>
<td>0.9359</td>
<td>4.8021</td>
</tr>
<tr>
<td>800000</td>
<td>1.0577</td>
<td>11.8691</td>
<td>20.516</td>
<td>20.516</td>
<td>-</td>
<td>-</td>
<td>11.723</td>
<td>25.116</td>
<td>-</td>
<td>-</td>
<td>1.0734</td>
<td>5.444</td>
</tr>
</tbody>
</table>

### Table 2. Dependencies of the forms \( y = ax^b \) for sorting algorithms

<table>
<thead>
<tr>
<th>Algorithms name</th>
<th>Analytical equation of the curve</th>
<th>Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>Threaded sorting</td>
<td>( y = ax^2 )</td>
<td>7.478E-11</td>
</tr>
<tr>
<td>Sort by choice</td>
<td>( y = ax^2 )</td>
<td>1.725E-8</td>
</tr>
<tr>
<td>Simple inserts</td>
<td>( y = ax^2 )</td>
<td>5.538E-8</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>( y = ax^2 )</td>
<td>3.358E-8</td>
</tr>
<tr>
<td>Binary inserts</td>
<td>( y = ax^2 )</td>
<td>1.736E-8</td>
</tr>
</tbody>
</table>
Accordingly, the table of coefficients of the equations for the given algorithms is as follows:

Table 3. Dependencies of the forms \( y = ax^b \) for sorting algorithms

<table>
<thead>
<tr>
<th>Algorithms name</th>
<th>Analytical equation of the curve</th>
<th>Coefficients</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort</td>
<td>( y = ax^b )</td>
<td></td>
<td>4.15E-6</td>
<td>1.0756</td>
</tr>
<tr>
<td>Merge sort</td>
<td>( y = ax^b )</td>
<td></td>
<td>8.76E-6</td>
<td>1.07866</td>
</tr>
<tr>
<td>Shell sort</td>
<td>( y = ax^b )</td>
<td></td>
<td>6.28E-6</td>
<td>1.06969</td>
</tr>
<tr>
<td>Pyramid sort</td>
<td>( y = ax^b )</td>
<td></td>
<td>4.12E-6</td>
<td>1.09330</td>
</tr>
</tbody>
</table>

In this group the Pyramid sort Shell sort algorithm was the fastest.

The last considered algorithms:
- Sort by the built-in Python function.
- Bucket sorting.
- Sort by grade (Figure 7).

Accordingly, the table of coefficients of the equations for the given algorithms is as follows:

Table 4. Dependencies of the forms \( y = ax^b \) for sorting algorithms

<table>
<thead>
<tr>
<th>Algorithms name</th>
<th>Analytical equation of the curve</th>
<th>Coefficients</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in Python function</td>
<td>( y = ax^b )</td>
<td></td>
<td>1.392E-7</td>
<td>1.03039</td>
</tr>
<tr>
<td>Bucket sorting</td>
<td>( y = ax^b )</td>
<td></td>
<td>6.609E-7</td>
<td>1.02246</td>
</tr>
<tr>
<td>Sort by grade</td>
<td>( y = ax^b )</td>
<td></td>
<td>5.478E-7</td>
<td>1.05756</td>
</tr>
</tbody>
</table>

3. Summary

The program implementation of sorting algorithms is obtained. The program realization of complex for comparison of sorting algorithms is obtained. Using the obtained tools, an analysis of algorithms for sorting by speed was performed depending on the number of members of the data array.

References


Ph.D. Larissa Gumeniuk
e-mail: l.gumeniuk@lntu.edu.ua

Lutsk National Technical University, PhD (technical), Head of Department of Automation and Computer – Integrated Technologies.
Research interests: Modeling ofreliability and safety ofthe automated control systems.
Has more than 60 publications in this area.

Ph.D. Vladimir Lotysh
e-mail: admin@lntu.edu.ua

Lutsk National Technical University, PhD (technical), Department of Automation and Computer – Integrated Technologies.
Scientific interests include open-source software applied for simulations of problems using distributed platforms.
Author of nearly 80 publications in this area.

Ph.D. Pavlo Gumeniuk
e-mail: p.gumeniuk@lntu.edu.ua

Lutsk National Technical University, PhD (technical), Department of Automation and Computer – Integrated Technologies.
Research interests: programming, robotics.

Otznymo/received: 21.10.2017 Przyjęto do druku/accepted: 11.05.2018