Informatics in Education, 2017, Vol. 16, No. 1, 121–140

DOI: 10.15388/infedu.2017.07

Algorithm Animations for Teaching and Learning the Main Ideas of Basic Sortings

Ladislav VÉGH[1], Veronika STOFFOVÁ[2]
[1]*Department of Mathematics and Informatics, J. Selye University, Komárno, Slovakia*
[2]*Department of Mathematics and Informatics, Trnava University in Trnava, Slovakia*
e-mail: veghl@ujs.sk, veronika.stoffova@truni.sk

Received: November 2016

Abstract. Algorithms are hard to understand for novice computer science students because they dynamically modify values of elements of abstract data structures. Animations can help to understand algorithms, since they connect abstract concepts to real life objects and situations. In the past 30–35 years, there have been conducted many experiments in the field of usage of animations and visualizations in education, but they showed mixed results. In this paper, we review past research within the field and summarize recommendations regarding the graphic design and interactivity of the animations. In the second part of the paper, we present our interactive card sorting animations with conceptual views. The goal of these animations is to help students understand the main ideas and differences between basic sorting algorithms. In a pedagogical experiment related to these animations, 92 first-year computer science students of J. Selye University in Komarno (Slovakia) were asked to fill in a pre-test, experiment with the interactive animations, and fill in a post-test. The results showed that animations helped students to understand essential aspects of sorting algorithms. However, the participants were not able to understand the sorting algorithms in detail, so other types of animations are needed to teach algorithms in-depth.

Keywords: multimedia learning, interactive algorithm animations, teaching algorithms.

1. Introduction

To learn programming and understand algorithms is one of the hardest tasks for first-year computer science students. To acquire skills necessary for programming, four types of knowledge are needed (Bellstrom and Thoren, 2009):

- Basic mathematical knowledge of solutions to the problem.
- Knowledge of the programming environment (IDE), e.g. how to write and edit the source code, how to compile and run the program, how to add breakpoints and use the watch window.

- Programming language, e.g. knowledge about the data structures, functions, procedures, and syntax of the selected programming language.
- Transforming the knowledge into the logic of the program, e.g. combining the data structures, functions and control structures in a program to solve a specified task. This knowledge also includes the understanding of a given program code and different algorithms, as well as adopting them into own applications.

Students usually have the most problems with the fourth type of knowledge. It might be helpful for novice programmers to get to know common algorithms, e.g. calculating the sum or average of numbers, different operations on arrays, sorting algorithms, and basic recursive algorithms. These algorithms can be used as templates and building blocks for solving more complex problems. However, it is not easy for the first-year computer science students to comprehend these algorithms. The computer science algorithms usually use abstract data structures, and they dynamically change data. Animations and visualizations can create a bridge between abstract concepts and real-life situations (Rudder, Bernard and Mohammed, 2007).

2. How to Create and Use Algorithms Animation in Education – Literature Review

Animations and visualizations illustrate data structures in a graphical form, and they use dynamic graphics to show processes. Students like learning with animations, but using them in education is not effective in every case. The experiments in this field showed mixed results in the last 30–35 years (Byrne *et al.*, 1999; Hansen *et al.*, 2002; Hundhausen and Douglas, 2000; Hundhausen *et al.*, 2002; Kann *et al.*, 1997; Kehoe, 2001).

There are two broad categories of tools used in the studies of educational algorithm animations.

- Some of the experiments used visualization systems, like JHAVÉ, Jeliot, BlueJ, ALVIS Live!, Balsa-II, Polka. These systems are similar to a debugging mode of development environments, but in addition they contain a graphical representation of the data structures (Fleischer and Kucera, 2002). This kind of tools usually visualize the processes and data structures using the source code written by students (Lattu, Meisalo, and Tarhio, 2003). The main advantage of these tools is that students can write and modify their own code. The disadvantage of these visualizations is that the data structures and processes are displayed in the same form for every algorithm and the visualization does not take into the consideration the typical characteristics of the visualized algorithms.
- The second group of the studies used animations which were made in order to demonstrate only one type of algorithms or only a specific algorithm. Creating this kind of educational animations require more time and effort, but in return the animations better highlight the individual characteristics of algorithms, they may be focused on the important steps in the algorithms, and they can contain more interactive elements. These features of the animations help students to understand

the visualized processes better (Fleischer and Kucera, 2002; Kann *et al.*, 1997). Nowadays, these types of animations are developed in Adobe Flash or HTML5/ JavaScript.

2.1. *Principles of Multimedia Learning*

Many research papers suggest that educationally effective animations need to be graphically well-designed, following the principles of the multimedia learning (Mayer, 2009; Rudder *et al.*, 2007). Besides the appropriate graphical representation of the data structures and processes, animations should be interactive; thus, students can actively participate in the visualization processes instead of passive observations (Grissom, McNally, and Naps, 2003; Katai and Tóth, 2010; T. L. Naps *et al.*, 2002).

To design educationally effective animations, creators should be familiar with Mayer's cognitive theory (see Fig. 1) and principles of multimedia learning (Mayer, 2009). The cognitive theory describes how the human mind works.

While applicating this theory and some pedagogical experiments, Mayer defined twelve principles of multimedia learning grouped into the following three categories:

- Principles of reducing extraneous processing in multimedia learning (coherence principle, signaling principle, redundancy principle, spatial contiguity principle, temporal contiguity principle).
- Principles of managing essential processing in multimedia learning (segmenting principle, pre-training principle, modality principle).
- Principles of fostering generative processing in multimedia learning (multimedia principle, personalization principle, voice principle, image principle).

All these principles describe how, when, and in which manner should be texts, sounds, voices, and images used in multimedia learning materials. Experiments show that these principles are helpful especially for students with a low level of knowledge (Kehoe *et al.*, 2001; Mayer, 2009).

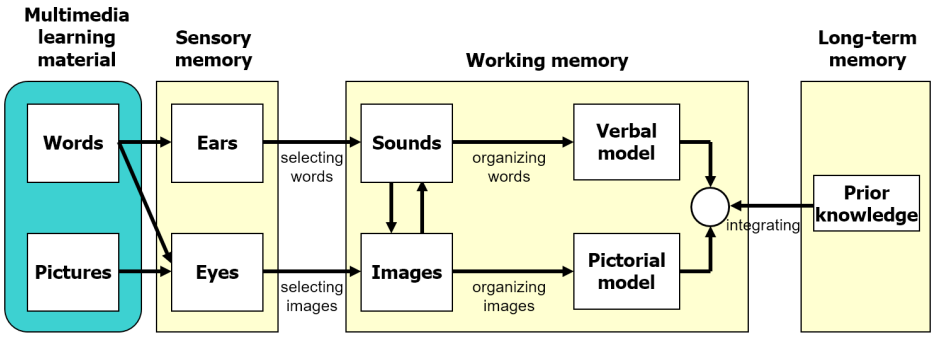

Fig. 1. Mayer's cognitive theory of multimedia learning.

2.2. *Design of Algorithm Animations*

Recommendations regarding the design of computer science algorithm animations are following:

- **The right model should be chosen for representing the data structures.** Because the algorithms work with abstract data structures, it is important to choose the right model representing these data structures (Esponda-Arguero, 2010; Fleischer and Kucera, 2002; Végh, 2011) – e.g. the values of elements in an array can be represented by the heights of columns, lightness of balls, or values of playing cards (see Fig. 2).
- **Algorithms should be demonstrated on small data set.** Previous studies suggest only 6–8 elements to use for a demonstration of algorithms (Fleischer and Kucera, 2002). In our opinion, a small number of elements may not be sufficient to illustrate the properties of more complex algorithms e.g. quicksort. We recommend using rather 8 or 16 elements for quicksort algorithm. It is also important to choose the adequate values of items in the input data set. For many algorithms, it is satisfying to pick random numbers, however, for the first presentation of algorithms like quicksort it is better to carefully choose the input values to be the data set divided into two equal parts during the running of the sorting algorithm.
- **Animations should be completed with explanations.** Explanations may help students to understand the visualized data and processes easier. They can be in the form of text, or in the form of voice. The explanation does not need to be necessarily part of the animation. It can be an oral implementation of a teacher during the lecture or textual explanation right before the animation in a textbook (Fleischer and Kucera, 2002; Mayer, 2009; Naps *et al.*, 2002).

When the explanation is in textual form, it is important to give students enough time to read and comprehend it. According to the principles of Mayer's multimedia learning,

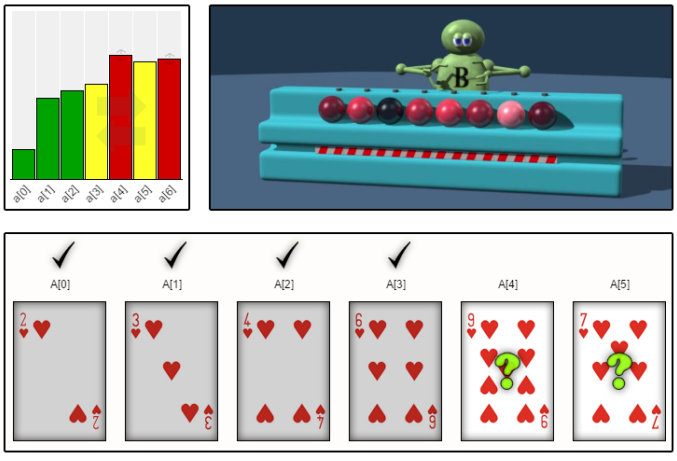

Fig. 2. Representation of array using different models.

it is not recommended to show the text and animate the objects on the screen at the same time. It is better to let students start the animation themselves after reading and understanding the explanation. Another possible solution could be to use narration during the animation instead of textual explanation (Mayer, 2009).

The results of several experiments emphasized that the animations may be used more efficiently when they are not displayed alone, but when they are part of a learning environment (Kann *et al.*, 1997; Kehoe *et al.*, 2001; Rudder *et al.*, 2007). This environment could be an electronic textbook with hypertext structure, enriched with embedded animations, diagrams, and examples (Hansen *et al.*, 2002). In these cases, it is vital, however, that the explanatory texts should correspond with the diagrams and animations, e.g. in a quicksort algorithm the pivot should be chosen using the same method. Otherwise, the animations will not help, but they will confuse students (Naps and Grissom, 2002).

- **Different views of algorithm animations enhance understanding of the visualized processes.** Students may observe the animation in different ways; they can concentrate on the animated objects, examine the sequence of commands in the source code and watch the control variables of cycles. It might be useful if the animations contain different views, e.g. in one part of the visualization are shown the animated columns with control variables of cycles under them while in the other part is shown the source code with the highlighted lines of current steps (Naps and Grissom, 2002).
- **Showing the source code or pseudocode of the visualized algorithms might be useful.** By showing the code with the highlighted lines of the current steps, students may connect the lines of the code with the events in the animation (Fleischer and Kucera, 2002; Naps *et al.*, 2002). The source code in a programming language is recommended to be shown only in programming courses. While teaching algorithms, it is better to show a pseudocode instead of the source code, because the pseudocode describes the algorithm at a higher level. Thus, students may learn the algorithm in abstract level, independently of any programming language; and they will not get lost in the details of the source code (Fleischer and Kucera, 2002).
- **The graphics should be simple, the colors and sounds should also carry information.** It is important to use simple graphic elements in the animations, which do not draw attention from the visualized processes. Colors and sound effects can also carry information (Fleischer and Kucera, 2002). In our applications of sorting algorithm, red color means that the elements are not sorted, yellow color is the color of the comparison or swap while green color may be used to visualize the sorted part of the array.
- **Showing information about the correctness and effectiveness of the algorithms might be helpful.** Showing the correctness and the effectiveness of the visualized algorithm in some form may be valuable for students when they try to understand and compare different algorithms solving similar problems (Fleischer and Kucera, 2002). Displaying the number of comparisons or swaps in sorting algorithms might give useful information especially for advanced students. A more descriptive solution can be a visualization of different sorting algorithms at the

same time (Naps *et al.*, 2002). Such examples can be found at `www.sorting-algorithms.com` and `www.sorting.at` websites.

- **Using animations with similar look and control buttons within the same course may help students.** A computer science course devoted to the field of algorithms contains lots of animations. Using same models, views, and control buttons in animations might help students. If visually different kinds of animations with different control buttons were used in a course, students would need more time to comprehend the visualized processes and acquire the control of every animation (Fleischer and Kucera, 2002).

2.3. *Educational Perspective*

Bloom's taxonomy of educational objectives (Bloom *et al.*, 1956) also should be taken into consideration when developing any learning material. The revised Bloom's taxonomy (Anderson *et al.*, 2001) contains six cognitive process dimensions (see Fig. 3).

Using this taxonomy, we can assign the results of cognitive activities observed during learning algorithms and data structures to six cognitive process dimensions in Bloom's revised taxonomy (Naps *et al.*, 2002):

1. **Remembering**: Students know the names of data structures (e.g. array, tree, graph, heap etc.) and the names of the algorithms (e.g. insertion sort, quicksort).
2. **Understanding**: They know the steps of algorithms; they are able to explain the algorithms using images and words. They are able to rewrite the algorithms in the same programming language they learned; they are able to run and test the programs. They are able to understand and repeat the analysis of the best and worst case of the algorithms.
3. **Applying**: Students are able to apply the previously learned algorithms for solving similar problems, in different programming environments, or using special input data. They are able to carry out the analysis of the best and worst case of simple algorithms.

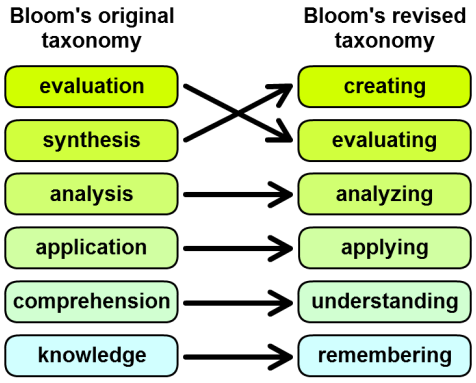

Fig. 3. Bloom's original and revised taxonomy of educational objectives.

4. **Analyzing**: Students understand the differences and relations of algorithms solving the same or similar problems. They are able to support their arguments and/or prove the correctness of algorithms. They are able to analyze more complex problems, identify important objects needed for solutions and divide the problems into smaller, manageable problems.
5. **Evaluating**: Students are able to discuss the advantages and disadvantages of different algorithms solving same or similar problems. They are able to think about how to modify or combine the algorithms to solve new, more complex problems.
6. **Creating**: Students are able to solve complex problems, where different data structures, algorithms and techniques are needed to use simultaneously.

Bloom emphasized that it is important to acquire the knowledge connected to these activities gradually, from the first cognitive process dimension (remembering) to the last one (creating). The algorithm animations may be used successfully especially in the first two cognitive process dimensions (remembering, understanding). Despite this fact, it is recommended to use animations in education, since students may understand the basics quicker and deeper. Later, it would be easier for them to reach higher cognitive process dimensions and more time would remain for those activities in the classrooms (Grissom *et al.*, 2003).

2.4. *Interactivity*

Interactivity is another important factor in the field of educational animations. Many research results suggest that animations are considerably helpful in teaching and learning algorithms only if students are active participants of visualized processes (Furcy *et al.*, 2008; Grissom *et al.*, 2003; Hundhausen and Douglas, 2000; Hundhausen *et al.*, 2002; Kuk *et al.*, 2012; Naps *et al.*, 2002; Patwardhan and Murthy, 2015; Stoffa, 2003). Visualized simulation experiment has to be planned, controlled and implemented in order to serve to the acquisition and discovery of new knowledge for the user – the student – based on their own observations (Stoffa, 2004).

Hundhausen *et al.* (2002) compared the results of 21 experiments: nine of the experiments were focusing on the graphical design of the animations while twelve experiments were focused on the interactivity of the animations. The meta-analysis showed that only 33% of the results focused on the graphical design were significant; however, among results focusing on the interactivity of the animations, 83% were significant. The study also emphasizes that students' active participation is usually more important in the learning process than the graphical design of the animations. This observation is in line with the theory of constructive learning (Hundhausen and Douglas, 2000; Hundhausen *et al.*, 2002; Naps *et al.*, 2002).

The level of interactivity in animations may differ: from a simple observation, through a modification of animated objects to a development of one's own animations. Animations with a low level of interactivity are focusing on the behaviorist-style of learning while animations with a high level of interactivity result in a higher conceptual and procedural learning (Patwardhan and Murthy, 2015). Especially the second learning styles lead to cognitive processes and active learning (Urquiza-Fuentes and Velazquez-

Iturbide, 2013). These examinations also emphasize that interactivity should have a major role in educational animations and visualizations.

Students' participation in visualization processes may differ as well: viewing, responding, changing, constructing or presenting (Furcy *et al.*, 2008; Grissom *et al.*, 2003). The connection between these types of participation is shown in Fig. 4; the viewing fills the whole space in the Venn diagram since this activity is present in every other activity (Naps *et al.*, 2002).

Regarding the interactivity of computer science algorithm animations, there were made following recommendations:

- **The control of the animations should be flexible.** Except the controls for starting and stopping the animations, it is suggested to add control buttons for stepping forwards or backwards in the visualization (Fleischer and Kucera, 2002; Naps and Grissom, 2002; Naps *et al.*, 2002). Mayer also emphasized the importance of the possibility to stop the animation. The default time between the logically related parts of the visualized processes may not be enough for every student: some students need more time to think over and comprehend the steps of the algorithms. Even better solution is when the animations automatically stop after few logically related steps. Students can think of the visualized processes and continue observing the next steps of the animations by pressing a control button. In this case, students do not have to think about the moments, when it is worth to stop the animations, thus they can concentrate more on the visualized processes (Hansen *et al.*, 2002; Mayer, 2009).
- **The speed of the animations should be varying, or should the user be able to change the speed.** Different parts of the animations require different speed, e.g. in sorting algorithms the most important parts are the comparisons and swaps so the animations should be displayed in a slower speed, or they should be stopped during these steps. When we try to watch the control variables of cycles during the sorting algorithms, the first few changes of these variables are the most important to understand the main ideas of the algorithms so later the animations can be displayed at a higher speed (Fleischer and Kucera, 2002). It is also a good solution if students can change the speed of the animations.
- **Modifying or changing the input data in the animations helps students to better understanding of the behavior of the algorithms.** Entering their own input data encourages students to participate more actively. By modifying data in the

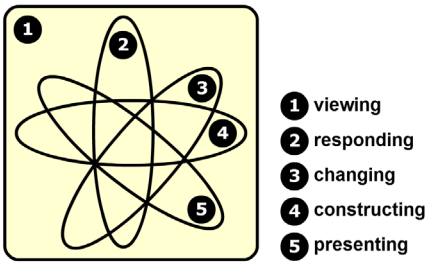

Fig. 4. Possible overlapping of different types of participations in visualized processes.

animations, students can experiment with the visualization and observe different behaviors of the algorithms (Fleischer and Kucera, 2002; Furcy *et al.*, 2008; Hansen *et al.*, 2002; Naps *et al.*, 2002). The results of pedagogical experiments show that students who may enter own input data or modify data in the animations get significantly better results in tests (Hundhausen *et al.*, 2002). Experimenting with the animations may be even more intriguing when not only the input data can be modified, but the values of the variables can be changed during the animations.

- **Animations should adapt to students' knowledge level, or different animations of the same algorithms are recommended to use.** For novice students, it might be hard to understand the algorithms, if the animations are too detailed, they contain many windows, or there are too many options to be set up. For beginners, it is better to use simple animations with predefined data sets. However, for advanced students, it might be valuable if they can enter their own data, modify some options, or observe detailed views of the animations (Fleischer and Kucera, 2002; Naps *et al.*, 2002).
- **Questioning students during the animations might be useful.** Asking questions related to the steps of the animations encourages students to pay more attention (Fleischer and Kucera, 2002; Furcy *et al.*, 2008; Hansen *et al.*, 2002; Naps *et al.*, 2002). On the other hand, examinations during the animations might distract attention from the visualized processes. This is the reason why it is important to choose the right moment and the right form of the questions. Students may be asked by the animation software, they can get questions on papers, in voice by a lecturer, or the questions can be part of electronic textbooks containing the animations (Grissom *et al.*, 2003). To get the answers to the questions is not important in every case, sometimes it is enough if students start thinking about the possible solutions (Hansen *et al.*, 2002).
- **Animations should be entertaining.** Students learn easier if the animations entertain them from the beginning until the end. It is not practical to repeat consecutively the same, long steps (Fleischer and Kucera, 2002). During their experiments, Rudder *et al.* (2007) inserted game elements into the animations, e.g. game activities like "spot the error", "predict the output" and "sort in order". All these activities improve students' level of critical thinking.

2.5. *Types of Algorithm Animations*

Hansen *et al.* (2002) proposes to use three different types of animations successively. All three animations should demonstrate the same algorithm in a different form. The recommended types of animations are following:

1. The first types of animations show the main ideas of the algorithms – they do not go into the details. On the one hand, these animations establish the understanding of abstract elements using real-life examples, on the other they serve as a motivation (Bernát, 2014; Hansen *et al.*, 2002). Real-life examples help students to understand and apply abstract concepts in learning programming and algorithms – in

this way students are encouraged to connect real life with logical programming from the beginning when they only start to be familiar with the first algorithms (Rudder *et al.*, 2007). Examples of these types of animations can be found at our website: `http://anim.ide.sk/sortingcards.php`.

2. The second types of animations are the micro-level animations. These animations go into the details and show the operations made during the execution of the algorithms. Micro-level animations use a small data set of 6 to 8 elements to demonstrate the algorithms. The animations usually contain pseudocode or source code of the algorithms, where the lines of the actual steps are highlighted (Hansen *et al.*, 2002). Examples of these types of animations can be found at `http://anim.ide.sk/sorting_algorithms_1.php` web page.
3. The third types of animations are macro-level animations, where 40–50 elements are used to demonstrate the algorithms. These animations illustrate the features related to the effectiveness of the algorithms, many of the details of the algorithms are hidden (Hansen *et al.*, 2002). Examples of this type of animations can be found at `http://www.sorting-algorithms.com/` website.

In the next sections of this paper, we focus on the first type of animations. In our interactive animations, we use playing cards to demonstrate the essential aspects of non-recursive sorting algorithms.

In summary, to develop effective educational animations, it is important to take into consideration the principles of multimedia learning. The appropriate usage of images, texts, and sounds can reduce extraneous processing and increase essential and generative processing (Mayer, 2009). Furtermore, visualizations should be enriched by interactivity. Educational animations become efficient only when students are engaged with the visualizations beyond passively viewing them (Furcy *et al.*, 2008; Grissom *et al.*, 2003; Hundhausen *et al.*, 2002). Finally, interactive animations for teaching and learning should be based on pedagogical considerations (Lee and Rossling, 2010). It is important to acquire the knowledge gradually following Bloom's revised taxonomy (Anderson *et al.*, 2001). In algorithm animations, it is recommended to use tree types of animations successively: animations with conceptual view, detailed view, and populated view (Hansen *et al.*, 2002). During the last 30–35 years, there were many suggestions regarding the graphical design and interactivity of educationally effective algorithm animations, which we tried to summarize in this literature review, and use them for developing our interactive algorithm animations.

3. Materials and Methods

Principles of multimedia learning, recommendations for the design of algorithm animations, educational perspective, and interactivity providing the student activity are very important factors for the creation of educational visualized simulation models. Our many years of experiences in implementing and using interactive animation and didactic simulation models in education, strongly confirm these facts and are consistent with the results of above-cited authors.

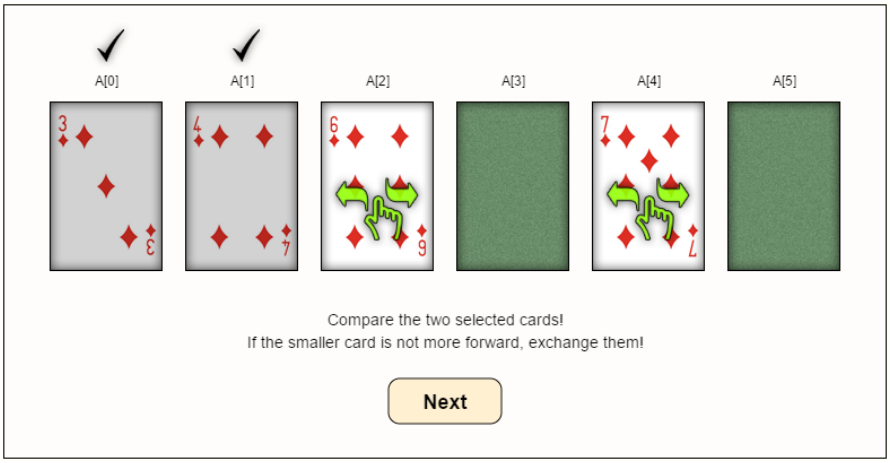

Fig. 5. Interactive game-based animation of simple exchange sort algorithm.

Considering the research results from literature review and our own experiences, we tried to develop interactive playing card animations to illustrate the main ideas and differences between some of the non-recursive sorting algorithms. The collection contains five algorithm animations: simple exchange sort (see Fig. 5), bubblesort, insertion sort, minsort and maxsort. The animations are available at `http://anim.ide.sk/sortingcards.php` web page. Our goal was to create game-based animations with a high level of interactivity where students have to sort the cards in ascending order by using drag-and-drop operations, but strictly following the rules of the sorting algorithms.

The animations were developed in HTML5 using JavaScript technologies. We also used the CreateJS libraries (`www.createjs.com`) for animating the objects. For a better understanding of algorithms, we used the same color for all cards, which was selected from four different colors at the beginning of the animation. Cards J, Q, K, and A were not used to avoid any confusion.

3.1. *Participants*

In the experiment, there were involved all 92 first-year computer-science students of J. Selye University who have attended the "Introduction to programming and algorithms" course during the academic year 2014/15 and 2015/16. Our goal was to determine if students can recognize the essential aspects of illustrated sorting algorithms and the main differences between them.

Most of the students learned about the sorting algorithms before the experiment in high school or itself, but some students did not know anything about the sorting algorithms. All of the students solved programming problems using arrays before the experiment, e.g. finding the minimum or maximum, counting selected elements, adding the values of elements together, mirroring the array. They were also able to read and understand pseudocode.

3.2. *Procedure*

Because students had different knowledge about the sorting algorithms, we asked students to fill in a pre-test before the experiment. Afterwards, they had 1 hour to experiment with the game-based sorting algorithm animations and fill in a post-test. Because there were students with no previous knowledge about the sorting algorithms, we asked them to mark only those answers in both tests they knew. Thus, we tried to diminish the number of students' guesses.

3.3. *Data collection instruments*

Both paper questionnaires (pre-test and post-test) contained a same table (see Fig. 6), where students had to decide which statement-algorithm combinations are true, marking the corresponding cells in the table (for easier referring to the cells, we added letters to the columns and numbers to the rows – these marks were not included in students' tests).

In the second part of the pre-test and post-test, students were asked to match names of the sorting algorithms to their pseudocodes. Using this assignment we tried to determine whether students were able to understand the sorting algorithms in more details.

	A. Simple exchange sort	***B. Bubblesort***	***C. Insertion sort***	***D. Selection sort: Minsort***	***E. Selection sort: Maxsort***
1. The algorithm always compares two neighboring elements in the array.					
2. The algorithm compares every element with all elements located behind it.					
3. First, the algorithm chooses one element from the unsorted part; next, the algorithm exchanges the selected element with the first or last element of the unsorted part.					
4. In the unsorted part of the array, **the smallest element** is always moved to **the beginning** (the sorted sequence is starting to form in the beginning of the array).					
5. In the unsorted part of the array, **the largest element** is always moved to **the end** (the sorted sequence is starting to form in the end of the array).					
6. Elements in the sorted part of the array (in the beginning or the end of the array) **are not modified (not moved)** during the sorting.					
7. Elements in the sorted part of the array (in the beginning or at the end of the array) **can be modified (moved)** during the sorting.					

Fig. 6. Test for understanding the main differences between the sorting algorithms.

4. Results and Discussion

After the experiment, we counted the number of correctly and incorrectly marked answers in every cell of the table in Fig. 6. The differences in the number of marks during the pre-test and post-test are shown in Table 1 (the right answers are marked with "X", the disputable answers are marked with "?").

After the experiment, we noticed that there were two disputable statements (C1, C4) for insertion sort algorithm:

- C1: "The algorithm always compares two neighboring elements in the array". This statement is true for simple insertion sort algorithm. However, it is false for the improved insertion sort algorithm. Because the animation focuses only on the main idea of the sorting algorithm, it does not define which type of insertion algorithm students have to think about.
- C4: "In the unsorted part of the array, the smallest element is always moved to the beginning (the sorted sequence is starting to form in the beginning of the array)". The first part of the statement is not true for the insertion sort algorithm because the smallest element is not moved to the beginning of the unsorted part, but it is inserted into the right place in the sorted part. However, the second part of the statement in the parenthesis it true, because the sorted sequence is starting to form in the beginning of the array.

Because either true or false answers may be acceptable for C1 and C4 algorithm-statement combinations, we did not take into account the results of these cells.

Fig. 7 shows the percentage of correctly marked true algorithm-statement combinations and incorrectly marked false algorithm-statement combinations. The number of correctly marked true algorithm-statement combinations increased by 53.1% during the experiment, from 645 marks (54.7%) to 1001 marks (83.7%). This increase suggests that interactive game-based animations helped students to recognize the main ideas of the sorting algorithms. The graph also indicates that the number of incorrectly marked false algorithm-statement combinations decreased by 39.7% during the experiment, from 335 marks (18.2%) to 202 marks (11.0%).

To determine, whether these changes are significant or not, we made several paired sign tests. We have chosen sign tests instead of paired-samples t-tests or Wilcoxon

Table 1

Differences in the number of marked algorithm-statement combinations during the pre-test and post-test

	A	B	C	D	E
1st statement:	–46	+18 **X**	–4 **?**	–5	–4
2nd statement:	+28 **X**	–12	+14	–7	–10
3rd statement:	+14	–5	–5	+21 **X**	+15 **X**
4th statement:	+40 **X**	–4	+26 **?**	+8 **X**	–1
5th statement:	+4	+48 **X**	+4	–4	+1 **X**
6th statement:	+33 **X**	+33 **X**	–16	+34 **X**	+32 **X**
7th statement:	–13	–18	+36 **X**	–11	–8

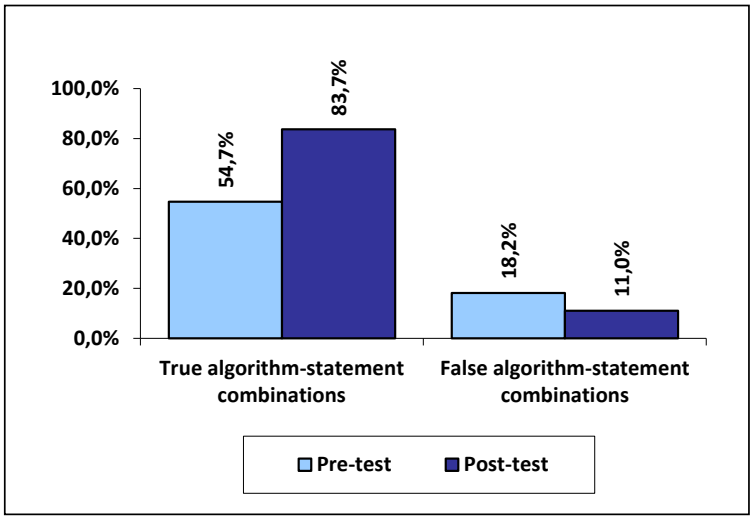

Fig. 7. Percentage of the correctly marked true algorithm-statement combinations and incorrectly marked false algorithm-statement combinations in pre-test and post-test.

signed-rank tests because the assumptions of the latter tests were violated – the data were not normally nor symmetrically distributed.

First, we measured the increase in the correctly marked true algorithm-statement combinations. Out of the 92 participants involved in the experiment, 81 students marked more true algorithm-statement combinations, 5 students marked fewer true algorithm-statement combinations, and 6 students marked the same number of algorithm-statement combinations in the post-test compared to the pre-test. Overall, participants marked more true algorithm-statement combinations in the post-test (median: 11 marks) than in the pre-test (median: 7 marks), the statistically significant increase in the median of the differences is 4 marks, $z = 8.087$, $p < 0.0005$.

Next, we tried to measure the decrease in the incorrectly marked false algorithm-statement combinations. Out of the 92 participants involved in the experiment, 61 students marked fewer false algorithm-statement combinations, 23 students marked more algorithm-statement combinations, and 8 students marked the same number of algorithm-statement combinations in the post-test compared to the pre-test. Overall, participants marked fewer false algorithm-statement combinations in the post-test (median: 2 marks) than in the pre-test (median: 4 marks), the statistically significant decrease in the median of the differences is –1 marks, $z = –4.037$, $p < 0.0005$.

These first results prove that interactive card animations helped students to understand the main ideas of sorting algorithms. This finding supports many research studies that recommend the usage of animations with conceptual view and real-life objects for establishing the understanding of abstract elements (Bernát, 2014; Hansen *et al.*, 2002; Rudder *et al.*, 2007).

After these positive results, we examined students' answers more deeply. Fig. 8 shows the percentage of correctly marked true statements (green columns) and incor-

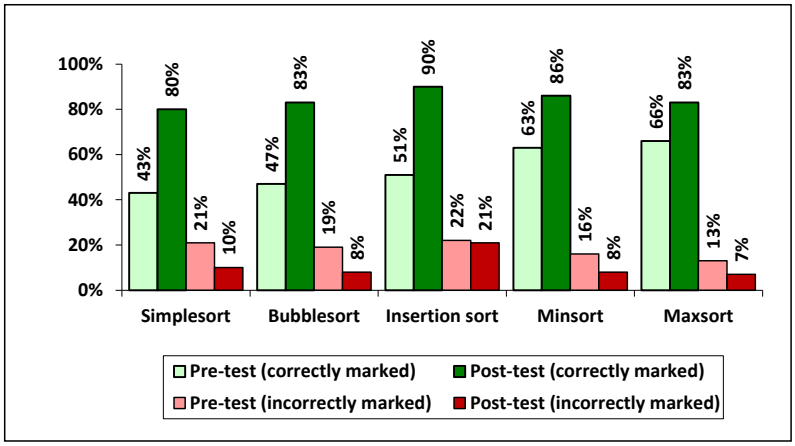

Fig. 8. Percentage of the correctly marked true algorithm-statement combinations (green) and incorrectly marked false algorithm-statement combinations (red) in pre-test and post-test for every sorting algorithm.

rectly marked false statements (red columns) in the pre-test and post-test for every sorting algorithm.

To determine if these changes in the correctly marked true statements (green columns in Fig. 8) are significant, we made several paired sign tests. The results are shown in Table 2. We can see that there are significant increases of correctly marked true algorithm-statement combinations.

We made also paired sign tests to determine if the changes in the incorrectly marked false statements (red columns in Fig. 8) are significant. The results are shown in Table 3. We can see that there are some significant results for incorrectly marked false algorithm-statement combinations.

Table 2

Results of the sign tests (correctly marked true algorithm-statement combinations)

	Total N	Positive Differences (post-test – pre-test)	Negative Differences (post-test – pre-test)	Number of Ties (post-test – pre-test)	Median in Pre-test	Median in Post-test	Median of Differences	Standardized Test Statistic	Asymptotic Sig. (2-sided test)
Simplesort	92	65	11	16	1	3	1	6.080	< 0.0005
Bubblesort	92	67	6	19	1	3	1	7.022	< 0.0005
Insertion sort	92	41	5	46	1	1	0	5.160	< 0.0005
Minsort	92	49	8	35	2	3	1	5.298	< 0.0005
Maxsort	92	45	11	36	2	3	0	4.410	< 0.0005

Table 3
Results of the sign tests (incorrectly marked false algorithm-statement combinations)

	Total N	Positive Differences (post-test – pre-test)	Negative Differences (post-test – pre-test)	Number of Ties (post-test – pre-test)	Median in Pre-test	Median in Post-test	Median of Differences	Standardized Test Statistic	Asymptotic Sig. (2-sided test)
Simplesort	92	15	43	34	1	0	0	–3.545	< 0.0005
Bubblesort	92	11	39	42	1	0	0	–3.818	< 0.0005
Insertion sort	92	26	26	40	1	1	0	0.000	1.000
Minsort	92	10	31	51	0	0	0	–3.123	0.002
Maxsort	92	10	29	53	0	0	0	–2.882	0.004

Next, we examined the number of marks for every true algorithm-statement combination in pre-test and post-test (see Fig. 9). We can see on the chart that students marked more correct answers in post-test than in pre-test.

Similarly, we examined the number of marks for every false algorithm-statement combination in pre-test and post-test (see Fig. 10). As we can see on the graph, students marked fewer incorrect answers in post-test than in pre-test. However, for algorithm-statement combinations A3 and C2, we can see increases of incorrect marks, and the decrease for algorithm-statement combination C3 is not as expected.

A3 is the following statement for simple exchange sort algorithm: “First, the algorithm chooses one element from the unsorted part; next, the algorithm exchanges the selected element with the first or last element of the unsorted part”.

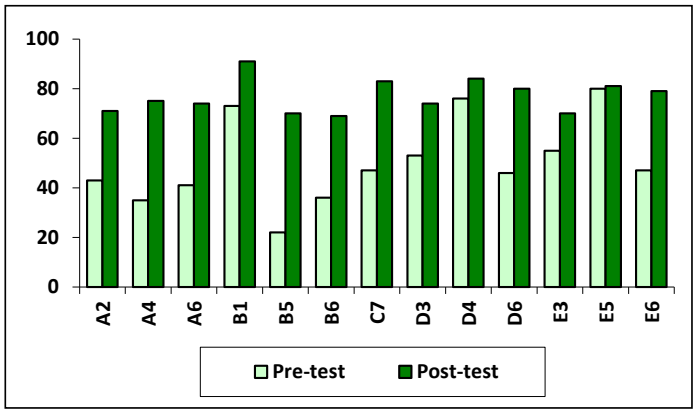

Fig. 9: Number of correct marks in pre-test and post-test for every true algorithm-statement combination.

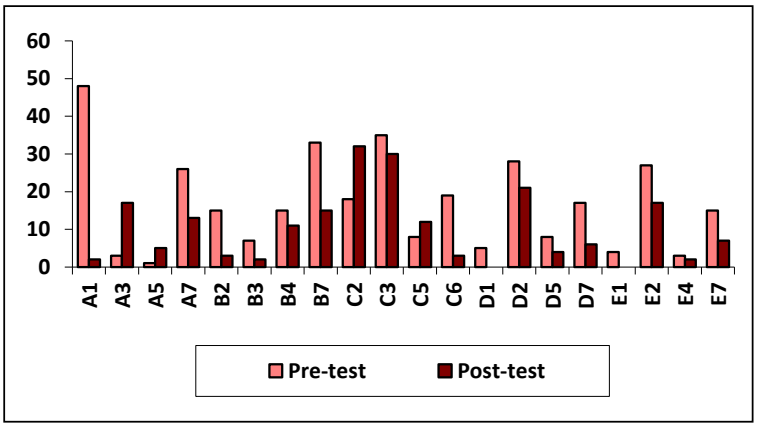

Fig. 10: Number of incorrect marks in pre-test and post-test for every false algorithm-statement combination.

C2 is the following statement for insertion sort algorithm: "The algorithm compares every element with all elements located behind it".

C3 is the following statement for insertion sort algorithm: "First, the algorithm chooses one element from the unsorted part; next, the algorithm exchanges the selected element with the first or last element of the unsorted part".

All these statements are false for the given algorithms, but the statements might seem to be true if someone does not think them over. The reason why students marked these statements true might be that they did not think in detail. However, thinking in detail was not the goal of these animations. For understanding the algorithms in depth, micro-level animations are recommended to use (Bernát, 2014; Hansen *et al.*, 2002).

In the last part of the experiment, we wanted to determine if students comprehended the algorithms in detail. For this reason, we used an assignment where participants had to match the names of the algorithms to their pseudocodes. The percentages of correctly assigned pseudocodes to the algorithms are shown in Table 4.

Because we got dichotomous data in these assignments (0 = incorrectly paired pseudocode to the algorithm, 1 = correctly paired pseudocode to the algorithm), we used McNemar's tests instead of the sign tests. The results did not show any significant changes (simplesort: $N = 88$, $\chi^2(1) = 2.370$, $p = 0.124$; bubblesort: $N = 89$, $\chi^2(1) = 0.000$,

Table 4

Percentages of correctly marked pseudocodes to algorithms

	Simple exchange sort	Bubblesort	Insertion sort	Selection sort: Minsort	Selection sort: Maxsort
pre-test	64%	58%	66%	68%	67%
post-test	72%	58%	67%	67%	66%

p = 1.000; insertion sort: N = 87, exact p = 1.000; minsort: N = 88, exact p = 1.000; maxsort: N = 89, exact p = 1.000).

This result also supports the fact, that animations with conceptual view are not sufficient to learn the sorting algorithms in detail. In the educational process, micro-level animations with detailed view should follow our interactive card animations, where the loop control variables, pseudocode and other details of the algorithm are visualized (Végh, 2016). The usage of animations with different level of details is also recommended by (Hansen *et al.*, 2002).

5. Conclusion

In conclusion, results show that interactive card animations helped students to understand the algorithms. They were able to recognize the main ideas of sorting algorithms, but they did not understand the algorithms in detail. The sign tests showed that participants were able to mark significantly more correct algorithm-statement combinations, and fewer incorrect algorithm-statement combinations in post-tests than in pre-tests. However, in the second part of the assignments, where students had to pair algorithm names to their pseudocodes, the McNemar's tests did not show any significant changes. For understanding the algorithms in-depth, more detailed, micro-level animations should follow the animations presented in this paper. Thus, students – after recognizing the essential aspects and differences between the sorting algorithms – can easier start learning algorithms in detail (Bernát, 2014; Hansen *et al.*, 2002; Stoffa, 2004).

In our study, we proved that the interactive card animations with conceptual view could be successfully used to understand the main aspects of basic sorting algorithms and recognize the differences between them. However, we do not know yet, to what extent they helped to acquire knowledge about the sorting algorithms compared to other teaching methods and other educational materials – this could be a possible topic for our future research.

References

Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., . . . Wittrock, M.C. (2001). *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. USA: Addison Wesley Longman, Inc.

Bellstrom, P., Thoren, C. (2009). Learning how to program through visualization: A pilot study on the bubble sort algorithm. *2009 Second International Conference on the Applications of Digital Information and Web Technologies (Icadiwt 2009)*, 90–94. doi:10.1109/icadiwt.2009.5273943

Bernát, P. (2014). The methods and goals of teaching sorting algorithms in public education. *Acta Didactica Napocensia,* 7(2), 10.

Bloom, B.S., Englehart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R. (1956). *The Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain* (B.S. Bloom Ed.). New York: David McKay Company, Inc.

Byrne, M.D., Catrambone, R., Stasko, J.T. (1999). Evaluating animations as student aids in learning computer algorithms. *Computers & Education,* 33(4), 253–278. doi:10.1016/s0360-1315(99)00023-8

Esponda-Arguero, M. (2010). Techniques for visualizing data structures in algorithmic animations. *Information Visualization,* 9(1), 31–46.

Fleischer, R., Kucera, L. (2002). Algorithm animation for teaching. *Software Visualization,* 2269, 113–128.

Furcy, D., Naps, T., Wentworth, J. (2008). Sorting Out sorting – the sequel. In: *Iticse '08: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*. 174–178.

Grissom, S., McNally, M.F., Naps, T. (2003). Algorithm visualization in CS education: comparing levels of student engagement. Paper presented at the *Proceedings of the 2003 ACM Symposium on Software Visualization*, San Diego, California.

Hansen, S., Narayanan, N.H., Hegarty, M. (2002). Designing educationally effective algorithm visualizations. *Journal of Visual Languages and Computing,* 13(3), 291–317. doi:10.1006/s1045-926x(02)00027-7

Hundhausen, C., Douglas, S. (2000). Using visualizations to learn algorithms: Should students construct their own, or view an expert's? In: *2000 Ieee International Symposium on Visual Languages, Proceedings*, 21–28.

Hundhausen, C.D., Douglas, S.A., Stasko, J.T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing,* 13(3), 259–290. doi:10.1006/s1045-926x(02)00028-9

Kann, C., Lindeman, R.W., Heller, R. (1997). Integrating algorithm animation into a learning environment. *Computers & Education,* 28(4), 223–228. doi:10.1016/s0360-1315(97)00015-8

Katai, Z., Tóth, L. (2010). Technologically and artistically enhanced multi-sensory computer-programming education. *Teaching and Teacher Education,* 26(2), 244–251. doi: `http://dx.doi.org/10.1016/j.tate.2009.04.012`

Kehoe, C., Stasko, J., Taylor, A. (2001). Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies,* 54(2), 265–284. doi:10.1006/ijhc.2000.0409

Kuk, K., Jovanovic, D., Jokanovic, D., Spalevic, P., Caric, M., Panic, S. (2012). Using a game-based learning model as a new teaching strategy for computer engineering. *Turkish Journal of Electrical Engineering and Computer Sciences,* 20, 1312–1331. doi:10.3906/elk-1101-962

Lattu, M., Meisalo, V., & Tarhio, J. (2003). A visualisation tool as a demonstration aid. *Computers & Education,* 41(2), 133–148. doi:10.1016/s0360-1315(03)00032-0

Lee, M.-H., Rossling, G. (2010). Integrating categories of algorithm learning objective into algorithm visualization design: a proposal. In: *Iticse 2010: Proceedings of the 2010 Acm Sigcse Annual Conference on Innovation and Technology in Computer Science Education*. 289–293.

Mayer, R.E. (2009). *Multimedia Learning* (second ed.). New York, USA: Cambridge University Press.

Naps, T., Grissom, S. (2002). The effective use of quicksort visualizations in the classroom. *J. Comput. Sci. Coll.,* 18(1), 88–96.

Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., . . . Velázquez-Iturbide, J.Á. (2002). Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.,* 35(2), 131–152. doi:10.1145/782941.782998

Patwardhan, M., Murthy, S. (2015). When does higher degree of interaction lead to higher learning in visualizations? Exploring the role of 'Interactivity Enriching Features'. *Computers & Education,* 82, 292–305. doi:10.1016/j.compedu.2014.11.018

Rudder, A., Bernard, M., & Mohammed, S. (2007). Teaching programming using visualization. In: *Proceedings of the Sixth IASTED International Conference on Web-Based Education*, 487–492.

Stoffa, V. (2003). Computer-aided learning of programming. Paper presented at the *Proceedings of the 4th International Conference on Computer Systems and Technologies: e-Learning*, Rousse, Bulgaria.

Stoffa, V. (2004). Modelling and simulation as a recognising method in the education. *Educational Media International,* 41(1), 51–58.

Urquiza-Fuentes, J., Velazquez-Iturbide, J.A. (2013). Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. *Computers & Education,* 67, 178–192. doi:10.1016/j.compedu.2013.02.013

Végh, L. (2011). *Animations in Teaching Algorithms and Programming (Animácie vo vyučovaní algoritmov a programovania)*. Paper presented at the Nové technologie ve vzdělávání, Olomouc, CZ.

Végh, L. (2016). Javascript library for developing interactive micro-level animations for teaching and learning algorithms on one-dimensional arrays. *Acta Didactica Napocensia,* 9(2), 23–32.

L. Végh is a PhD candidate in the Faculty of Informatics at Eötvös Loránd University in Budapest (Hungary). He received his Mgr. degree in Teaching Mathematics and Informatics from the Constantine the Philosopher University in Nitra (Slovakia) in 1999, and his PaedDr. degree in Teaching Informatics from the J. Selye University in Komárno (Slovakia) in 2008. Between 2000 and 2005 he worked as a teacher of mathematics and informatics in primary education, since 2005 he works as an assistant lecturer at J. Selye University in Komarno (Slovakia). His main research interests include the usage of algorithm animations, simulations, game-based learning, and virtual worlds in computer science education. He is author of about 70 publications (articles in journals, proceedings of conferences, parts of text books, etc.).

V. Stoffová is a university professor at Faculty of Education, Trnava University in Trnava (Slovakia) and has a part time job at Palacký University in Olomouc (Czech Republic). She received her Ing. degree (1974) and PhD. degree (1982) in Technical cybernetics at Slovak Technology University in Bratislava. Degree of dr. hab. which she received in Technical cybernetics at Army Academy in Brno (Czech Republic, 1984) and in Teaching mathematics at Constantine the Philosopher University in Nitra (2000). She received the degree of university professor after an inauguration process in Teaching mathematics at the last-mentioned university (2003). She worked as university teacher at Slovak Technology University in Bratislava (1974–1981), at Army Academy in Liptovský Mikulá (1981–1987), at Constantine the Philosopher University in Nitra (1997–2005) and at J. Selye University in Komárno (2004–2015). Her main research interests are oriented towards the computer modelling, simulation and visualisation of systems in different fields of science. She is author or co-author of more than 250 publications (articles in reviewed journals, proceedings of conferences, chapters in monographs, text books, etc.).