**14/1** (2016), 45–62
DOI: 10.5485/TMCS.2016.0415
tmcs@science.unideb.hu
http://tmcs.math.unideb.hu

**Teaching Mathematics and Computer Science**

# An interactive animation for learning sorting algorithms: How students reduced the number of comparisons in a sorting algorithm by playing a didactic game

Ladislav Végh and Veronika Stoffová

*Abstract.* Learning programming and understanding algorithms is one of the hardest tasks for novice computer science students. One of the basic algorithms they learn during the introductory programming and algorithms courses are the sorting algorithms. Students like learning these and other algorithms by animations and didactic games, however, these animations are not educationally useful in every case. In this article, we present our educational sorting game, which can be used to introduce the topic of sorting algorithms. The didactic game can be used later too, as a demonstrative tool for explaining the more efficient, quicksort algorithm. We conducted a pedagogical experiment, in which we examined the process of development of sorting algorithms by students while they used the mentioned didactic game. The results showed that students were able to create an algorithm to solve the sorting problem, and they improved its effectiveness by reducing the number of comparisons in the algorithm. They were also able to understand the importance of the efficiency of algorithms when we demonstrated them the quicksort algorithm using the same tool after the experiment.

*Key words and phrases:* teaching algorithms, game-based animation.

*ZDM Subject Classification:* U60.

## 1. Introduction

Programming and understanding algorithms are some of the fundamental skills that computer science students need to learn. Teaching and learning basic algorithms can be easier if the teaching and learning process is enhanced by the usage of visualizations, animations, and interactive didactic games. Students prefer learning materials containing animations and games [2], [6], [9], [11].

This assumption was also proved by our survey, where we asked computer science students at J. Selye University, Komárno, Slovakia during the winter semester (winter term) of the academic year 2014/15 to choose one of the forms of materials they prefer for learning a new algorithm. The options were as follows:

- Learning material that contains animation of an algorithm; e.g. a web page with interactive animation or YouTube video.

- Learning materials with static images, e.g. a web page or book that contains series of images representing the key steps of an algorithm.

- Learning materials with textual explanations, e.g. a web page or book that contains a detailed description of the steps of an algorithm.

56 students were involved in the questionnaire investigation. The results showed that significantly more students (85.7%) preferred to use learning materials containing animations than learning materials containing images or textual explanations, $\chi^2(2, \text{N}=56) = 70.107$, $p < 0.0005$. Only 12.5% of students preferred static images, and only 1.8% choose textual explanation.

However, previous studies in the field of using animations in education showed that they are not educationally effective in every case [6], [7], [8]. In spite of these results, using animations in education can be in most of the situation motivating for students, and they can help students to concentrate on the main steps of the algorithm. This slight help can be especially important in the modern, media-rich world, where students get used to distractions by advertisements, images, and videos in the media [21], [5]. After the mixed results, researchers started to investigate further, in which cases are animations and visualizations effective in education. Many previous research works suggest that the interactivity is important in educational animations, in some situations even more, than the graphical representation of the algorithm [6], [8], [5], [15], [14], [17], [16]. Well designed and interactive animations can even reduce the high dropout at universities [18].

We collected some of the recommendations related to the graphical design and interactivity of the algorithm animations for teaching. The following recommendations were used for developing our game-based animation presented in this paper.

- Because algorithms use abstract concepts, it is important to select the right model. The model usually represents the data structure used in the algorithm [3], [4]. Figure 1 shows some of the possible representations of the one-dimensional array. They can be represented by the heights of the columns, brights of the ball, or by the numbers on the playing cards [19].

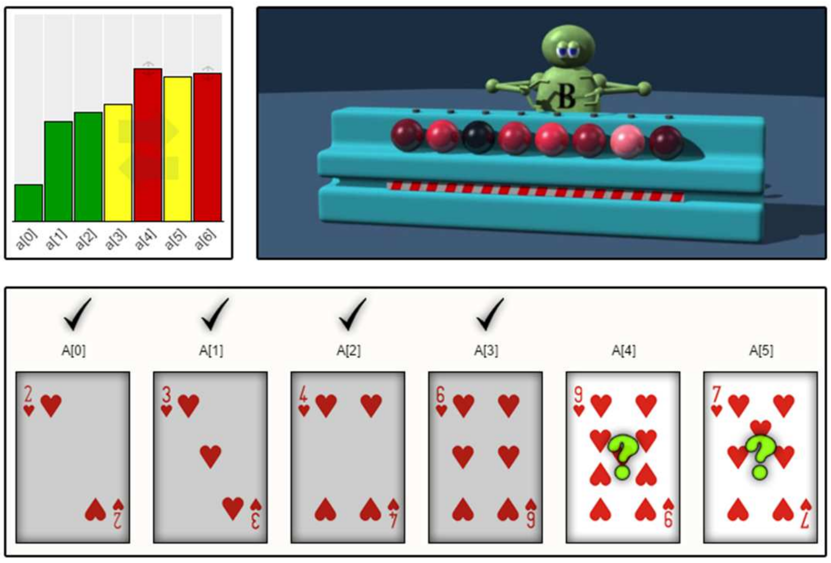


*Figure 1.* Different visual representations of the one-dimensional array

- The animated algorithm should be represented on a small data set, usually on 6-8 elements [4].
- The animation should contain explanations of visualized processes. The explanations can convey the information in the textual form or voice. When the textual form is used, it is important to give enough time for students to read the explanations, because according to the cognitive theory of the multimedia learning educants can not concentrate on the reading of the text and understanding of the animation at the same time [12]. The explanation does not necessarily need to be part of the animation, for example, it can be shown right above the animation on a web page or in an electronic textbook, or the teacher can explain the animation in the classroom [4].

- The sound effects and the colors of the objects should carry information too [4] - e.g. in a sorting algorithm, the sorted part of the array might have a green color while the unsorted part of the array red color.
- The control of the animation should be flexible. This can be simply reached by adding buttons to stop/play the animation, step the algorithm forward/backward [14], [4], [13]. It is better to play the animation in small logical parts than to play the whole animation continuously. When the animation stops after each step, students have time to think about the visualized processes [6], [12].
- Animations should be entertaining. Adding game elements into the animations motivates students and enhances their critical thinking [15].

One of the main topics during the introductory programming and algorithms courses are sorting algorithms, which can be taught by plenty of methods. At primary school, experimenting with real-life objects of different weight may facilitate students' understanding of the sorting algorithms [1]. Dance, music, rhythm and role-playing may also be used to learn the sorting algorithms, involving the whole body and multiple senses into the learning process [10]. Sorting algorithms may also be taught and explained by interactive animations. For novice programmers, it is better to use simple visualizations with fewer details. Later, animations which display all the steps seem to be adequate for understanding the algorithms more deeply [6], [1].

Simple exchange sort is probably the first algorithm which students learn. Usually this algorithm is followed by other sorting algorithms with time complexity $O(n^2)$, like selection sort, insertion sort, and bubblesort. Finally, students learn sorting algorithms with time complexity $O(n.\log_2 n)$, like quicksort and mergesort.

In the main part of this paper, we deal with some of these sorting algorithms in education, especially with the original and the improved versions of the bubblesort, and quicksort algorithm.

## 2. Materials and methods

In this section, we present our interactive box-sorting animation - an online didactic game (`http://anim.ide.sk/sortingboxes.php`) - which can be used to introduce the topic of the sorting algorithms, and demonstrate the main steps of the quicksort algorithm. Using this didactic game we conducted a pedagogical experiment. Our goal was to figure out if first-year computer science students with no previous knowledge about the efficiency of sorting algorithms can reduce the number of comparisons in the algorithms, which they developed on their own.

As a model, we have chosen a group of boxes without labels or any other markers (see Figure 2: (a)). Students have to sort these boxes from the lighter box to the heavier, but they can compare only two boxes at the same time, similarly as a sorting algorithm compares only two elements of the array at the same time. An equal-arm scale is available in the game to compare these boxes.

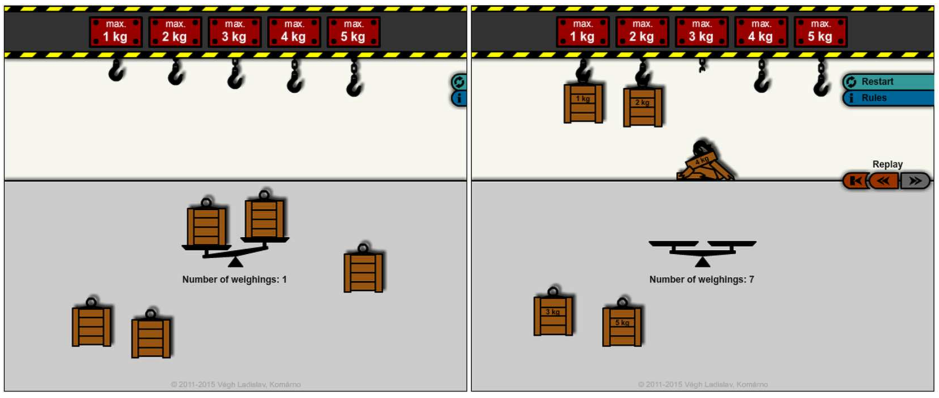


*Figure 2.* (a) Sorting boxes by their weight and (b) Possibility of going through the steps of animation after an unsuccessful try

The task of game is successfully solved, when students put every box to a corresponding hook. However, if they put a heavier box on a hook that cannot carry it, the hook will break, and the game will end unsuccessfully (see Figure 2: (b)). In this case, the weights of boxes and a control panel are shown. Using the control panel student can go through the whole animation and figure out where they made the mistake.

The simple didactic game has three levels. The first level contains only three boxes; during this easy level, students can get familiar with the game and its goal. The second, medium level contains five boxes; during this level, students start developing their sorting algorithms. The final, third level contains seven boxes; during this level, students can improve and verify their sorting algorithms on more elements. After solving all three levels, a short statistics is shown for students, in which they can see how many comparisons they made and how many of them were redundant. The latest ones are e.g. when they compared the same boxes more times. Another unnecessary comparison is when the place of the box in the sorted order can be figured out from other comparisons. E.g. if box A is lighter than box B, and box B is lighter than box C, then it is logically deducible that box A is also lighter than box C, so comparing box A with C is unnecessary.

The didactic game was created in Adobe Flash CS5 using Actionscript 3.0 scripting language. The game can be easily embedded into HTML pages, e.g. into electronic textbooks. Furthermore, we included two features for research purposes into the application:

- To minimize the possibility of solving the didactic game by luck, we created a specific algorithm. The weights of the boxes are not generated at the beginning of the game as someone would expect, but they are generating gradually during the game as the user compares the boxes. If the user does not make all the necessary comparisons, the algorithm will generate the worst case, and the game ends unsuccessfully. This algorithm was explained in details in the publication [20].
- Because we wanted to examine students' solutions, we saved the number of comparisons and the number of redundant comparisons into a MySQL database. On the web page, an ID number, which was different for every user, was shown under the game. During the experiment, we asked our students to write down this number on the questionnaire instead of their name. Thus, the questionnaire was anonymous, and we were able to connect the answers from the questionnaire to the saved steps of the solutions in the database and filter them.

The pedagogical experiment was conducted during the academic year 2014/15 and 2015/16. Our goal was to figure out if students can develop their sorting algorithms and improve them by reducing the number of comparisons. Therefore, we encouraged our students to find an algorithm, where the number of redundant comparisons is zero. This kind of algorithm is, e.g. the quicksort, or mergesort, where the time complexity is $O(n.\log_2 n)$. However, the insertion sort algorithm with time complexity $O(n^2)$ also does not make any redundant comparisons, and the bubblesort algorithm can also be improved by reducing the number of comparisons, as we will see in students' solutions.

In the pedagogical experiment were involved 99 first-year computer science students from J. Selye University in Komárno, Slovakia, (50 students in 2014/15 and 49 students in 2015/16). They were asked to play the didactic game and fill out an anonymous questionnaire during a 1-1.5 hour period. Some of the students were familiar with basic sorting algorithms, but none of them learned about the efficiency of the algorithms, or sorting algorithms with time complexity O(n.$\log_2$n) before the experiment.

## 3. Results and discussion

During the experiment, every student found a solution to sort the boxes in an ascending order. As we can see in Figure 3, some of the students sorted the boxes only with redundant comparisons, but most of the students were able to reduce the number of unnecessary comparisons during the experiment and solved the task without redundant comparisons in the end. There were only a few students who did not make any unnecessary comparison since the beginning until the end of the experiment.

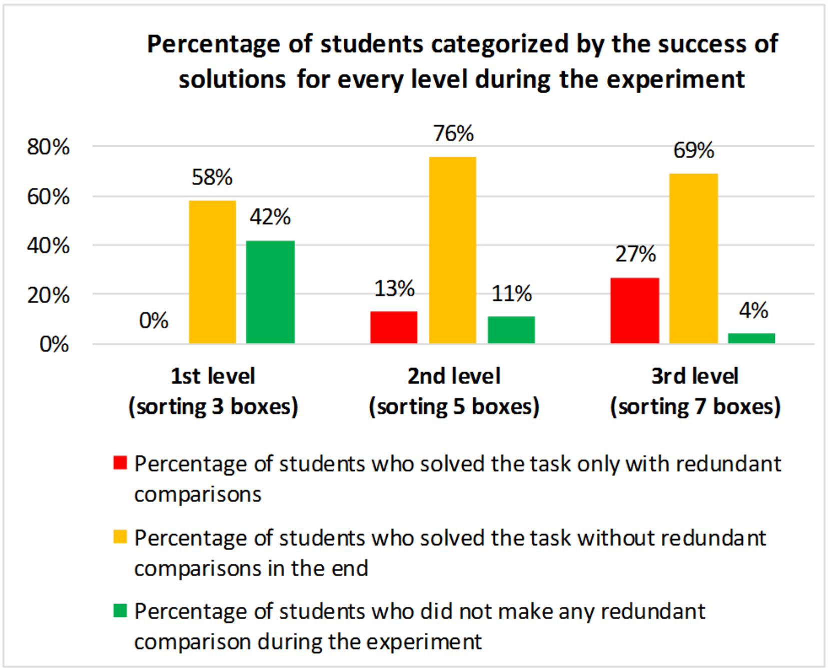


*Figure 3.* Percentage of students categorized by the success of solution for every level during the experiment

Next, we tried to examine if students were able to reduce the number of redundant comparisons in their successful solutions during the experiment, or they were even able to reduce them to zero. Table 1 shows the average numbers and medians of unnecessary comparisons in students' first six successful solutions for the first and second level of the didactic game.

*Table 1.* Statistics of numbers of redundant comparisons in students' first six successful solutions for the first and second level of the game

| | Level 1 (sorting 3 boxes) | | | | Level 2 (sorting 5 boxes) | | | |
|---|---|---|---|---|---|---|---|---|
| | Mean | Std. Dev. | Mdn. | N | Mean | Std. Dev. | Mdn. | N |
| Attempt1 | 0.62 | 1.251 | 0 | 99 | 2.52 | 2.894 | 1 | 99 |
| Attempt2 | 0.16 | 0.397 | 0 | 99 | 1.80 | 2.134 | 1 | 98 |
| Attempt3 | 0.16 | 0.467 | 0 | 99 | 1.33 | 1.706 | 0 | 97 |
| Attempt4 | 0.06 | 0.246 | 0 | 94 | 1.05 | 1.615 | 0 | 91 |
| Attempt5 | 0.11 | 0.313 | 0 | 83 | 0.70 | 1.174 | 0 | 80 |
| Attempt6 | 0.05 | 0.229 | 0 | 73 | 0.84 | 1.542 | 0 | 68 |

Figure 4 shows the mean numbers of unnecessary comparisons in students' successful solutions during the first level (sorting three boxes) and second level (sorting five boxes). The figure shows some decreases during the observed periods.

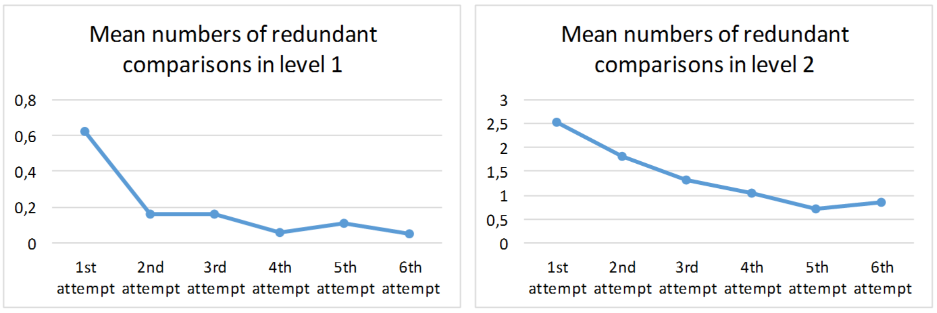


*Figure 4.* Mean numbers of redundant comparisons in students' first six successful solutions during the first and second level of the didactic game

Table 2 and Figure 5 shows the mean numbers and medians of redundant comparisons in students' successful solutions during the third level (sorting seven boxes). We can recognize more decrease in these graphs, than in the previous two charts. To determine if the decrease in the numbers of redundant comparisons is significant, we were using related-samples Friedman's two-way analysis of variance by ranks. We decided to use this test instead of one-way repeated measures ANOVA, because the assumption of normality was violated.

*Table 2.* Statistics of numbers of redundant comparisons in students' first six successful solutions for the third level of the game

| | Level 3 (sorting 7 boxes) | | | |
|---|---|---|---|---|
| | Mean | Std. Dev. | Mdn. | N |
| Attempt1 | 6.87 | 5.452 | 7 | 99 |
| Attempt2 | 4.80 | 4.815 | 3 | 97 |
| Attempt3 | 3.89 | 4.503 | 2 | 92 |
| Attempt4 | 3.48 | 4.435 | 2 | 83 |
| Attempt5 | 2.92 | 4.574 | 0 | 72 |
| Attempt6 | 2.41 | 3.836 | 1 | 61 |

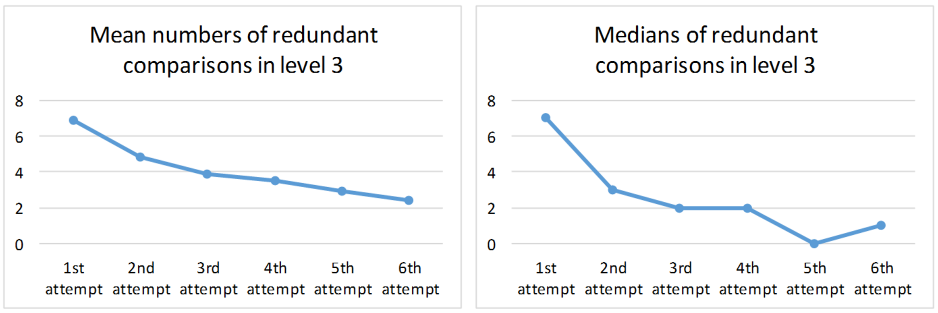


*Figure 5.* Mean numbers and medians of redundant comparisons in students' first six successful solutions during the third level of the didactic game

Friedman test showed that the number of redundant comparisons in students' first six successful solutions for the third level was statistically significantly different at different attempts during the pedagogical experiment, $\chi^2(5) = 35.560$, $p < 0.0005$. Pairwise comparisons (SPSS Statistics, 2013) with Bonferroni correction for multiple comparisons (see Table 3) revealed statistically significant differences in the number of redundant comparisons between the 1st attempt (Mdn = 7) and 5th attempt (Mdn = 0) ($p = 0.001$), 1st attempt (Mdn = 7) and

6th attempt (Mdn = 1) ($p < 0.0005$), 2nd attempt (Mdn = 3) and 5th attempt (Mdn = 0) ($p = 0.027$), and 2nd attempt (Mdn = 3) and 6th attempt (Mdn = 1) ($p = 0.004$).

*Table 3.* The results of the post hoc tests for third level (sorting seven boxes)

| | | Test Statistics | Std. Error | Std. Test Statistics | Sig. | Adj. Sig. |
|---|---|---|---|---|---|---|
| Pair 1 | 6th - 5th attempt | 0.180 | 0.339 | 0.532 | 0.594 | 1.000 |
| Pair 2 | 6th - 4th attempt | 0.598 | 0.339 | 1.766 | 0.077 | 1.000 |
| Pair 3 | 6th - 3rd attempt | 0.852 | 0.339 | 2.516 | 0.012 | 0.178 |
| Pair 4 | 6th - 2nd attempt | 1.238 | 0.339 | 3.654 | 0.000 | 0.004 |
| Pair 5 | 6th - 1st attempt | 1.508 | 0.339 | 4.452 | 0.000 | 0.000 |
| Pair 6 | 5th - 4th attempt | 0.418 | 0.339 | 1.234 | 0.217 | 1.000 |
| Pair 7 | 5th - 3rd attempt | 0.672 | 0.339 | 1.984 | 0.047 | 0.709 |
| Pair 8 | 5th - 2nd attempt | 1.057 | 0.339 | 3.121 | 0.002 | 0.027 |
| Pair 9 | 5th - 1st attempt | 1.328 | 0.339 | 3.920 | 0.000 | 0.001 |
| Pair 10 | 4th - 3rd attempt | 0.254 | 0.339 | 0.750 | 0.453 | 1.000 |
| Pair 11 | 4th - 2nd attempt | 0.639 | 0.339 | 1.887 | 0.059 | 0.887 |
| Pair 12 | 4th - 1st attempt | 0.910 | 0.339 | 2.686 | 0.007 | 0.109 |
| Pair 13 | 3rd - 2nd attempt | 0.385 | 0.339 | 1.137 | 0.255 | 1.000 |
| Pair 14 | 3rd - 1st attempt | 0.656 | 0.339 | 1.936 | 0.053 | 0.794 |
| Pair 15 | 2nd - 1st attempt | 0.270 | 0.339 | 0.798 | 0.425 | 1.000 |

These results prove our interactive animation helped students to develop sorting algorithms and make them more efficient by reducing the number of comparisons to the minimum.

We also wanted to know what kind of algorithms students used during the pedagogical experiment and how they reduced the number of comparisons. For this reason, we tried to observe the student's solutions during the experiment and asked students in the questionnaire to write down the whole process of sorting.

The following videos demonstrate some ideas, how the didactic game can be completed:

(1) *Using bubblesort algorithm:* `https://youtu.be/47Ikl5wyRMM`

During the sorting, we put the boxes on the equal-arm scale one after the other. After each comparison, we leave the heavier box on the scale, and we remove the lighter box. We put the removed box on the left free place of the two neighboring free spots (the heavier would be placed to the right free place according to the bubblesort algorithm, but we leave it on the scale for the next weighing with the next element). Using this algorithm the heaviest box will remain on the scale in the end. After this, we can repeat the whole process from the beginning with the remaining boxes.

(2) *Using bubblesort with marking:* `https://youtu.be/YAf1Rgj0z2A`
Similar algorithm than the previous one, but when we remove a box from the scale, we try to pay attention to the results of the previous comparisons. If the lastly removed box is heavier than the previously removed box, we put it next to it. If we cannot determine which of the two previously removed boxes is heavier, we mark the last box by putting it above the previous box, so we will know that this marked box needs to be compared with other previously sorted boxes. After finding the heaviest box from all boxes, we do not need to compare every box again, but we need to find only the right places of the marked boxes. Even though this algorithm is complicated because of the marking (grouping); there are not redundant comparisons during the sorting.

(3) *Using quicksort algorithm:* `https://youtu.be/eqO1x4Fj5RM`
In this method, we use the quicksort algorithm, so we will choose one box and compare it with every other box. If the compared box is lighter than the selected box, we put the compared box into the left group, otherwise to the right group. After comparing the selected box with every other box, we will find the right place of the selected box, and we will repeat the whole process with both (left and right) groups.

(4) *Using mergesort algorithm:* `https://youtu.be/zLpZWiYT7Fo`
In this method, we use the mergesort algorithm. First, we create groups of 2 boxes. Next, we merge these groups, and we create groups of 4 boxes. Lastly, we create a group of 8 boxes by merging (in the didactic game only seven boxes).

(5) *Merging groups of 3-4 boxes:* `https://youtu.be/FWdqbmSs1kw`
First, we create sorted groups of 3 elements without redundant comparisons. Next, we insert the remaining element into one of the groups. Finally, we merge the two sorted groups.

According to our observations during the pedagogical experiment, most of the students started to sort the boxes by using the first method (bubblesort), but they quickly figured out that there are too many redundant comparisons, which they can logically deduct from the previous comparisons. After this, they tried to minimize the number of comparisons in some way, so they come to the second method of sorting (bubblesort with marking).

There were some students, who solved the problem in the end with the third method (quicksort) or fifth method (merging groups of 3-4 boxes), but none of the students have tried the fourth method (mergesort).

Students described in the questionnaire the algorithm they used. After reviewing these descriptions, we assign to every answer one of the above-mentioned methods. From the experiment we can see, that the significant majority of the students used the second method to sort the boxes (bubblesort with marking): $\chi^2(3, N=70) = 25.657, p < 0.0005$ (see Figure 6).

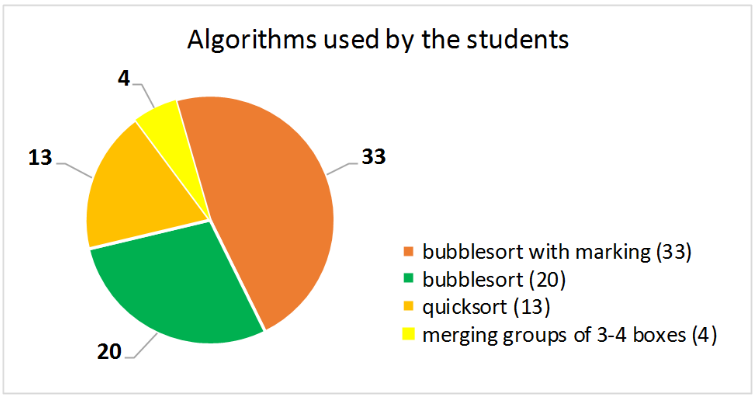


*Figure 6.* Sorting algorithms used by the students

It is important to mention that the bubblesort algorithm improved by the students makes fewer comparisons than the original bubblesort or the well-known improved bubblesort algorithm, and it does not contain any redundant comparison (which can be logically deduced from the previous comparisons). However, the number of swaps was not reduced, so practically it makes the same number of exchanges than the original bubblesort algorithm. These data are illustrated in the following example (see Table 4), where we sorted 10000 random numbers by using different versions of the bubblesort algorithm and the quicksort algorithm.

The questionnaire also contained six after-game assignments, which are in the appendix of this paper. Using these assignments we wanted to determine if students can solve these logical tasks, which are closely related to the identification of the redundant comparisons. Most of the students were able to solve the assignments after the experiment; the results are shown in Table 5.

*Table 4.* Sorting 10000 random numbers by different algorithms

| | number of comparisons | number of exchanges |
|---:|---:|---:|
| original bubblesort | 49995000 | 24837078 |
| improved bubblesort | 49940418 | 24837078 |
| students' improved bubblesort | 24847071 | 24837078 |
| quicksort | 105265 | 33469 |

*Table 5.* Efficiency of students

| 1st assignment | 2nd assignment | 3rd assignment | 4th assignment | 5th assignment | 6th assignment |
|:-:|:-:|:-:|:-:|:-:|:-:|
| 96% | 95% | 88% | 96% | 84% | 75% |

An interesting future research would be to determine that the didactic game in what extent helped to solve these types of logical tasks.

Finally, we wanted to know students' opinion about the educational game, so we asked them to rate the clarity, user-friendliness, graphical design, and give us an overall rating of the didactic tool by using a 10-point Likert scale. The results are summarized in Table 6.

*Table 6.* Students' ratings on the 10-point Likert scale

| | Clarity | User-friendliness | Graphical design | Overall rating |
|---:|:-:|:-:|:-:|:-:|
| Mean: | 8.53 | 9.49 | 9.44 | 9.14 |
| Std. Dev.: | 1.60 | 1.10 | 1.19 | 1.31 |

Students liked our didactic game, and they gave it high ratings. Some of the students also mentioned that they would like to learn more algorithms by animations and educational games.

## 4. Conclusions and plans

The results showed that our didactic game helped students to develop their sorting algorithms. They were also able to reduce the number of comparisons in algorithms significantly. However, most of them only improved the bubblesort algorithm, so they were not able to recognize (figure out) a more efficient algorithm, like quicksort or mergesort.

Students were also able to solve logical tasks related to the redundant comparisons after experimenting with the didactic tool, and rated the game with high scores.

During the education we recommend to use this tool before the sorting algorithms topic, as we did in our pedagogical experiment; but it can be also used as a demonstrative tool for explaining the quicksort algorithm. Students can easier understand the importance of effectiveness in algorithms when they first try to develop their algorithms by experimenting with the didactic tool, and later we show them the quicksort algorithm by using the same tool. Students who were not able to develop the quicksort algorithm during the pedagogical experiment were amazed how easily and quickly the boxes can be sorted when we showed them the quicksort algorithm later.

This pedagogical experiment also proves that interactive animations and didactic games have its place in education, and they can be used to support the learning process. This fact encouraged us to create an open online portal of algorithm animations and visualizations (`www.algoanim.ide.sk`). We have started to collect and develop more interactive animations and didactic games for teaching and learning algorithms, which will be available for everyone on this portal.

## References

[1] P. Bernát, The methods and goals of teaching sorting algorithms in public education, *Acta Didactica Napocensia* **7**, no. 2 (2014), 10.

[2] M. D. Byrne, R. Catrambone and J. T. Stasko, Evaluating animations as student aids in learning computer algorithms, *Computers & Education* **33**, no. 4 (1999), 253–278.

[3] M. Esponda-Arguero, Techniques for visualizing data structures in algorithmic animations, *Information Visualization* **9**, no. 1 (2010), 31–46.

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

[5] S. Grissom, M. F. McNally and T. Naps, Algorithm visualization in CS education: comparing levels of student engagement, in: *Proceedings of the 2003 ACM symposium on Software visualization*, ACM, 2003, 87–94.

[6] S. Hansen, N. H. Narayanan and M. Hegarty, Designing educationally effective algorithm visualizations, *Journal of Visual Languages and Computing* **13**, no. 3 (2002), 291–317.

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

[8] C. D. Hundhausen, S. A. Douglas and J. T. Stasko, A meta-study of algorithm visualization effectiveness, *Journal of Visual Languages and Computing* **13**, no. 3 (2002), 259–290.

[9] C. Kann, R. W. Lindeman and R. Heller, Integrating algorithm animation into a learning environment, *Computers & Education* **28**, no. 4 (1997), 223–228.

[10] Z. Katai and L. Toth, Technologically and artistically enhanced multi-sensory computer-programming education, *Teaching and Teacher Education* **26**, no. 2 (2010), 244–251.

[11] C. Kehoe, J. Stasko and A. Taylor, Rethinking the evaluation of algorithm animations as learning aids: an observational study, *International Journal of Human–Computer Studies* **54**, no. 2 (2001), 265–284.

[12] R. E. Mayer, *Multimedia Learning*, second edition, Cambridge University Press, New York, USA, 2009.

[13] T. Naps and S. Grissom, The effective use of quicksort visualizations in the classroom, *J. Comput. Sci. Coll.* **18**, no. 1 (2002), 88–96.

[14] T. L. Naps, G. o. Rößling, V. Almstrum, W. Dann, R. Fleischer, Ch. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger and J. Á Velázquez-Iturbide, Exploring the role of visualization and engagement in computer science education, *SIGCSE Bull.* **35**, no. 2 (2002), 131–152.

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

[16] V. Stoffa, Modelling and simulation as a recognising method in the education, *Educational Media International* **41**, no. 1 (2004), 51–58.

[17] V. Stoffa, Az animáció szerepe az elektronikus tankönyvekben, *Információs társadalom* **VIII**, no. 3 (2008), 113–125.

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

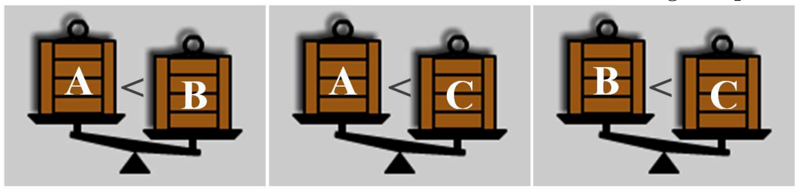
[19] L. Végh, Animations in Teaching Algorithms and Programming (Animácie vo vyučovaní algoritmov a programovania), in: *Nové technologie ve vzdělávání*, (J. Dostál, ed.), Palacký University, Olomouc, CZ, 2011, 47–51.

[20] L. Végh, From bubblesort to quicksort with playing a game (Hravou formou od bublinkového triedenia po rýchle triedenie), in: *XXIX. International Colloquium on the Management of Educational Process*, (J. Neubauer and E. Hájková, eds.), University of Defence, Brno, CZ, 2011, 539–549.

[21] J. R. Young, Homework? What Homework? Students seem to be spending less time studying than they used to, *The Chronicle of Higher Education* **49**, no. 15 (2002), A35–A37.

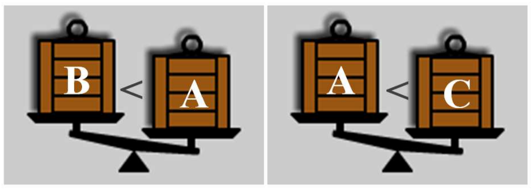
## Appendix - after game assignments

(1) What would be the order of boxes if we had made the following comparisons?



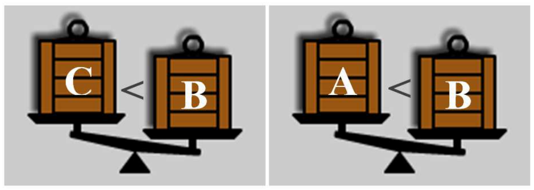
☐ A < C < B
☐ A < B < C
☐ B < A < C
☐ other order: ........................................
☐ there were not enough comparisons to decide the order of boxes

(2) What would be the order of boxes if we had made the following comparisons?



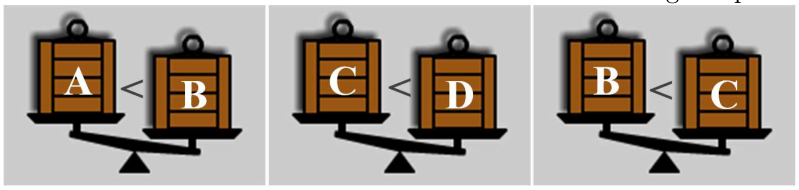
☐ B < C < A
☐ A < C < B
☐ B < A < C
☐ other order: ........................................
☐ there were not enough comparisons to decide the order of boxes

(3) What would be the order of boxes if we had made the following comparisons?



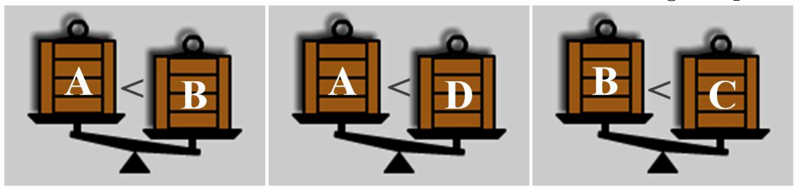
☐ A < B < C
☐ C < A < B
☐ A < C < B
☐ other order: ........................................
☐ there were not enough comparisons to decide the order of boxes

(4) What would be the order of boxes if we had made the following comparisons?



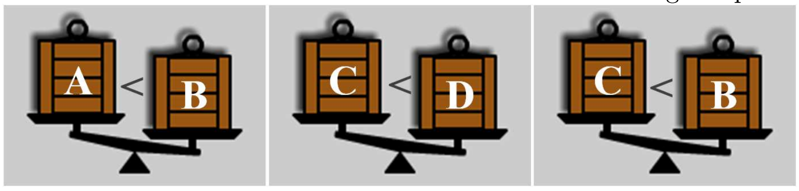
- ☐ A < B < C < D
- ☐ B < C < A < D
- ☐ A < B < D < C
- ☐ other order: .......................................
- ☐ there were not enough comparisons to decide the order of boxes

(5) What would be the order of boxes if we had made the following comparisons?



- ☐ A < B < C < D
- ☐ A < D < B < C
- ☐ A < B < D < C
- ☐ other order: .......................................
- ☐ there were not enough comparisons to decide the order of boxes

(6) What would be the order of boxes if we had made the following comparisons?



- ☐ A < B < C < D
- ☐ A < C < B < D
- ☐ A < C < D < B
- ☐ other order: .......................................
- ☐ there were not enough comparisons to decide the order of boxes

LADISLAV VÉGH
J. SELYE UNIVERSITY
KOMÁRNO, SLOVAKIA

*E-mail:* `veghl@ujs.sk`

VERONIKA STOFFOVÁ
TRNAVA UNIVERSITY
TRNAVA, SLOVAKIA

*E-mail:* `veronika.stoffova@truni.sk`

*(Received February, 2016)*