

Interactive Approach to Learning of Sorting Algorithms

<https://doi.org/10.3991/ijoe.v15i08.10530>

Radoslav Mavrevski (✉), Metodi Traykov
South-West University "Neofit Rilski", Blagoevgrad, Bulgaria
radoslav_sm@abv.bg

Ivan Trenchev
South-West University "Neofit Rilski", Blagoevgrad, Bulgaria
University of Library Studies and Information Technologies, Sofia, Bulgaria

Abstract—Today we live in a society of high technologies, advanced information and communication systems in every field, including education. So, in modern education, teachers make full use of the possibilities of modern Information and Communication Technologies (ICT). In this case, the attitude of the teachers towards the use of computers, to achieve the educational goals, is very important. To have the technologies sustained and significant effect, students in secondary and higher schools need to understand how to use them. The goal of this article is to help of students in secondary and higher schools to acquire enough practical programming skills and to learn the sorting algorithms, i.e. the article considers basic sorting algorithms. We developed and describe here software with name “Visual sorting” that shows visual, the execution of the basic sorting algorithms: Bubble sort; Selection sort; Insertion sort; Merge sort. Also, our software provides inter-active tracking of the performance (step by step) of different sorting algorithms.

Keywords—Sorting algorithms, programing, .NET framework

1 Introduction

Today we live in a society of high technologies, advanced information and communication systems in every field, including education. In last years the high technologies, radically changed the people's lifestyles, the businesses and the education in secondary and higher schools [1-4]. As a result from the rapid development of new Information and Communication Technologies (ICT) in education, which enables a two-way connection in real time, the forms of regular, part-time, distance and individual educational are interweaves.

Increasingly, the people have need access to information and if the schools and universities cannot provide it, the people will turn to other institutions. That is why educational institutions should not lag behind technological progress. From this point of view, the education field is becoming in constant consumer of high-tech products.

Today, the access to information is accomplished through computer, using computer networks and software products, which offer many more alternatives than any

textbook. Through ICT the students in secondary and higher schools have access to huge number of information sources (electronic libraries, electronic books and magazines, websites of various institutions and people, etc.). This information stream is two-way at any time of the day – from web space to the students (colleagues, teachers, experts, followers, friends and etc.) in secondary and higher schools and vice versa. So, the attitudes of teachers to the use of computers and Internet to achieve the educational goals are very important, if we want to have successful learning, using the full potential of modern ICT.

Only innovations are not enough. Without the right set of skills, the students in secondary and higher schools will not be able to take advantage of the opportunities that they will face in the future. That is why the global companies, such as Microsoft, are committed to both technology innovation, and education and training. To have the technologies sustained and significant effect, the people should learn how to use them, and this educational process should last a lifetime.

The aim of this article is to help of students in secondary and higher schools to acquire in-depth knowledge and enough practical programming skills for Sorting algorithms and .NET framework. To achieve this aim, using C# programming language, we developed software for visual implementation of some basic sorting algorithms. This software shows step by step sorting process and performance of individual sorting algorithm, which can be very useful in learning and understanding sorting algorithms in education of student in secondary and higher schools.

The C# programming language is an object-oriented language, which is used by programmers around the world to develop applications for .NET platform. This language is built-on the base of the C, C++, Visual Basic (VB) and Java programming languages and provides a complete development environment for applications. The C# programming language combines the power of C, the C++ object-orientation, and VB graphical interface [5-9].

In the sections below, we:

- Make short descriptions of basic sorting algorithms
- Present our software with name “Visual sorting”
- Show the run of “Visual sorting” software

2 Material and Method

Often, when we work with large dataset (the data are from one type), it is necessary to introduce some rules that make it easier to process them. The ordering of the elements could give us a significantly more efficient search algorithm, compared to the case where the data is not sorted.

The sorting is a process of rearranging (permuting appropriately) the dataset elements in a particular order [7]. The sorting is a major activity with a wide range of applications: dictionaries, telephone directories, indexes, and anywhere where is required fast searching of different objects. Depending from the type of the sorted data, the sorting process can be done in a variety of ways.

There are different classifications of the sorting algorithms. Probably the most popular classification is according of data location. Based on this criterion, we can find two types sorting: internally (data is in RAM's computer and often is possible direct access to any element of the dataset) and externally (the data is in external memory (flash drive, Hard Disk Drive and etc.) and often, we access them strictly sequential, starting from the first element). According to the operations, performed over the elements, we have sorting by comparison (usually by using the operators $<$, $>$ and $=$) and sorting by transformation (using arithmetic operations, without direct comparison of pairs of elements). Other important classifications are based on certain properties of the sorting algorithms. Example, persistent and unsustainable. We call one method a persistent method, when in sorting process the relative order of elements with equal values remains unchanged. The persistent methods are preferred, when the dataset elements are already sorted, according to other criterion.

Main requirement to sorting algorithms is minimum costs of additional memory. Also, important requirement is the minimum number of comparisons and exchanges in sorting process. Usually, the sorting process is done by a simple exchange of the locations of two elements in array [7].

The sorting algorithms, those are included in developed by us and described here software “Visual sorting”, are as following: Bubble sort, Selection sort, Insertion sort and Merge sort.

2.1 Bubble sort

Bubble sort has a worst-case and average complexity of $O(n^2)$, where n is the number of elements being sorted. It is recommended for small values of n .

Let a_0, a_1, \dots, a_{n-1} to be a sequence that we want sort in increasing order. The algorithm is as follows:

1. Let the variable $right = n-1$.
2. For the sequence $a_0, a_1, \dots, a_{right}$ we compare sequentially each two adjacent elements a_i and a_{i+1} . If $a_i > a_{i+1}$, we exchange them and keep the exchange position i in variable k . If $a_i \leq a_{i+1}$, we don't change the locations of elements. The process continues until the end of the sequence. If, in this process, after the exchange of i and $i + 1$ elements has not been done other exchanges, therefore the elements $k+1, k+2, \dots, n-1$ are sorted in correct order.
3. $right = k$.
4. If $right > 0$, we go to Step 2 and Step 3.
5. If $right = 0$, the sequence is sorted.

2.2 Selection sort

Selection sort has $O(n^2)$ complexity. The main idea of this method is following: we find the smallest (biggest) element, put it in the beginning of the array, and excluding it from consideration. We repeat these steps for all elements from the sequence that we want to sort [10].

Let $a_i, a_{i+1}, \dots, a_{n-1}, i = 0, 1, \dots, n-2$, to be a sequence that we want to sort in increasing order. The Selection sort algorithm is as follows:

1. We find k -th element, so $a_k = \min \{a_i, a_{i+1}, \dots, a_{n-1}\}$.
2. We exchange the a_k and a_i elements.

2.3 Insertion sort

Insertion sort has $O(n^2)$ complexity. This method implements the following idea. Let we have a dataset $A = \{a_0, a_1, \dots, a_{i-1}\}$. We need to put the element a_i in correct position in the dataset A , so the new dataset $A' = \{a_0, a_1, \dots, a_i, \dots, a_{i-1}\}$ to be sorted in increasing order. We repeat this action for $i = 1, 2, \dots, n-1$.

2.4 Merge sort

In the basis of this sorting algorithm is the "Divide and Conquer" method, i.e. the algorithm divide the input sequence into two parts, sort each of them in increasing/decreasing order and then combine (merges) the obtained sub sequences. The Merge sort algorithm is one of the most effective sorting algorithms. This algorithm has $O(n \log n)$ time complexity.

The source code for the sorting algorithms in C++ and Pascal programming languages are presented in *Appendix*.

2.5 The "Visual Sorting" software

In this article we show developed by us software with name "Visual Sorting". The aim of this software is to make visual representation of performers (step by step) of described above sorting algorithms. We can run our software as desktop application, i.e. we do not need to make installation, just copy and paste the folder with software files on our computer. To develop the "Visual Sorting" software we use C# 2010 Express Edition and MS Visual Studio 2010 Express Edition with Framework 2.0. As we said above, the software contains realizations of described in previous sections sorting algorithms:

- Bubble sort – has $O(n^2)$ complexity
- Selection sort – has $O(n^2)$ complexity
- Insertion sort – has $O(n^2)$ complexity
- Merge sort – has $O(n \log n)$ complexity

For each of these sorting algorithms, our software allows interactive tracking of performance (step by step). For each step of the performance of the chosen algorithm, the array elements that are under consideration are colored in appropriate colors and the software shows message for the action being performed at the respective step from the algorithm execution.

3 Results and Discussion

3.1 Main form of the “Visual Sorting” software

Figure 1 show the main form of the “Visual Sorting” software.

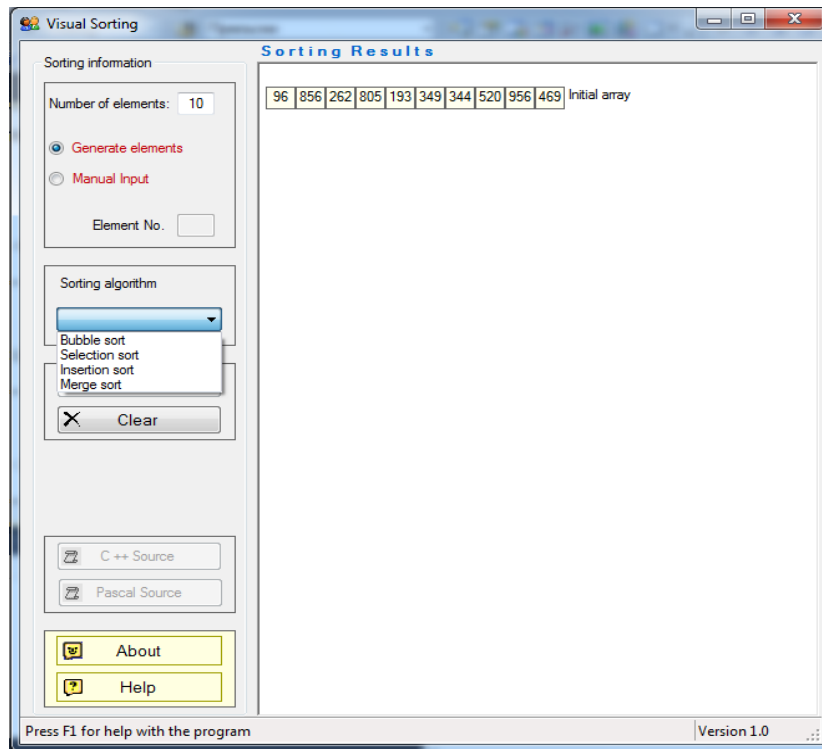


Fig. 1. “Visual Sorting” – Main form

From main form we can select the number of elements (the maximum number is limited to twenty elements) that need to be sorted. If we enter a number large than twenty elements, the software will display a warning message for too many elements. Once we entering the number of elements, the radio buttons "Generate elements" and "Manual input" are enabled. The button "Generate elements" automatically generates array with the specified number of elements, and the button "Manual Input" allow us to enter sequentially elements (manual generate an array), using the field "Element No." and pressing the "Enter" key. Once the array is generated (automatically or manually), the software displayed it in the "Sorting Results" panel as follow:

96	856	262	805	193	349	344	520	956	469	Initial array
----	-----	-----	-----	-----	-----	-----	-----	-----	-----	---------------

After that the "Sorting algorithm" list is enabled. This list allows us to choose the sorting algorithm that want to monitor. Once we choose a sorting algorithm, the buttons "Sort", "C++ Source", and "Pascal Source" are enabled.

To start the sorting process, we need to press the "Sort" button, which will performs the first sorting step, will displays the result in the "Sorting Results" panel, and below the "Sort" button will displays the message "Perform the Steps", button "Continue" and button "Stop" (see Figure 2). The "Continue" button sequentially performs and visualizes the individual steps of the sorting process, and with the "Stop" button the sorting process may be interrupted.

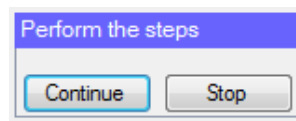


Fig. 2. The message "Perform the Steps"

The result of the implementation of the chosen algorithm can be deleted by pressing the "Clear" button, which is located under the "Sort" button.

Figure 3 shows the obtained result using Bubble sort and array with six elements (automatically generated).

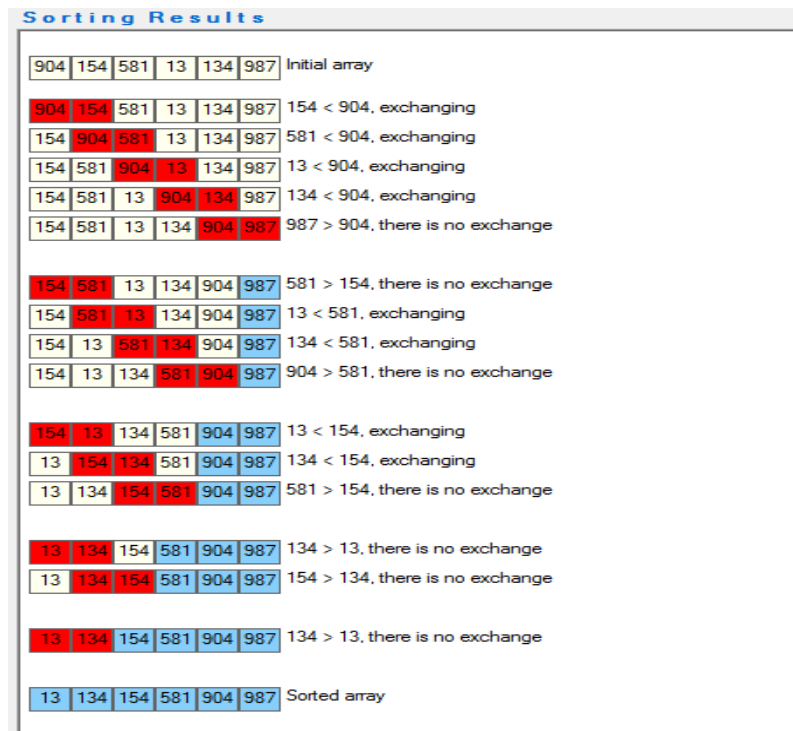


Fig. 3. Obtained result using Bubble sort algorithm and array with six elements

The software marks with white color the elements of initial array, with red color the elements that are compared in the current step, and with blue color already sorted elements (see Figure 3). Also, for each of the steps, the software describes the current action (whether or not the relevant elements are exchanged). The next figure (see Figure 4) shows the implementation of Merge sort algorithm.

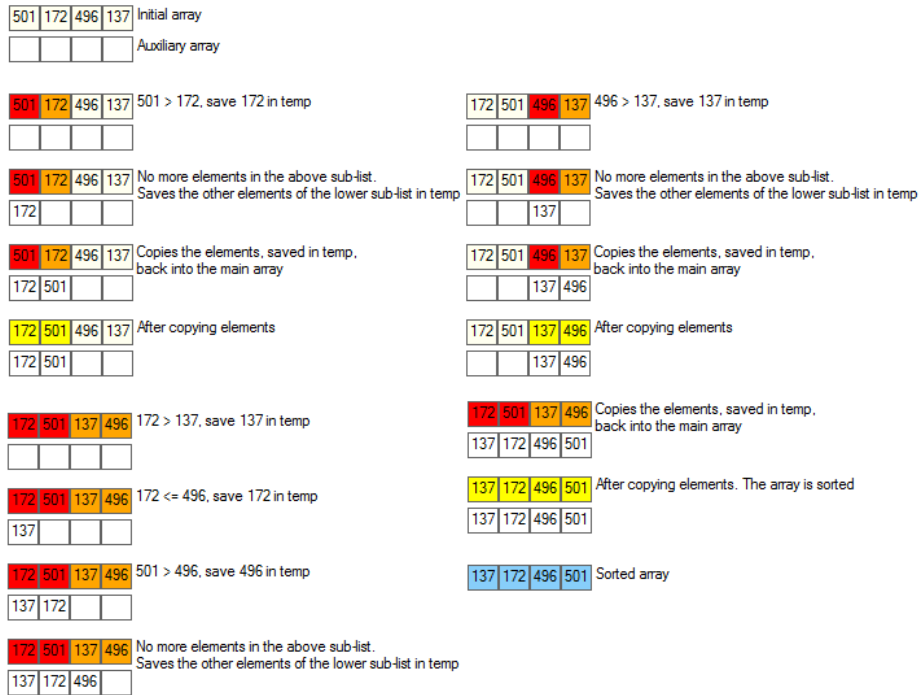
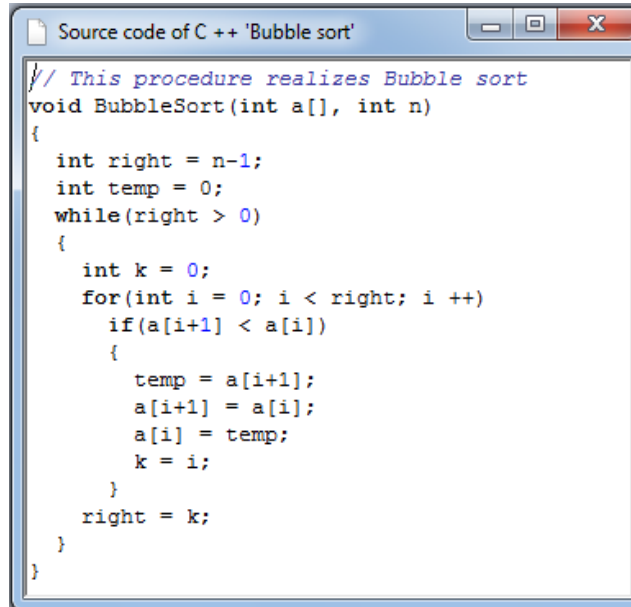


Fig. 4. Implementation of Merge sort algorithm

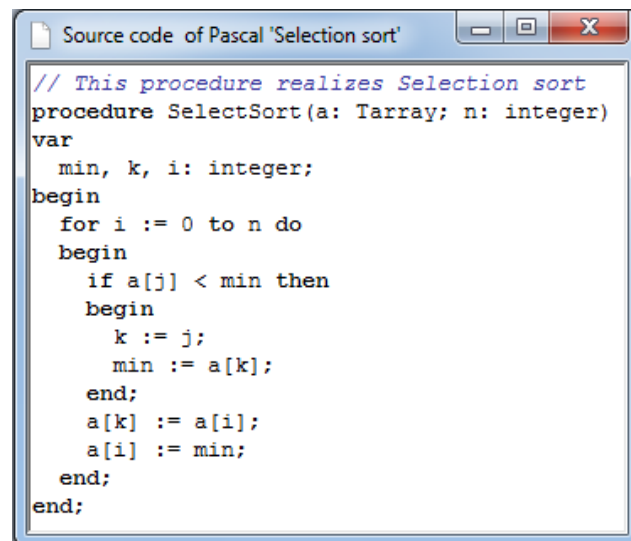
3.2 The form "Source code"

Developed software “Visual Sorting” can show C++ or Pascal source code for any of the described in previous section algorithms. The buttons "C ++ Source code" and "Pascal Source code" (Main form), open a form that contains C++ (see Figure 5) or Pascal (see Figure 6) source code for the chosen algorithm.

A screenshot of a window titled "Source code of C ++ 'Bubble sort'". The window contains C++ code for a bubble sort function. The code is as follows:

```
// This procedure realizes Bubble sort
void BubbleSort(int a[], int n)
{
    int right = n-1;
    int temp = 0;
    while(right > 0)
    {
        int k = 0;
        for(int i = 0; i < right; i ++)
            if(a[i+1] < a[i])
            {
                temp = a[i+1];
                a[i+1] = a[i];
                a[i] = temp;
                k = i;
            }
        right = k;
    }
}
```

Fig. 5. Form: C++ source code for Bubble sort

A screenshot of a window titled "Source code of Pascal 'Selection sort'". The window contains Pascal code for a selection sort procedure. The code is as follows:

```
// This procedure realizes Selection sort
procedure SelectSort(a: Tarray; n: integer)
var
    min, k, i: integer;
begin
    for i := 0 to n do
    begin
        if a[j] < min then
        begin
            k := j;
            min := a[k];
        end;
        a[k] := a[i];
        a[i] := min;
    end;
end;
```

Fig. 6. Form: Pascal source code for Selection sort

Using this form, the students can study the source code in C++ and Pascal programming languages, for each of the algorithms that are included in developed software.

4 Conclusion

In conclusion we can say that the use of the visual and event-oriented programming in the Computer science courses is excellent choice and they give us perfect results. For successful implementation of training courses that use the full potential of modern technology, very important are both the mindset and attitude of the lecturers to these technologies, and their use in the learning process, to achieve the educational goals [11-19].

In this article we considered the sorting algorithms that are included in the developed by us software with name "Visual sorting". Also, we presented our software to interactive tracking of the progressive execution of sorting algorithms.

5 Acknowledgement

This research would not have been possible without the financial assistance of the following project: "Application of the mixed reality in the training and promotion of the cultural heritage for the purposes of the in the university information environment" financed by National Science Fund of the Ministry of Education and Science of the republic of Bulgaria with Contract № KP – 06 – OPR 05/14 from 17.12.2018, led by Prof. DSc Irena Peteva.

6 References

- [1] Mavrevski R. (2014). Selection and comparison of regression models: estimation of torque-angle relationships. *C. R. Acad. Bulg. Sci.* 67(10), 1345-1354.
- [2] Traykov M., M. Trencheva, R. Mavrevski, A. Stoilov, I. Trenchev. (2016) Using partial differential equations for pricing of goods and services. *Scientific Annals of Economics and Business* 63(2), 291-298. <https://doi.org/10.1515/saeb-2016-0122>
- [3] Traykov M, M. Trencheva, E. Stavrova, R. Mavrevski, I. Trenchev. (2018). Risk analysis in the economics through R Language. *WSEAS TRANSACTIONS on BUSINESS and ECONOMICS*, 15, 180-186.
- [4] Mavrevski R., M. Traykov, I. Trenchev, M. Trencheva. (2018). Approaches to modeling of biological experimental data with GraphPad Prism software. *WSEAS TRANSACTIONS on SYSTEMS and CONTROL*, 13, 242-24.
- [5] Arora, G., Aiaswamy, B. (2002). *Microsoft C# Professional Projects*. Course Technology PTR, Kentucky, USA.
- [6] Johnson, B., Young, M., Skibo, G. (2003). *Inside Microsoft Visual Studio .Net*. 2003rd edn. Microsoft Press, Washington, USA.
- [7] Cormen, H., Leiserson, E., Rivest, L., Stein, C. (2009). *Introduction to Algorithms*. 3rd edn. MIT Press, Massachusetts, USA.
- [8] Abramson, D., Watson, G. (2003). Debugging scientific applications in the .NET Framework. *Future Generation Computer Systems* 19(5), 665–678. [https://doi.org/10.1016/s0167-739x\(02\)00176-0](https://doi.org/10.1016/s0167-739x(02)00176-0)
- [9] Börger, E., Fruja, N., Gervasi, V., Stärk, R. (2005). A high-level modular definition of the semantics of C#. *Theoretical Computer Science* 336(2-3), 235–284.

- [10] Armstronga, D., Loehr, N., Warrington, G. (2015). Sweep maps: A continuous family of sorting algorithms. *Advances in Mathematics* 284(1), 159-185. <https://doi.org/10.1016/j.aim.2015.07.012>
- [11] Tudor S. (2012). The Role of Multimedia Strategies in Educational Process. *Procedia - Social and Behavioral Sciences* 78, 682-686. <https://doi.org/10.1016/j.sbspro.2013.04.375>
- [12] Meia H, Chena W, Maa Y, Guana H, Hua W. VisComposer. (2018). A Visual Programmable Composition Environment for Information. Visualization. *Visual Informatics* 2(2018), 71-81. <https://doi.org/10.1016/j.visinf.2018.04.008>
- [13] Siple P., Hatfield N., Caccamise, F. (1978). The role of visual perceptual abilities in the acquisition and comprehension of sign language. *American Annals of the Deaf* 123(7), 852-856.
- [14] Todman J., Seedhouse E. (1994). Visual-action code processing by Deaf and hearing children. *Language and Cognitive Processes* 9, 129-141. <https://doi.org/10.1080/01690969408402113>
- [15] Mavrevski R., Traykov M. (2019). Visualization software for hydrophobic-polar protein folding model. *Scientific Visualization*, 11(1), 11-19. <https://doi.org/10.26583/sv.11.1.02>
- [16] Denchev S. Trencheva T. (2016). Intellectual Property as a Basic Part of the University's Information Literacy. 2nd International Conference on Education and Management Science (ICEMS), Beijing, may 28-29, 2016, 74-78.
- [17] Trencheva T., Denchev S. (2015) THE UNIVERSITY'S R&D INSTITUTES AS A NEW EDUCATIONAL APPROACH. 9th International Technology, Education and Development Conference (INTED), Madrid, mar 02-04, 2015, 951-957.
- [18] Denchev S., Varbanova K., Peteva I., Tetevenska B. (2010). Bulgarian university libraries as an e-learning support center. ICSIT 2010 - International Conference on Society and Information Technologies, Proceedings pp. 385-389.
- [19] Denchev S. (1993). Science and technology in the New Bulgaria *Technology in Society*, 15(1), 57-63.

7 Authors

Radoslav Mavrevski is Chief Assistant in Department of Informatics, Faculty of Mathematics and Natural Sciences, member of University Center for Advanced, Bioinformatics Research, South-West University "Neofit Rilski", 66 Ivan Mihaylov Str., Blagoevgrad, Bulgaria. PhD on Informatics. Scientific Interest: programming, computer modelling, applied statistics and bioinformatics. He is one of the organizers of the South Eastern European Mathematical Olympiad for University Students (SEEMOUS) with International Participation, 2012, <http://seemous2012.swu.bg/> and XXVII REPUBLICAN STUDENT PROGRAMMING OLYMPIAD, 2015, <http://bcpc.eu/XXVII/>.

Metodi Traykov is Assistant in Department of Informatics, Faculty of Mathematics and Natural Sciences, member of University Center for Advanced, Bioinformatics Research, South-West University "Neofit Rilski", 66 Ivan Mihaylov Str., Blagoevgrad, Bulgaria. PhD on Informatics. Scientific Interest: programming and bioinformatics. He is one of the organizers of the XXVII REPUBLICAN STUDENT PROGRAMMING OLYMPIAD, 2015, <http://bcpc.eu/XXVII/>.

Ivan Trenchev is Associate professor in Department of Electrical Engineering, Electronics and Automatics, Faculty of Engineering, member of University Center for

Advanced, Bioinformatics Research, South-West University "Neofit Rilski", 66 Ivan Mihaylov Str., Blagoevgrad, Bulgaria and Associate professor in University of Library Studies and Information Technologies Sofia, Bulgaria. PhD on Informatics. Scientific Interest: virtual reality (VR), computer modelling and bioinformatics. He is one of the organizers of the XXVII REPUBLICAN STUDENT PROGRAMMING OLYMPIAD, 2015, <http://bcpc.eu/XXVII/>.

Article submitted 2019-01-21. Resubmitted 2019-03-04. Final acceptance 2019-04-04. Final version published as submitted by the authors.

Appendix

The source code for the sorting algorithms in C++ and Pascal programming languages:

Source code for Bubble sort in C++:

```
1: void BubbleSort(int a[], int n) {
2:     int right = n-1;
3:     int temp = 0;
4:     while(right > 0) {
5:         int k = 0;
6:         for(int i = 0; i < right; i++) {
7:             if(a[i+1] < a[i]) {
8:                 temp = a[i+1];
9:                 a[i+1] = a[i];
10:                a[i] = temp;
11:                k = i;
12:            }
13:        }
14:        right = k;
15:    }
16: }
```

Source code for Bubble sort in Pascal:

```
1: procedure BubbleSort(a: Tarray; n: integer);
2: var
3:     right, temp, k, i: integer;
4: begin
5:     right := n-1;
6:     temp = 0;
7:     repeat
8:         k = 0;
9:         for i := 0 to right do
10:            if a[i+1] < a[i] then
11:                begin
12:                    temp := a[i+1];
13:                    a[i+1] := a[i];
```

```
14:         a[i] := temp;
15:         k := i;
16:     end;
17:     right := k;
18:     until (right > 0);
19: end;
```

Source code for Selection sort in C++:

```
1: void SelectSort(int a[], int n) {
2:     int min, k;
3:     for(int i = 0; i < n; i++) {
4:         k = i;
5:         min = a[k];
6:         for(int j = i + 1; j < n; j++) {
7:             if(a[j] < min) {
8:                 k = j;
9:                 min = a[k];
10:            }
11:        }
12:        a[k] = a[i];
13:        a[i] = min;
14:    }
15: }
```

Source code for Selection sort in Pascal:

```
1: procedure SelectSort(a: Tarray; n: integer)
2: var
3:     min, k, i: integer;
4: begin
5:     for i := 0 to n do
6:         begin
7:             if a[j] < min then
8:                 begin
9:                     k := j;
10:                    min := a[k];
11:                end;
12:            a[k] := a[i];
13:            a[i] := min;
14:        end;
15: end;
```

Source code for Insertion sort in C++:

//This procedure inserts elements in a dataset that is in increasing order

```
1: void Insert(int a[], int i) {
2:     int j;
3:     int x = a[i];
4:     for(j = i - 1; j >= 0; j--) {
5:         if(x < a[j])
```

```
6:         a[j+1] = a[j];
7:         else
8:             break;
9:     }
10:    a[j+1] = x;
11: }
```

// This procedure realizes Insertion sort

```
1: void InsertSort(int a[], int n) {
2:     for(int i = 1; i < n; i++)
3:         Insert(a, i);
4: }
```

Source code for Insertion sort in Pascal:

//This procedure inserts elements in a sequence that is in increasing order

```
1: procedure Insert(a: Tarray; i: integer);
2: var
3:     j, x: integer;
4: begin
5:     x := a[i];
6:     for j := i-1 downto 0 do
7:         begin
8:             if (x < a[j]) then a[j+1] := a[j];
9:         end;
10:    a[j+1] := x;
11: end;
```

// This procedure realizes Insertion sort

```
1: procedure InsertSort(a: integer; n: integer);
2: var
3:     i: integer;
4: begin
5:     for i := 1 to n do Insert(a, i);
6: end;
```

Source code for Merge sort in C++:

//This procedure combine (merge) two arrays

```
1: void merge(const int a[], int n, const int b[], int m, int* c) {
2:     int i = 0, j = 0, k = -1;
3:     while(i < n && j < m) {
4:         if(a[i] < b[j]) {
5:             k++;
6:             c[k] = a[i];
7:             i++;
8:         } else {
9:             k++;
10:            c[k] = b[j];
11:            j++;
12:        }
```

```

13: }
14: if(i == n) {
15:     while(j < m) {
16:         k ++;
17:         c[k] = b[j];
18:         j ++;
19:     }
20: } else {
21:     while(i < n) {
22:         k ++;
23:         c[k] = a[i];
24:         i ++;
25:     }
26: }
27: }
// This procedure realizes Merge sort
1: void MergeSort(int a[], int n) {
2:     if(n < 2)
3:         return;
4:     int nLeft = n / 2;
5:     int nRight = n - nLeft;
6:     MergeSort(a, nLeft);
7:     MergeSort(a + nLeft, nRight);
8:     int* p = new int[n];
9:     merge(a, nLeft, a + nLeft, nRight, p);
10:    for(int i = 0; i < n; i ++)
11:        a[i] = p[i];
12:    delete [] p;
13: }

```

Source code for Merge sort in Pascal:

```

//This procedure combine (merge) two arrays
1: procedure merge(a: Tarray; n: integer; b: Tarray; m: integer; var c: Integer);
2: var
3:     i, j, k: integer;
4: begin
5:     i := 0;
6:     j := 0;
7:     k := -1;
8:     repeat
9:         if a[i] < b[j] then
10:            begin
11:                k := k + 1;
12:                c[k] := a[i];
13:                i := i + 1;
14:            end

```

```
15:     else
16:     begin
17:         k := k + 1;
18:         c[k] := b[j];
19:         j := j + 1;
20:     end
21: until ((i < n) and (j < m));
22: if i = n then
23:     begin
24:         repeat
25:             k := k + 1;
26:             c[k] := b[j];
27:             j := j + 1;
28:         until (j < m)
29:     end
30: else
31:     begin
32:         repeat
33:             k ++;
34:             c[k] = a[i];
35:             i ++;
36:         until (i < n)
37:     end;
38: end
// This procedure realizes Merge sort
1: procedure MergeSort(a: Tarray; n: integer);
2: var
3:     nLeft, nRight, i: integer;
4:     p: Tarray;
5: begin
6:     if n < 2 then Exit;
7:     nLeft := n / 2;
8:     nRight := n - nLeft;
9:     MergeSort(a, nLeft);
10:    MergeSort(a + nLeft, nRight);
11:    merge(a, nLeft, a + nLeft, nRight, p);
12:    for i := 0 to n do a[i] := p[i];
13: end
```