ACTA DIDACTICA NAPOCENSIA

Volume 7 Number 2, 2014

THE METHODS AND GOALS OF TEACHING SORTING ALGORITHMS IN PUBLIC EDUCATION

Péter Bernát

Abstract: The topic of sorting algorithms is a pleasant subject of informatics education. Not only is it so because the notion of sorting is well known from our everyday life, but also because as an algorithm task, whether we expect naive or practical solutions, it is easy to define and demonstrate. In my paper I will present some of the possible methods and goals of teaching sorting algorithms in the primary and the secondary school context. Some of the applicable demonstration and collaborative methods will be listed and illustrated with examples available on the internet. Next to defining the teaching goals offered by the specific methods, I will also make suggestions which methods are optimal for a given teaching phase.

Key words: primary and secondary school education; sorting algorithms; demonstration method; collaborating method

1. Introduction

The topic of sorting algorithms is a happy subject of informatics education. Not only is it so because the notion of sorting is well known from our everyday life, but also because as an algorithm task, whether we expect naive or practical solutions, it is easy to define and demonstrate.

Public education usually focuses on comparative sorting algorithms. In each of their steps such algorithms determine which of the two given elements has to precede the other in the outcome. Next to less efficient simple algorithms, like selection sort, insertion sort, and bubble sort, efficient algorithms, like quicksort, merge sort, Shell sort, and heapsort, are examples [1, 2].

Two age groups deal with the above topic in school. In early primary school, children learn to implement some of the sorting algorithms and compare them from the perspective of efficiency (still without computers). Secondary school students, however, move on to describing and realizing sorting algorithms with the computer, along with comparing them in detail.

The demonstration method and the collaborative method are plausible ways of teaching sorting algorithms. Demonstration is a tool to present and examine objects, phenomena, and processes, while collaboration is a method to learn and practice an activity [3]. Algorithms can be considered both as processes to scrutinize if their implementer is another person or the computer, and as activities to master if the implementers are ourselves.

Learning programming does not only imply getting familiar with the basic notions of programming. Just like other subjects, programming develops students' thinking capacity [4], along with their social skills. When teaching sorting algorithms, similar goals can be achieved.

The aim of my article is to categorize some of the applicable demonstration methods and collaborative methods, illustrating them with examples available on the internet. Next to defining the teaching goals offered by the specific methods, I will also make suggestions which methods are optimal for a given teaching phase.

Received July 0142

2. Teaching the implementation of sorting algorithms

Primary school kids learn about the basics of the computer, such as sorting algorithms, through games, still without computers [5]. Their task is to understand and execute some of the sorting algorithms, while comparing them regarding their efficiency.

2.1. Demonstration and collaboration with objects

The most suitable way to teach a couple of algorithms to this age group is the use of objects. To sort objects from a given perspective is good because it is close to everyday reality, on the one hand, and, because the activity-based learning is specifically fit for kids, on the other hand.

It is worth to choose objects, which can be lined up according to their weight. Since their comparison is possible only with the use of a two-pan balance, we will pay enough attention on the basic step of comparative settlements, and it will be easier to keep track of the number of comparisons. As objects we can apply identical capsules filled with different amount of coins, and the balances can be borrowed from the chemistry lab. If we glue the appropriate numbers to the bottom of the objects, later the order can be easily checked.

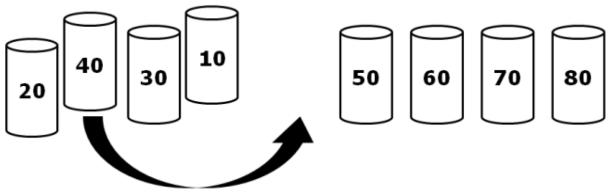


Figure 1. *The method of sorting and selecting weights [5]*

Here is a possible scenario [5]. The students, after entering into groups of four, receive 8 objects and a balance. The teacher tells them that they are supposed to compare the weight of two objects only in each step. As a warm-up task, we can ask them to determine the heavier of two objects and the heaviest of all objects. We can discuss how many steps need to be made to solve the latter task. Then we can ask them to sort 3 objects by weight and, if they manage, to sort all objects.

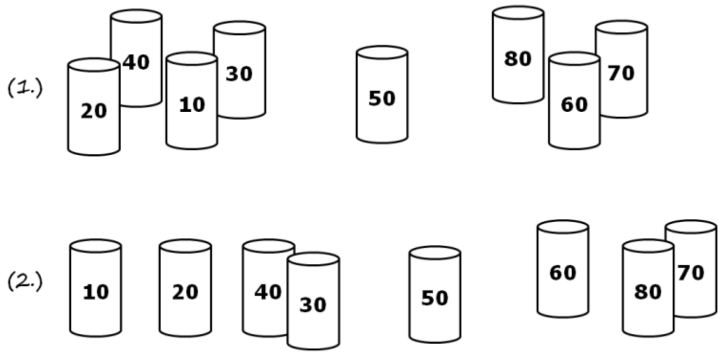


Figure 2. *Sorting weights with quicksort algorithm [5]*

After the experiments, we can teach them a simple sorting algorithm and an efficient algorithm. We can choose for example selection sort (figure 1) or quicksort (figure 2). If the students count the necessary comparisons in the first and the second case, they will realize that the sorting methods are of different efficiency. If there is more interest and time, further algorithms can be demonstrated, such as insertion sort and merge sort.

2.2. Executing educational goals

During these sessions, students get closer to important notions, such as algorithm, sorting (listing elements in a given order), efficiency (the number of steps to solve the same problem), and recursion (”divide and conquer”), without even noticing. Understanding, implementing, and comparing the demonstrated algorithms, that is, the attempts of the students, develop their thinking abilities. What is more, since the tasks are solved in groups, they even improve the social skills of the students.

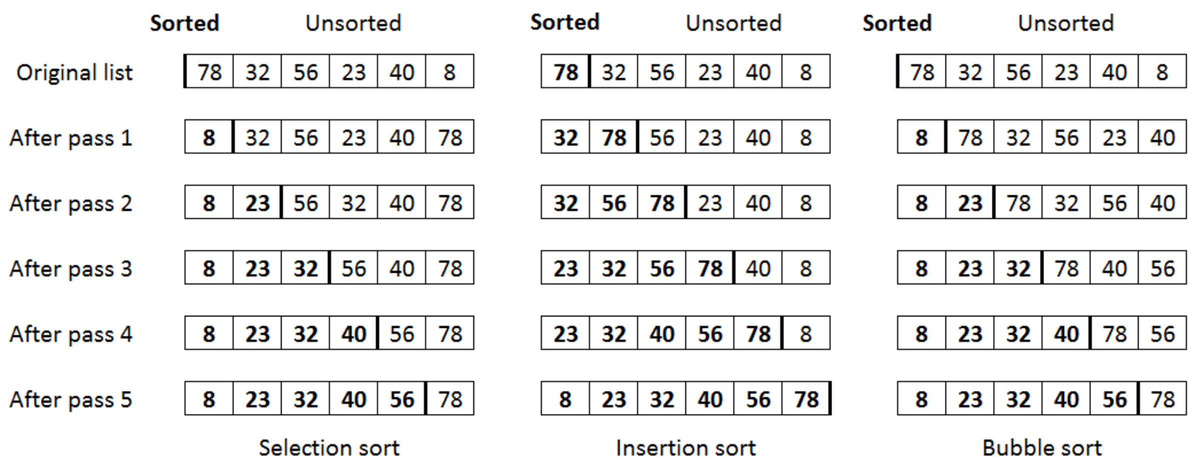
3. Teaching the computer-based execution of sorting algorithms

In secondary school education, special attention is paid to programming frequently used algorithms, such as sorting algorithms. Students are expected to understand, describe, code, and examine the efficiency of different sorting algorithms.

3.1. Demonstration with pictures

Visual demonstrations characteristically serve to make the main steps of the sorting algorithm visible. However, unlike the strategies, the basic steps of the method remain hidden.

The images used for a simple algorithm illustrate the list through the steps of the external loop. It is well visible on figure 3, demonstrating selection sort, insertion sort, and bubble sort together, that the list is divided into an organized and an unorganized part, with an end result of putting each element into the organized section. It can be examined whether the elements change their location before and after they get into the organized part.



Selection sort

	Sorted		Unsorted			
Original list	78	32	56	23	40	8
After pass 1	**8**	32	56	23	40	78
After pass 2	**8**	**23**	56	32	40	78
After pass 3	**8**	**23**	**32**	56	40	78
After pass 4	**8**	**23**	**32**	**40**	56	78
After pass 5	**8**	**23**	**32**	**40**	**56**	78

Insertion sort

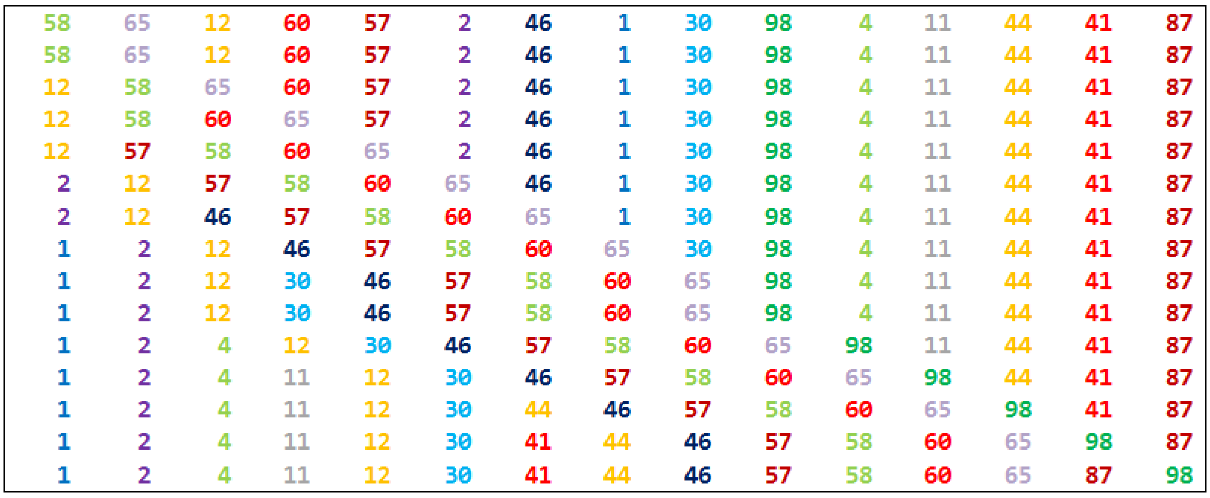
	Sorted		Unsorted			
Original list	**78**	32	56	23	40	8
After pass 1	**32**	**78**	56	23	40	8
After pass 2	**32**	**56**	**78**	23	40	8
After pass 3	**23**	**32**	**56**	**78**	40	8
After pass 4	**23**	**32**	**40**	**56**	**78**	8
After pass 5	**8**	**23**	**32**	**40**	**56**	**78**

Bubble sort

	Sorted		Unsorted			
Original list	78	32	56	23	40	8
After pass 1	**8**	78	32	56	23	40
After pass 2	**8**	**23**	78	32	56	40
After pass 3	**8**	**23**	**32**	78	40	56
After pass 4	**8**	**23**	**32**	**40**	78	56
After pass 5	**8**	**23**	**32**	**40**	**56**	78

Figure 3. *The demonstration of selection, insertion, and bubble sort with pictures [6]*

Pictures similar to these can be made with computer programs as well, which means that even random or special lists can be viewed in the case of different algorithms. It is worth to assign different colors to the different elements so to make movement and idleness conspicuous (figure 4).



58	65	12	60	57	2	46	1	30	98	4	11	44	41	87
58	65	12	60	57	2	46	1	30	98	4	11	44	41	87
12	58	65	60	57	2	46	1	30	98	4	11	44	41	87
12	58	60	65	57	2	46	1	30	98	4	11	44	41	87
12	57	58	60	65	2	46	1	30	98	4	11	44	41	87
2	12	57	58	60	65	46	1	30	98	4	11	44	41	87
2	12	46	57	58	60	65	1	30	98	4	11	44	41	87
1	2	12	46	57	58	60	65	30	98	4	11	44	41	87
1	2	12	30	46	57	58	60	65	98	4	11	44	41	87
1	2	12	30	46	57	58	60	65	98	4	11	44	41	87
1	2	4	12	30	46	57	58	60	65	98	11	44	41	87
1	2	4	11	12	30	46	57	58	60	65	98	44	41	87
1	2	4	11	12	30	44	46	57	58	60	65	98	41	87
1	2	4	11	12	30	41	44	46	57	58	60	65	98	87
1	2	4	11	12	30	41	44	46	57	58	60	65	87	98

Figure 4. *The demostration of insertion sort with a program (own program)*

The pictures made for efficient recursive sorting algorithms depict the lists through the specific levels of the recursion. On figure 5, illustrating quicksort and merge sort, we can see that both algorithms split the initial list into smaller and smaller ones, until we are left with one-item lists, which then get merged several times to form the ordered final list. Notice that for quicksort the process of splitting is more complex than merging, while for merge sort it is the other way around [7].

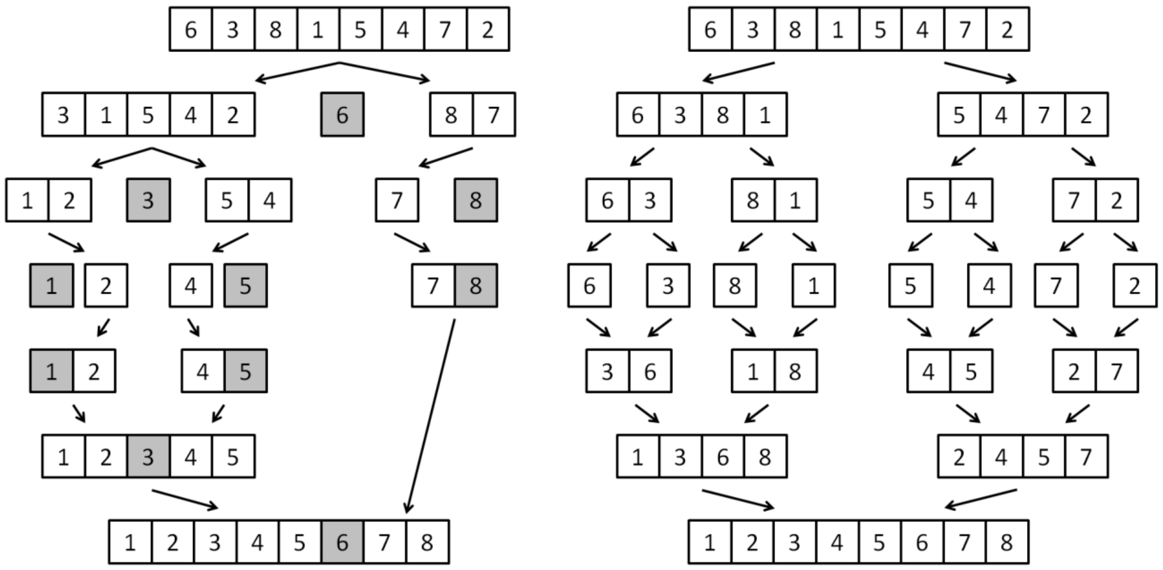


Figure 5. *The demonstration of quicksort (own picture) and merge sort [6] with pictures*

To discover the recursion in these methods, we need to observe the demonstrative pictures not from an up-down angle but rather ”inside out.” For example, the highest level of recursion is given by the first step and last step together. We split the initial list into two smaller lists, we sort them, then, by merging the two parts we arrive to the solution. Recursively, we continue and sort each of the two parts with the same method. As the lowest level of the recursion, we will find the one-item lists, which need no further sorting.

3.2. Demostration with animations

Animations can be divided into two groups. The members of the first group demonstrate the operation of sorting algorithms. While they display each of the basic steps of such algorithms, the essence of the methods remains in the dark.

The operation of the algorithm is easier to follow if we can pause the animation, change its speed, or even reverse it; or if we can highlight the active items with arrows, colors, or other forms of marking. Some of the interactive animations can visualize the operation of the algorithm with random lists, which allows for the display of different cases.

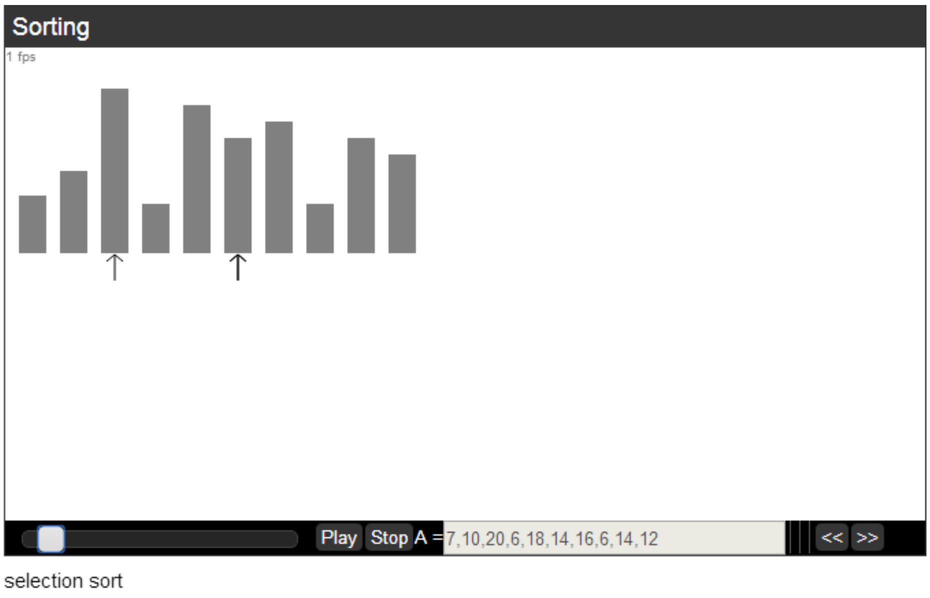


Figure 6. *The demonstration of selection sort with animation [8]*

The webpage with the tell-tale title *Visualization of Data Structures and Algorithms* [8] aims to demonstrate the most frequently used data structures and algorithms through interactive animations of a standard interface. It offers animations for selection sort, bubble sort, quicksort, and merge sort. With the help of the animations, we can sort arbitrary lists (the items are displayed as columns of different height), set the speed of play-mode, and even jump back and forth between the different steps by using a slider (figure 6).

The other group of animations visualizes the efficiency of sorting algorithms by presenting sorting processes next to each other. The compared processes inform us about the efficiency either of one sole sorting algorithm in the context of different lists, or of different sorting algorithms in the context of comparable lists. We can even set the animations to count the comparisons and copies during their operation.

The webpage of *Sorting Algorithm Animations* [9] demonstrates the efficiency of eight sorting algorithms: insertion sort, selection sort, bubble sort, Shell sort, merge sort, heapsort, and two versions of quicksort. The algorithms are checked with four different initial conditions: with random lists, nearly sorted lists, reversed lists, and lists with few unique keys. The 32 animations can be played all at once, but we can also select and observe the behavior of one given sorting algorithm or one given type of lists.

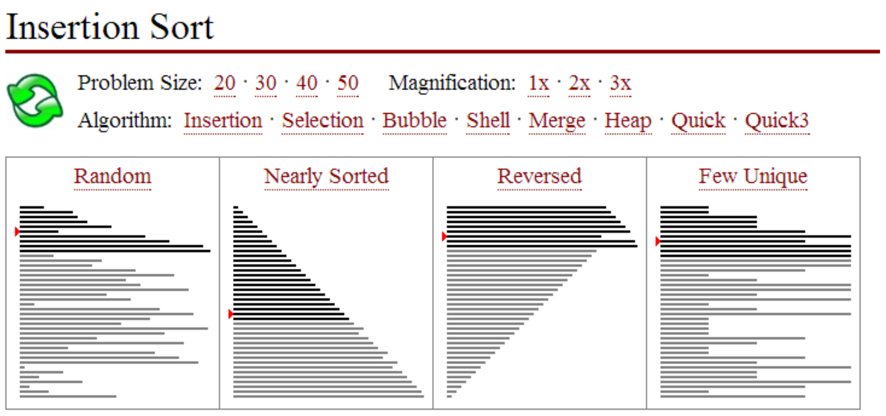

Insertion Sort
Problem Size: 20 · 30 · 40 · 50
Magnification: 1x · 2x · 3x
Algorithm: Insertion · Selection · Bubble · Shell · Merge · Heap · Quick · Quick3
Random
Nearly Sorted
Reversed
Few Unique

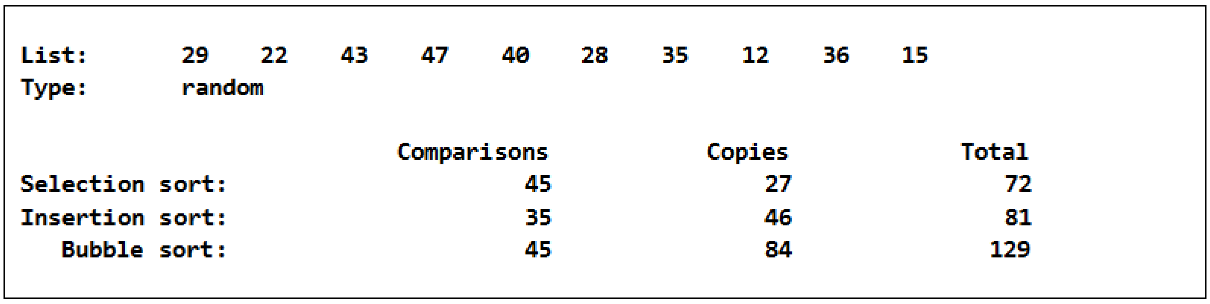
Figure 7. *Demonstration of the efficiency of insertion sort with animation [9]*

Animations presenting the operation of one single sorting algorithm in the context of different lists (figure 7) facilitate a more thorough understanding of the algorithm, on the one hand. On the other, they enable us to examine whether its functioning changes with special lists compared to a random list. It is for example a significant question if a given sorting algorithm can finish with a nearly sorted list earlier.

However, if we use the animations to demostrate the sorting of a specific list by different algorithms, we can easily decide which of the sorting algorithms is the best to use if this given type of list has to be sorted [10]. Suprisingly, the so-called efficient algorithms perform rather poorly with certain lists.

Finally, if we play all 32 animations together, we can draw a useful conclusion, namely, that there is no such thing as best sorting algorithm that would perform best in each of the cases.

Although the program running on the above webpage fails to count the comparisons and copies, we can easily make programs (not neceassarily with animations) that are able to determine these values in the case of different lists or different algorithms. The screenshot in figure 8 shows how a program compares the operation of three simple algorithms with a randomly generated list.



```
List:       29   22   43   47   40   28   35   12   36   15
Type:       random

                         Comparisons          Copies          Total
Selection sort:                   45              27             72
Insertion sort:                   35              46             81
   Bubble sort:                   45              84            129
```

Figure 8. *Measuring the efficiency of selection, insertion, and bubble sort with own program*

3.3. Collaboration with simulation programs

In simulation programs made for sorting algorithms, the algorithms are executed by the users. Although these programs involve less motor and sensory organs compared to real experimentation, the advantages are that they do not require physical tools or preparation, and everyone is able to work on their own. In addition, they can provide extra services next to the usual supervision: the animations can check the progress of the students immediately or even teach them algorithm.

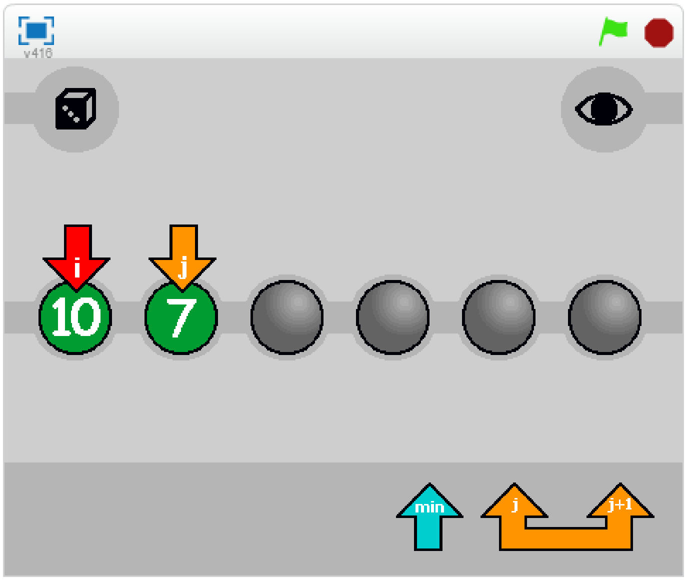
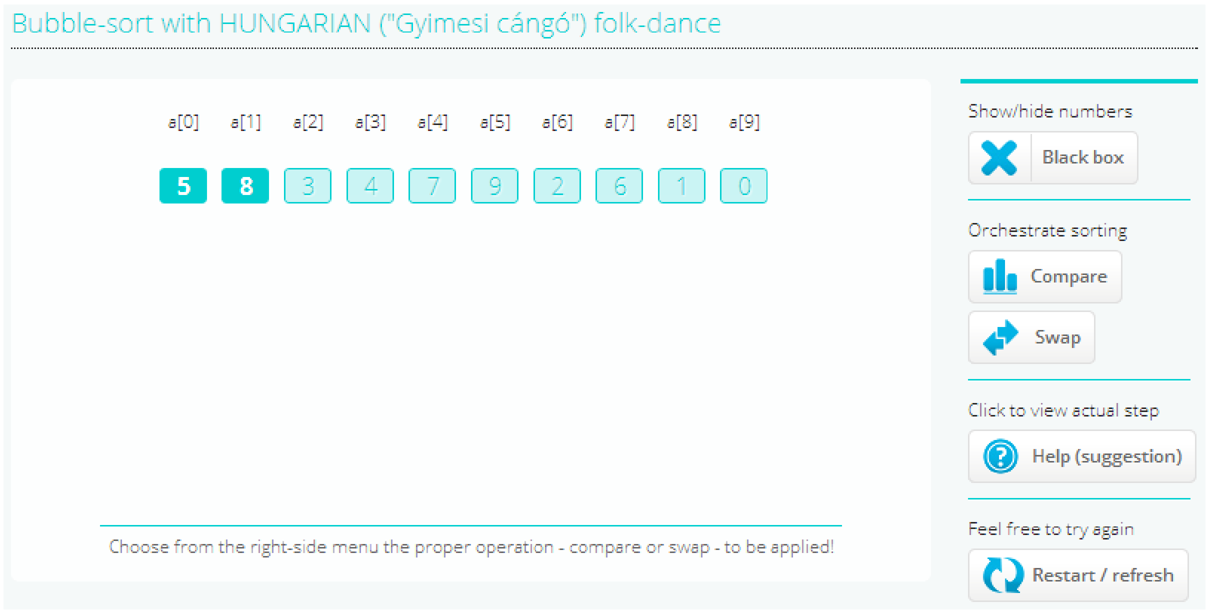


Figure 9. *Sorting in a simulation program (own program) [11]*

The above figure demonstrates a program I made to visualize a possible classroom task. The simple algorithms can be tried with a 6-item list. To display or change the items, we can use mobile *arrows* marked as i and j, traditional loop counters in algorithms and programs. Next to these, the *min* marker can be applied with selection sort, while the *double arrow*, for the jth and j+1st item, goes with bubble sort. The *eye* displays all the items so we can check the solution easily, whereas a click on the *cube* provides us with a new random list.



Choose from the right-side menu the proper operation - compare or swap - to be applied!

Figure 10. *Sorting in the simulation program of the* Algo-rythmics*'s webpage [12]*

We will find a more serious application on the *Algo-rythmics* webpage [12], renowned for its videos presenting sorting algorithms through folk dancing. Insertion sort, selection sort, bubble sort, quicksort, and Shell sort are demonstrated and offered for trial on the webpage (figure 10). In interactive mode, we can select two elements and decide if we want to compare or exchange them. The program warns us when we are about to make a wrong step according to the rules of the given algorithm. With one single click, the item values can be hidden so that comparison can become necessary. If we get stuck, the Help button assists us to get back on track.

3.4. Executing educational goals

The computer-based implementation of sorting algorithms may require the application of some kind of a data structure, the organization of nested loops, and the production of a recursive procedure. Sorting algorithms are among the most complex programming tasks; their description and coding are mentally demanding challenges, for which we can offer assistance tailored to the abilities of the students.

4. Conclusions

We can choose from a plethora of methods how to teach sorting algorithms. We need to consider not only the specific needs of our students, but also the peculiarities of the different methods: each one fits a given phase of the learning process best. In early primary school, objects of different weight facilitate students to understand, practice, and examine the efficiency of algorithms. In secondary school, however, other methods work better for the comprehension, description, and coding of the algorithm, let alone for its efficiency check.

To understand the essence of algorithms, pictures seem to be the most efficient tools, because they visualize only the most important parts. For the comprehension of the details, however, animations, which display all the steps, appear more adequate. For checking students' comprehension, before they start describing or coding an algorithm, simulation programs are the best; in addition, they can be effective tools for practicing the specific algorithms. Algorithm description and coding are best promoted by animations that display loop counters and markers typically used in the algorithm. Lastly, for efficiency check we can use animations and programs that were designed for this specific purpose.

Although my article focused on visual and motor methods used specifically for teaching sorting algorithms, it would be useful to consider and aim for such methods in the entire programming education [13]. As the Chinese proverb reminds us: ”I hear and I forget. I see and I remember. I do and I understand.”

References

[1] Szlávi, P.; Zsakó, L. (2008): *Módszeres programozás: Programozási tételek (Methodic programming: Programming theses), Mikrológia 19*, ELTE IK, Budapest

[2] Rónyai, L.; Ivanyos, G.; Szabó, R. (2005): *Algoritmusok (Algorithms)*, Typotex, Budapest

[3] Falus, I. (2006): *Didaktika – Elméleti alapok a tanítás tanulásához (Didactics – Theoretical basics to teaching learning)*, Nemzeti Tankönyvkiadó, Budapest

[4] Szlávi, P.; Zsakó, L. (2012): ICT competences – Algorithmic thinking, *Acta Didactica Napocensia*, Vol. 5. No. 2., pp. 10

[5] Bell, T.; Witten, I. H.; Fellows, M.: Computer Science Unplugged, An enrichment and extension programme for primary-aged children *http://csunplugged.org/sites/default/files/activity_pdfs_full/unpluggedTeachersMar2010-USletter.pdf* [05/20/2014]

[6] Sorting Algorithms Section 3, C.Eng 213 Data Structures *http://cow.ceng.metu.edu.tr/Courses/download_courseFile.php?id=5451* [05/20/2014]

[7] Kátai, Z. (2007): *Algoritmusok felülnézetből (Overview to algorithms)*, Cluj-Napoca: Scientia, ISBN 978-973-7953-74-2

[8] Visualization of Data Structures and Algorithms *http://rosemarietan.com/fyp* [05/20/2014]

[9] Sorting Algorithm Animations *http://www.sorting-algorithms.com* [05/20/2014]

[10] Kátai, Z. (2006): *Módszerek és eszközök az informatikaoktatás hatékonyságának növelésére (Methods and tools for developing the efficiency of teaching informatics)*, Doktori (PhD) értekezés, Debreceni Egyetem TTK

[11] Bernát, P.: Cserés rendezések (Exchange sorting) *http://scratch.mit.edu/projects/87514* [05/20/2014]

[12] Algo-rythmics *http://algo-rythmics.ms.sapientia.ro* [05/20/2014]

[13] Szlávi, P.; Zsakó, L. (2003): Methods of teaching programming. *Teaching Mathematics and Computer Science 1*, No. 2, pp. 247-258

Authors

Péter Bernát, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, e-mail: bernatp@inf.elte.hu