Increasing the Engagement Level in Algorithms and Data Structures Course by Driving Algorithm Visualizations

Slavomír Šimoňák
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice, Slovak Republic
E-mail: slavomir.simonak@tuke.sk

Keywords: algorithms, data structures, algorithm visualization, study supporting system, active learning

Received: July 12, 2019

The paper presents the results of our research in the field of applying algorithm visualizations within Data structures and algorithms subject. We accomplished several experiments relating the ability of students to solve simple problems in a pure visual way in one case and by programming the solution using a particular programming language in another one. The experiments are described and the results are analyzed within the paper. In accordance with our previous informal experiences and the results of the analysis we found there can be some part of students, which will be able to apply an algorithm to concrete problem in a visual way, but will not be able to express it clearly enough and implement it in given programming language. As an attempt to cope with the situation, we propose a new teaching approach, together with a prototype of study supporting system, based on the idea that students would participate on creating simple visualizations, not just using them. The purpose of such approach is to help students to develop both types of skills - understanding the algorithms and implementing them as well, by increasing the engagement level and supporting the active learning.

Povzetek: Predstavljen je način predavanja o algoritmih in podatkovnih strukturah s pomočjo algoritmične vizualizacije.

1 Introduction

Data structures and algorithms, the subject of our interest within the paper, is one of fundamental subjects taught within the bachelor study program at our department. Since the subject is positioned in the second year of study, students are supposed to have the basic knowledge and some practical skills in programming [12]. The goal of the subject is to further enhance this skills and to provide the students with fundamental knowledge on data structures, methods for designing algorithms and to asses their efficiency. As the area of algorithms and data structures is more general and abstract, it is also more complicated to learn for many students [5]. One of widely adopted approaches to help with this situation is based on using algorithm visualizations within the subjects covering the area [8, 6]. Also within our subject, except the conventional ways of teaching (using pseudocodes of algorithms or diagrams), we also use algorithm visualizations for couple of years [17, 16]. Although the results achieved by using visualizations are promising [19, 20], we further try to improve the educational process in order to make it more attractive and efficient.

If we want to write a correct program for solving a given problem, we need to have pretty clear idea on the algorithm solving the problem. But we also need some practical skills to be able to implement the algorithm in a particular programming language. So if one of this two basic elements is missing it would be very challenging task, so we can give up, or it will take too long to write the required program. While the first one seems to be absolutely necessary, as without knowing the particular algorithm we will not be able to create its implementation, it does not automatically imply, that a student knowing the algorithm will be able to create the corresponding program. So our hypothesis is, that there will be some part of students, which will be able to apply an algorithm to concrete problem in a visual way, but will not be able to create its corresponding program implementation within the reasonable time limit.

In experiments described further within the paper we want to contribute to this idea by comparing the results of solving simple problems by students just in visual way in one case and by programming the solution in another one. We think it is very useful to utilize algorithm visualizations in order to help students to understand the basic principles of algorithms operation. But according to our experiences, we also believe there is still some gap between understanding the basic principles of particular algorithm and the ability to express it clearly enough in a given programming language. By the experiments we wanted to obtain some empirical results in order to support or to invalidate our informal experiences in this respect. After evaluating the results of experiments we provide some of our ideas how the quality of the understanding of the given topic as well

as the ability to implement particular algorithms could be improved jointly.

The rest of the paper is organized as follows. In section 2 we provide a short description on selection of some interesting and influential works on the topic of algorithm visualization and its effectiveness. Section 3 gives an overview of the algorithm visualization systems we developed to support the teaching process in the field of algorithms and data structures. Experiments we conducted are described in section 4 and the results of experiments are evaluated and analyzed in section 5 of the paper. Section 6 proposes a new approach, aiming at increasing the engagement level, supported by the prototype of new study supporting system. Section 7 concludes the paper and provides some ideas for further development.

2 Related work

There are many tools for algorithm visualizations available presently, which would indicate that algorithm visualizations are widely used in a field of algorithm and data structures education. But, as the results from the recent research in the filed indicate, the important part of effectiveness of algorithm visualization is how students are engaged in a learning activity. Authors of a meta-study [7] conclude, that studies in which students only viewed visualizations, usually did not indicate significant learning advantages over students using conventional learning materials. This can be perceived in a way, that the mere presence of visualizations does not guarantee that students will better understand algorithms. The results of this research also suggest that the most successful educational applications of algorithm visualizations are those in which the visualization is used as a vehicle for engaging students into the process of learning. So the form of the learning activity in which visualization technology is used is more important than the style of visualizations used.

One of modern approaches is based on learning through playing educational games [3, 4]. It is believed that using educational games can provide a wide range of benefits (like increased effectiveness, interest and motivation), but those are questionable or at least not rigorously established [11]. In the paper [5] authors present an educational game intended to help students in understanding the stack data structure on conceptual as well as practical level. The Stack Game was developed in three parts, corresponding to learning objectives (understanding the concept of stack, application of stacks, stack implementation), bound together by a meaningful storyline.

An interesting exercise support system, based on combining exercise tasks with automatic evaluation and integrated algorithm animation is described in [14]. The system is based on the established ANIMAL system, since it supports the ad-hoc generation of animation based on data provided by a user. So the creation of new exercise sheets based on existing sheets by the means of modifying the input parameters is supported by the system. The correct answer does not have to be given directly, since it can be determined by the system automatically, based on provided evaluation scripts.

While the above mentioned approaches are interesting, the approach described in this work for increasing students' engagement and motivation is slightly different and it is based on the idea that they would participate on creating visualizations not just using them. The approach, together with the prototype of associated supporting tool, will be described in greater details later in this paper.

Very interesting in this respect is the engagement taxonomy [10], which defines five levels of interaction [8] between a student and an algorithm visualization:

- viewing,
- responding,
- changing,
- constructing,
- presenting.

Several hypotheses are proposed in [10], which can be interpreted in a way, that the higher level or the more forms of engagement are used, the more efficient the learning becomes. Within our currently available tools (like Algomaster or VizAlgo, described briefly in the following section) first three levels of interaction from the engagement taxonomy are easily accessible for students. Mastering the fourth level (constructing visualizations) however, is bit more complicated, since it requires some knowledge about the structure of the application and its plugin modules. There has been some attempts to simplify the process of creating plugins for Algomaster platform [2], but the solution created has still limited area of usability.

As the approach presented later in this work is based on the idea of involving students deeper into the process of creating particular visualizations, it can be considered to provide the fourth level of interaction within the engagement taxonomy. Hence we hope it could help to increase the learning efficiency by providing the higher level of interaction, as well as to develop practical programming skills by implementing the algorithm under consideration.

3 Algorithm visualization tools

Within our subject we use algorithm visualizations as an education supporting tools for several years. We started with visualizations available from different authors. But, while it was a quick and simple solution, we encountered the limitations of various kinds time to time, so we started developing our platforms for visualizing algorithms and data structures. Platforms were developed in order to fulfill our specific needs regarding the selection of algorithms, naming conventions or the ability to adapt the visualizations whenever we decided. The platforms are only briefly

described here, to provide a reader with basic context, since one of them (Algomaster) was also involved in experiments described in the next section. For more information, additional references are provided as well. The first of the platforms was named VizAlgo [16] and it was developed with emphasis on two main goals: extensibility and portability. The first of the goals is reflected within the structure of the application, which consists of two cooperating parts - the core module and the set of relatively independent plugin modules. The core module is intended to provide the support for displaying and controlling the algorithm execution, while plugin modules are responsible for visualizations of particular algorithms. Choosing the Java development platform was connected with the second of the goals mentioned above. The platform is still in development and over the years not only the set of available visualizations was changing, but also the core functionalities and user interface [1] evolved (Figure 1).

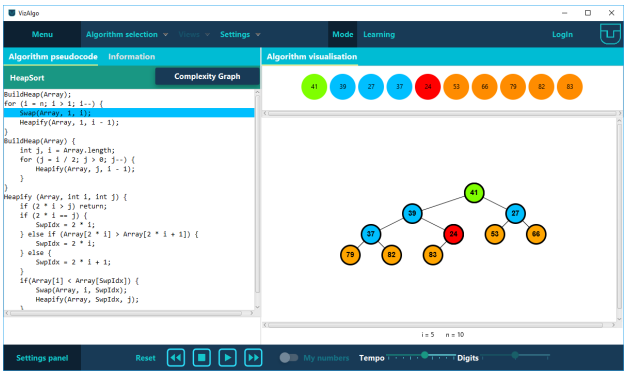

Figure 1: VizAlgo platform

According to experiences gained with the VizAlgo, the second of the platforms, Algomaster [18], also has the plugin-based architecture, but it was intended to provide some more advanced features. The features include functionality for algorithm stepping in both directions [13], call stack visualization for recursive algorithms and a special mode for practical student testing in a visual way. In contrast to the VizAlgo, the Algomaster is based on .NET framework development and execution platform [9]. Later on, the platform was extended significantly [2] in order to provide the support for visualization of complex algorithms with the ability of changing input data during the visualization. Examples of visualizations using the new features are operations on B-tree, 2-3 tree or AVL tree (Figure 2).

In addition to the ability to define input data dynamically, extensions were also oriented towards a real-time student testing and support for simplified development of plugins for the platform. In order to simplify the creation of Algomaster plugins, a separate application named AlgoCreator (Figure 3) was developed.

The application uses a pattern for generating plugins of particular algorithm class, e.g. pattern for comparison-based sorting algorithms. A pattern consists of text templates for source code generation and an interpreter for interpreting user defined model. In short, a process of creating a plugin module can be described as follows: a user can select one of available patterns, provide basic algorithm-related information and the algorithm pseudocode, define the behavior of the algorithm and initiate library generation. The process is described in deeper detail in [2].

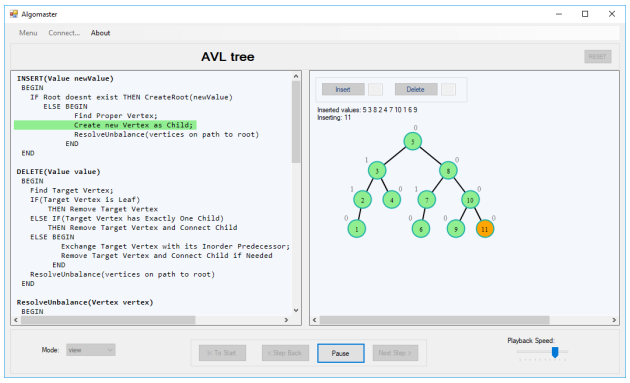

Figure 2: Algomaster platform

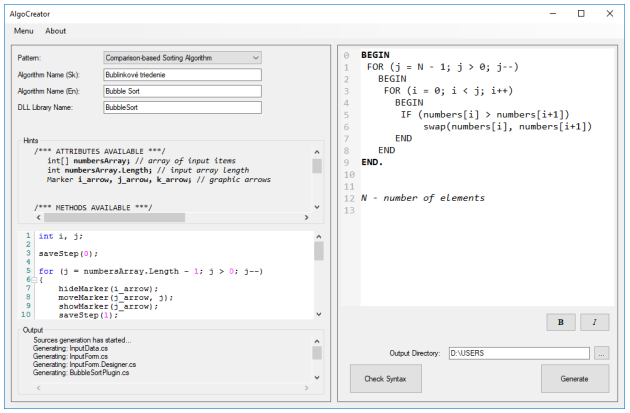

Figure 3: An environment of AlgoCreator application

4 Experiments

As it was yet mentioned within introductory part of the paper, the main motivation behind the accomplished experiments was the comparison of the ability of students to solve algorithmic problems in two distinct ways. One of them was based on visual "simulation" of given algorithm operation, using one of our visualization tools, described in section 3. The another one consisted of programming the particular algorithm in given programming language. Experiments considered in this work were conducted with students of four study groups (G_1-G_4) and they were focused on two basic areas. The first area was oriented on traversing trees using different strategies (T) and the second one on simple comparison-based sorting algorithms (S).

Thus assignments of the particular area consisted of two

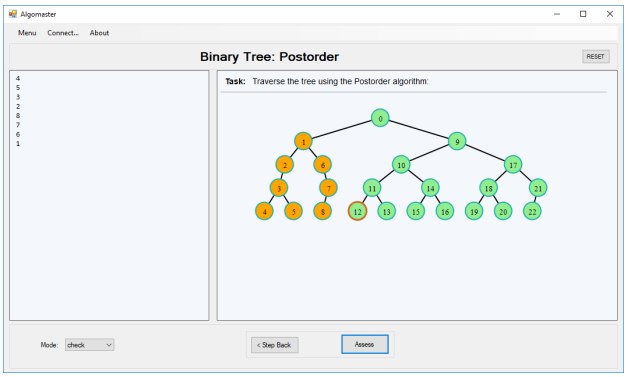

Figure 4: Algomaster in check mode - traversing binary tree

parts: solving the problem in pure visual way using the Algomaster platform (V) in one case (Figure 4) and programming the particular algorithm in C programming language (P) in the another one. This way we got the four combinations (two areas and two ways of solving a problem from the given area - TP, TV, SP and SV) for each of four study groups (G_1-G_4).

In the area of tree traversing, three basic traversing strategies were used (in-order, pre-order and post-order). In the area of sorting, simple sorting algorithms (like Insert sort and Bubble sort) were used. Within the following four tables (Table 1, Table 2, Table 3, and Table 4), individual scores are presented, achieved by students of particular study groups (G_1-G_4) in all experiments (TP, TV, SP, SV).

Experiment	Results
TP-G_1	0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0
TV-G_1	0.06 1 1 1 1 1 1 0.9 0 0 1 1 1 0.2 0.07 1 1 1 0 1 0
SP-G_1	0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 - 0 0 1 0
SV-G_1	1 1 1 1 1 0.06 0.15 1 1 0.77 1 1 1 0.57 0.05 1 - 1 0 1 0.18

Table 1: Results achieved by the students of the study group G_1

Experiments described within this section were conducted in Fall 2017. 82 students were considered on experiments in total, of which 72 were males and 10 were females. Since all the activities were not necessarily conducted on a single class, not all students were necessarily present on all activities. Such situation can be distinguished in particular table by the presence of "-" character within the Results column. This fact can be perceived as a slight disadvantage, but it is generally hard to influence the presence of students on classes. And since it was registered only in few individual cases from all considered students, we believe it was not affecting the results significantly.

Experiment	Results
TP-G_2	0 0 0 0 1 0 1 - 0 1 0 0 0 0 0 1 1 1 0 1 1
TV-G_2	1 0.67 0.09 1 1 0.21 0 1 0.38 0.4 1 0.14 1 1 1 0.6 1 1 1 0.79 0.67
SP-G_2	1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 1 0 1 1
SV-G_2	1 1 0.33 1 0 1 1 1 1 1 0.81 0 0.05 0.62 1 1 1 1 1 1 1

Table 2: Results achieved by the students of the study group G_2

Experiment	Results
TP-G_3	0 0 0 0 0 1 1 1 1 1 - 1 1 1 1 1 - 0 1 0
TV-G_3	0.46 0 1 0.75 1 0.14 1 1 0.13 1 0.2 1 0.1 1 1 1 1 1 1 1
SP-G_3	0 1 1 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0 1 0
SV-G_3	1 1 1 1 1 1 1 0.93 1 1 1 1 1 1 1 1 0.04 0.29 1 1

Table 3: Results achieved by the students of the study group G_3

5 Analysis of the results of experiments

Within this section we provide a sketch of approach for calculating some of the resulting values, summarize the obtained results and formulate some comments on them. For calculating the average scores (mean) of the first group (G_1) of students in particular experiments (TP, TV, SP, SV), the following formulas (1 - 4) were used. The *average score* (G_1Av_{TP}) achieved by the study group (G_1) in the experiment TP is given by the formula (1). Within the formula, G_1TS_{TP} represents the *total score* achieved by the group G_1 in the TP experiment and G_1NS_{TP} the *number of students* participating in the experiment. The mean values (G_1Av_{TV}, G_1Av_{SP}, G_1Av_{SV}) for remaining experiments (TV, SP, SV) of the study group G_1 were calculated analogically.

Experiment	Results
TP-G_4	1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0
TV-G_4	1 0.63 1 0 0.4 1 0 1 0.33 1 1 0 1 1 1 0.86 1 1 1 1
SP-G_4	1 0 0 0 0 0 1 1 - 1 0 1 1 1 0 1 1 0 1 0
SV-G_4	1 0.7 0.16 1 1 1 1 1 - 1 1 1 1 1 1 1 1 1 1 1

Table 4: Results achieved by the students of the study group G_4

$$G_1Av_{TP} = \frac{G_1TS_{TP}}{G_1NS_{TP}} = \frac{6}{21} = 0.286, \quad (1)$$

$$G_1Av_{TV} = \frac{G_1TS_{TV}}{G_1NS_{TV}} = \frac{14.23}{21} = 0.678, \quad (2)$$

$$G_1Av_{SP} = \frac{G_1TS_{SP}}{G_1NS_{SP}} = \frac{7}{20} = 0.35, \quad (3)$$

$$G_1Av_{SV} = \frac{G_1TS_{SV}}{G_1NS_{SV}} = \frac{14.78}{20} = 0.739. \quad (4)$$

Similarly, the mean values were calculated for remaining groups (G_2 - G_4), based on the data presented in tables Table 2, Table 3, and Table 4. Variance and standard deviation values for all experiments were calculated as well and the overall results are available in the table Table 5.

Exper.	Group	Mean	Variance	Std. deviat.
TP	G1	0.286	0.204	0.452
TV		0.678	0.201	0.448
SP		0.35	0.228	0.477
SV		0.739	0.153	0.391
TP	G2	0.4	0.24	0.490
TV		0.712	0.125	0.353
SP		0.667	0.222	0.471
SV		0.8	0.128	0.358
TP	G3	0.611	0.238	0.487
TV		0.739	0.147	0.383
SP		0.55	0.248	0.497
SV		0.913	0.064	0.253
TP	G4	0.35	0.228	0.477
TV		0.761	0.141	0.376
SP		0.526	0.249	0.499
SV		0.94	0.038	0.196

Table 5: Statistical results of experiments

As we can observe from the graph of average scores (Figure 5) achieved by students in particular activities, the scores achieved in visual tasks are usually significantly higher than the scores achieved in corresponding programming tasks. The only differences are TP/TV relation for the group G_3 and SP/SV relation for the group G_2. Also in these cases the scores achieved by solving problems in visual way are higher, but maybe not so significantly.

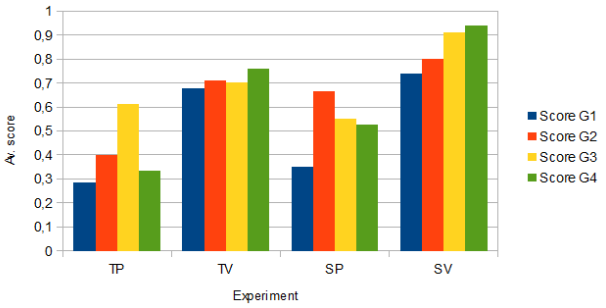

Figure 5: Graph of average scores (study groups)

When we further average the results obtained in particular experiments, better results in visual tasks become clearly visible (Figure 6). These results practically support our informal experiences and the hypothesis expressed within the introductory part of the paper.

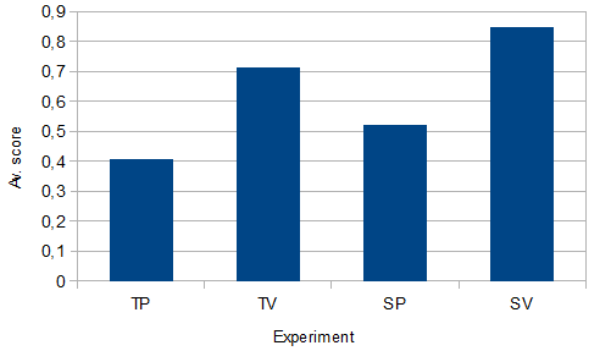

Figure 6: Graph of average scores (experiments)

The results can be also interpreted in a way, that algorithm visualizations provide the solid potential we would like to build upon and examine the new ways of utilizing them in the filed of algorithms and data structures education.

6 Proposal of the new approach and study supporting system

In order to cope with the situation and stimulate further students' algorithmic and programming skills, while taking advantage of algorithm visualizations, we propose a new teaching approach supported by the prototype of new study supporting system. As it was mentioned before, the teaching approach is based on the idea that students would participate on creating simple visualizations, and this way interact with algorithm visualization on a higher level of the engagement taxonomy. The role of the proposed system is to provide the environment, that allows students to control the pre-arranged visualizations from their code by using simple programming constructs. The approach, together with the system are intended to be used in conjunction with other teaching methods, not to replace them. The prototype of the system with a working name DSAV (Figure 7) combines algorithm visualizations with programming tasks and so increases the engagement level and supports active learning.

As a proof of concept, we implemented the support for several (Bubble sort, Selection sort, Insertion sort, Quick sort, Heap sort, Merge sort) sorting algorithms [15] and algorithms for traversing binary trees (Figure 8) using various strategies (Inorder, Preorder, Postorder and Level-order). We would like to enhance the system in the future, and perspective areas for such enhancements would be visualizations of operations on lists, trees, or graphs.

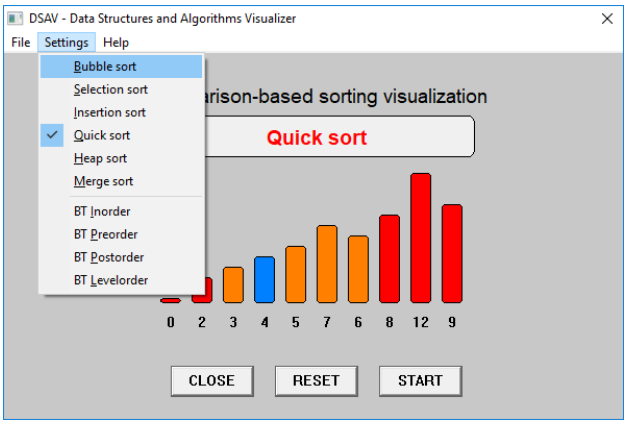

Figure 7: The working prototype of DSAV system

Technically, the system essentially consists of two parts: the main part, managing the user interface and visualization, and a separate thread implementing the algorithms to be visualized. There is a simple API consisting of several supporting operations which can be used appropriately by a programmer implementing a particular algorithm. The basic operation available is (`RedrawAndWait(int millis)`) telling the system to update the visualization according to current values in a data structure shared by both parts and wait for a specified amount of time.

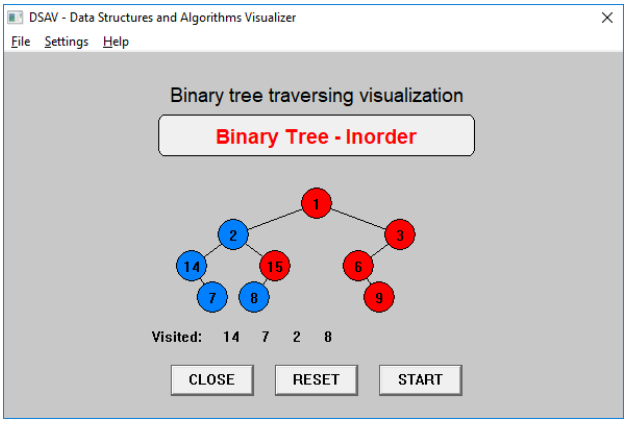

Figure 8: Visualization of traversing binary tree

In case of sorting algorithms we provide several simple API calls for rendering some elements of sorted array in different color. They can be useful in cases we want to put special emphasis on particular element (elements) of the sorted sequence (Figure 7). Some of them are given in the following list:

- `CSortClearColorArr()`
- `CSortSetColor(int index)`
- `CSortClrSetColor(int index)`
- `CSortSetColorInt(int begidx, int endidx)`
- `CSortClrSetColorRW(int index, int millis)`

As some of operations tend to be used often together, we also provide special calls for performing combined operations (e.g. `CSortClrSetColorRW(index, millis)` combines `CSortClrSetColor(index)` for rendering specified element in different color and `RedrawAndWait(millis)`). The reason for introducing such combined calls is to leave the code of particular algorithm closer to its original form. Figure 9 provides an example of using one of the operations within a simple sorting algorithm.

```
void selectionsort(int a[], int size)
{
    int i,j,k;
    for(i=0;i<size;i++)
    {
        int min = i;
        for(j=i+1;j<=9;j++)
            if(a[j]<a[min])
                min = j;
        k = a[i];
        CSortClrSetColorRW(min, 500);
        a[i] = a[min];
        a[min] = k;
        CSortClrSetColorRW(i, 500);
    }
}
```

Figure 9: A simple sorting algorithm implementation within the DSAV system

Analogically, there is a set of simple supporting API calls for visualization of tree traversing algorithms. Some of them are provided in the following list:

- `Btree3AGetDepth(int root)`
- `Btree3AGetLevel(int root, int d)`
- `BT3GetListVisited()`
- `Btree3ASetVisited(int root)`

An example of implementation of simple traversing algorithm is given in Figure 10.

Debugging outputs for particular algorithm can be printed using console output, if needed. The DSAV is a Win32 application, written in C/C++ programming language, since students mainly use this language in exercises within our subject presently.

7 Conclusion

Within the paper we described our experiments based on solving problems from given areas by students in two different ways. The first way was purely visual, accomplished

```
void inorder(int root){
    if(left[root]!=0)inorder(left[root]);
    Btree3ASetVisited(root);
    RedrawAndWait(300);
    if(right[root]!=0)inorder(right[root]);
}
```

Figure 10: A simple traversing algorithm implementation within the DSAV system

by using the Algomaster platform and the second way was based on programming a particular algorithm using C programming language.

The results acquired are presented and analyzed. We found that the scores achieved in visual tasks are usually significantly higher than the scores achieved in corresponding programming tasks. This correlates with our previous informal experiences and supports the validity of the hypothesis expressed within the introductory part of the paper.

The solution is proposed based on the idea of involving students into the process of creating algorithm visualizations. By the proposed solution we would like to help students not only to understand the basic principles of the particular algorithm in a convenient visual way, but also to stimulate their ability to implement it in particular programming language. Based on our experiences, confirmed by the results of accomplished experiments we believe, we should develop both of the skills in order to better prepare our students for their future professional career.

It would be interesting to further develop the proposed approach and the supporting system and study the contributions of the approach. Except the additional sorting algorithms, perspective areas for further extension would include visualizations of lists, trees, graphs or hash tables. We believe, that if system is enhanced properly and utilized in a right way, it would contribute to the quality of education in the subject. However, the further research is required in order to evaluate the benefits and efficiency of the proposed solution.

References

[1] Bačíková M., Porubän J.: Ergonomic vs. Domain Usability of User Interfaces, HSI 2013: 6th International Conference on Human System Interaction, June 6. - 8. 2013, Sopot, Poland, Piscataway, IEEE, 2013, pp. 1-8. https://doi.org/10.1109/hsi.2013.6577817

[2] Benej M., Šimoňák S.: Algomaster platform extension for improved usability, Journal of Electrical and Electronics Engineering, vol. 10, no. 1, 2017, pp. 27-30.

[3] Boyle E.A., Connolly T.M., Hainey T.: The role of psychology in understanding the impact of computer games, Entertainment Computing, vol. 2, no. 2, 2011, pp. 69-74. https://doi.org/10.1016/j.entcom.2010.12.002

[4] Boyle E.A., Hainey T., Connolly T.M., Gray G., Earp J., Ott M., et al. An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games, Computers & Education, 94, 2016, pp. 178-192. https://doi.org/10.1016/j.compedu.2015.11.003

[5] Dicheva D., Hodge A.: Active Learning through Game Play in a Data Structures Course, Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18), ACM, New York, NY, USA, 2018, pp. 834-839. https://doi.org/10.1145/3159450.3159605

[6] Grissom S., McNally M.F., Naps T.: Algorithm visualization in CS education: comparing levels of student engagement, Proceedings of the 2003 ACM symposium on Software visualization (SoftVis '03), ACM, New York, USA, 87-94. https://doi.org/10.1145/774833.774846

[7] Hundhausen C. D., Douglas S. A. and Stasko J. T.: A meta-study of algorithm visualization effectiveness, Journal of Visual Languages and Computing, 13, 2002, pp. 259-290. https://doi.org/10.1006/jvlc.2002.0237

[8] Karavirta V., Shaffer C. A.: Creating Engaging Online Learning Material with the JSAV JavaScript Algorithm Visualization Library, IEEE Transactions on Learning Technologies, vol. 9, no. 2, pp. 171-183, April-June 2016. https://doi.org/10.1109/tlt.2015.2490673

[9] Microsoft .NET, https://dotnet.microsoft.com/

[10] Naps T. L., Rößling G, et al.: Exploring the role of visualization and engagement in computer science education, Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '02), ACM, New York, NY, USA, 131-152. https://doi.org/10.1145/960568.782998

[11] Petri G., von Wangenheim C. G.: How games for computing education are evaluated? A systematic literature review, Computers & Education, vol. 107, April 2017, pp. 68-90. https://doi.org/10.1016/j.compedu.2017.01.004

[12] Pietriková E., Chodarev S.: Towards Programmer Knowledge Profile Generation, Acta Electrotechnica et Informatica, vol. 16, no. 1, 2016,

pp. 15-19. https://doi.org/10.15546/aeei-2016-0003

[13] Rößling G.: A First Set of Design Patterns for Algorithm Animation, Electronic Notes in Theoretical Computer Science, Volume 224, 2009, pp. 67-76. https://doi.org/10.1016/j.entcs.2008.12.050

[14] Rößling G., Mihaylov M., Saltmarsh J.: AnimalSense: Combining Automated Exercise Evaluations with Algorithm Animations, Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011, Darmstadt, Germany, June 27-29, 2011, pp. 298-302. https://doi.org/10.1145/1999747.1999831

[15] Silváši F., Tomášek M.: Lean Formalization of Insertion Sort Stability and Correctness, Acta Electrotechnica et Informatica, vol. 18, no. 2, 2018, pp. 42-49. https://doi.org/10.15546/aeei-2018-0015

[16] Šimoňák S.: Algorithm Visualization Using the VizAlgo Platform, Acta Electrotechnica et Informatica, vol. 13, no. 2, 2013, pp. 54-64. http://aei.tuke.sk/papers/2013/2/08_%C5%A0imo%C5%88%C3%A1k.pdf

[17] Šimoňák S.: Using algorithm visualizations in computer science education, Central European Journal of Computer Science, vol. 4, no. 3, 2014, pp. 183-190. https://doi.org/10.2478/s13537-014-0215-4

[18] Šimoňák S., Benej M.: Visualizing Algorithms and Data Structures Using the Algomaster Platform, Journal of Information, Control and Management Systems, vol. 12, no. 2, 2014, pp. 189-201.

[19] Šimoňák S.: Algorithm visualizations as a way of increasing the quality in computer science education, SAMI 2016, Danvers, IEEE, 2016, pp. 153-157. https://doi.org/10.1109/sami.2016.7422999

[20] Urquiza-Fuentes J., Velázquez-Iturbide J. Á.: Pedagogical Effectiveness of Engagement Levels - A Survey of Successful Experiences, Electronic Notes in Theoretical Computer Science, Volume 224, 2009, pp. 169-178. https://doi.org/10.1016/j.entcs.2008.12.061