ACTA DIDACTICA NAPOCENSIA
Volume 9, Number 2, 2016

JAVASCRIPT LIBRARY FOR DEVELOPING INTERACTIVE MICRO-LEVEL ANIMATIONS FOR TEACHING AND LEARNING ALGORITHMS ON ONE-DIMENSIONAL ARRAYS

Ladislav Végh

Abstract: The first data structure that first-year undergraduate students learn during the programming and algorithms courses is the one-dimensional array. For novice programmers, it might be hard to understand different algorithms on arrays (e.g. searching, mirroring, sorting algorithms), because the algorithms dynamically change the values of elements. In these situations, visualizations and animations might be helpful didactic tools. In this paper, we briefly overview animations with different views and introduce our Javascript library for developing interactive micro-level animations. Using this library, different sorting algorithm animations were created, which were used in a pedagogical experiment. The results showed that our interactive animations helped students to understand the visualized sorting algorithms.

Key words: developing algorithm animations, interactive animations, micro-level animations, teaching programming, sorting algorithms

1. Introduction

Learning programming, solving algorithm problems, and gaining algorithmization thinking skills is one of the hardest tasks for first-year computer science students. The main reason why algorithms are hard to understand is that they work with abstract data structures and change the values of the elements dynamically. Animations and visualizations might be helpful in learning process because they represent abstract data by concrete objects (e.g. playing cards, wooden boxes, rectangles, balls, lines), and they dynamically change the position (or other properties) of these objects.

Even though, that using animations are motivating for students and help them focus their attention on the visualized processes [1, 2], they are not educationally effective in every case [3-8].

After ambiguous results of numerous pedagogical experiments, researchers started to investigate features of didactically effective animations and visualizations. There are two broad categories of attributes which should be taken into consideration for developing animations for teaching and learning: graphical design and interactivity. Mayer [9] defined twelve principles of designing multimedia learning materials, which describe how texts, voices, and images are recommended to use for reducing extraneous processing, managing essential processing, and fostering generative processing in multimedia learning materials. In the field of interactivity of algorithm animations were also made some recommendations which deal with the control modes of the animations, possibilities of modifying input data, questioning students during the animations, and possibilities of adapting the animations to the level of students' knowledge [2, 3, 10-14].

Hansen et al. [3] recommended to use three types of algorithm animations successively:

1. The first type of animations, with conceptual view, introduces the main ideas of the algorithms, but it does not go into the details. These animations establish the understanding of abstract concepts using real-world situations, e.g. sorting algorithms by using wooden boxes or playing cards. Thus, students are encouraged to develop connections between real-life events and the logic of the algorithms in the beginning of their learning process [10, 15].

Received May 2016.

Furthermore, this type of animations also serves as motivation [3]. Examples of animations with conceptual views are on the following web pages: http://bit.ly/1O1NoVh (sorting playing cards), http://bit.ly/23tUUtS (using Lego bricks to demonstrate bubblesort algorithm), http://bit.ly/1Tr8to6 (sorting wooden boxes), http://bit.ly/1q32o9V (solving eight queen problem by backtracking), http://bit.ly/1WMXZF9 (using balls and robots to illustrate the main steps of the mergesort algorithm).

2. The second, micro-level animations illustrate the algorithms in details, on small data sets – usually on 6-8 elements. It is very common that this type of animations contains pseudocodes or source codes, in which the actual steps of the algorithms are highlighted [3, 10, 11, 13]. If the algorithms are taught during a programming course, where a given programming language is used, it is recommended to display the source codes of the algorithms. Otherwise, it is better to use the pseudocodes, because it contains a higher level of abstraction. Thus, students will not get lost in details, and they will learn algorithms independently of any programming language [11]. An example of animation with micro-level view can be found on http://bit.ly/1O1R897 (simple exchange sort algorithm). In the next sections of this paper, we will focus on this type of animations. We will present our Javascript library which can be used to create micro-level animations. Some interactive animations created with our library are available on the following web pages: http://bit.ly/1TvlqO0 (swapping two variables, summing values of elements, mirroring an array, finding the minimum/maximum in an array, or finding the index of the minimum/maximum in an array), http://bit.ly/26Z7ypl (simple exchange sort, bubblesort, improved bubblesort, insertion sort, improved insertion sort, minsort, and maxsort), http://bit.ly/24B9WjX (quicksort and mergesort algorithms).
3. The third, macro-level animations show the algorithms globally, on large data sets, usually on 20-50 elements. The details of the algorithms are hidden. Using these animations students can understand and compare the effectiveness of different algorithms. Macro-level animations can also enforce students' mental models. An example of macro-level animations can be found on http://bit.ly/1rZrubL (comparison of the running time of different sorting algorithms using different types of input data sets).

As we mentioned before, in this article we will deal with the second type of algorithm animations. Our goal was to develop micro-level animations for teaching and learning algorithms on one-dimensional arrays. Undergraduate computer science students learn these algorithms during the first semester of programming and algorithms courses. Because it is recommended to use animations with the same kind of graphical representation and control buttons during the whole course [11], we decided to create a Javascript library first (we called it "inalan" as an acronym from "interactive algorithm animations"). Next, we developed interactive animations using this library. Thus, creating animations was easier, and students had similar control buttons and graphical representation of the arrays in all animations. Finally, we conducted a pedagogical experiment where our goal was to determine if our interactive micro-level animations can help students to understand sorting algorithms in details.

2. Javascript library (inalan)

The inalan Javascript library (http://inalan.ide.sk) contains classes for developing interactive animations of algorithms on one-dimensional arrays. The library uses the canvas element of the HTML5 to display the animations. This technique enables students to watch the animations in web browsers without the necessity of installing any plugin. The different parts of an animation developed with the inalan library are shown in Figure 1.

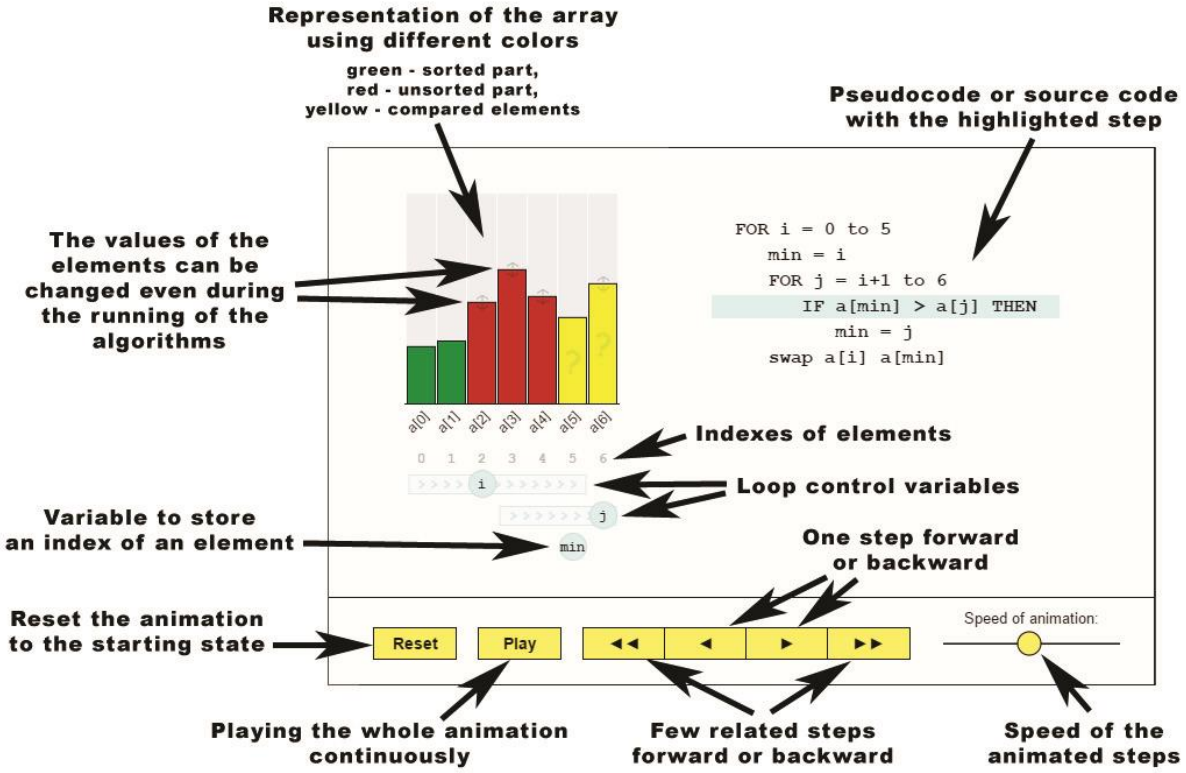

Figure 1. *Different parts of a micro-level animation developed with inalan library*

There are three recommended modes to use this kind of animations:

- Students can play the whole animation continuously. The animation can be anytime stopped and continued. Learners can reset the animation to the starting state and play the animation again from the beginning. In this mode, it is suggested to concentrate only on the changes in the representation of the array (green, red, or yellow columns). It is not recommended to follow the pseudocode or source code because learners usually do not have enough time to think about the algorithm in such details.
- In the second mode, students can use the double arrow (>>) button to play some related steps of the algorithm. After animating these related steps, the animation stops automatically, and students can think over the visualized steps. Next, they can continue with the following steps (>>), or they can step backward (<<) and watch again the lastly animated steps.
- In the last, the most detailed mode, students can observe the visualization of the algorithm step by step, using the single arrow (>) button. They can also step backward (<) whenever they need. This is the best mode to follow and understand the pseudocode or source code of the algorithm, as well.

Learners can change the values of elements during the animations by dragging the top of the red or yellow columns. This feature can be used for experimenting with different data sets in different parts of the algorithms. Students can run algorithm animations with their initial data; they can anytime stop the animations, step backward, change the values of elements and observe the behavior of the algorithms again with new data.

For developers of learning materials, the inalan Javascript library offers the following classes:

- *class Stage* – Every animation contains one object created from this class. This object connects the Javascript code with the canvas element of the HTML5 document, controls the animation, and draws different parts of the animation on the canvas.

- *class VisuVariable* – An object of this class represents simple variable by an interactively changeable red/green/yellow column.
- *class VisuArray* – Array of VisuVariable objects. An object of this class also contains index and loop control variables, which can be displayed on the canvas.
- *class VisuCode* – Pseudocode or source code of the visualized algorithm with the possibility of highlighting the actual steps of the algorithm.
- *class VisuLabel* – Simple textual comment. An object of this class can be used to display any explanation or comment on the canvas of the animation.
- *class VisuButton* – Simple button. Objects of this class can be found in the bottom control panel of the animation. Using this class, developers of interactive animations can add their own buttons anywhere into the animation.
- *class VisuScrollbar* – Simple scrollbar. An object of this class can be found in the bottom control panel for changing the speed of the animation.

The detailed documentation of classes with examples, their attributes and methods are available on the wiki of the inalan library: https://github.com/veghl/inalan/wiki.

To create an interactive animation using the inalan library, first of all we need to create a canvas element in HTML5 document, for example:

```
<canvas id="myCanvas" width="700" height="440"></canvas>
```

Next, we need to connect this canvas element to the Stage object in Javascript:

```
var stage = new inalan.Stage("myCanvas");
```

To create a VisuArray object and add it to the stage we can use the following Javascript code:

```
var a = new inalan.VisuArray("a", [0, 0, 0, 0, 0, 0, 0, 0], true);
a.randomize(30, 150);
a.x = 80;
a.y = 220;
stage.add(a, "a");
```

Similarly we can add VisuVariable, other VisuArray, VisuCode, VisuLabel, VisuButton, or VisuScrollbar objects to the stage. Next, we need to define the steps of the animation and loop conditions. Every step of the animation or condition of a loop is a function in Javascript, for example:

```
var setLoopControlVariable = function () {
      stage.get("a").setIndex("i", stage.vars.i);
      stage.get("a").setLoopMarker("i", 0, 3);
      return 200;
}

var swapElements = function () {
      stage.get("a").setIndex("7-i", 7 - stage.vars.i);
      stage.swap( stage.get("a")[stage.vars.i],
                  stage.get("a")[7-stage.vars.i] );
}

var incrementLoopControlVariable = function () {
      stage.get("a").deleteIndex("7-i");
      stage.get("a")[stage.vars.i].setGreenColor();
      stage.get("a")[7-stage.vars.i].setGreenColor();
      stage.vars.i++;
```

```
        return 0;
}

var checkLoopControlVariable = function () {
        return stage.vars.i <= 3;
}

var finalStep = function () {
        stage.get("a").deleteIndex("i");
        stage.get("a").deleteIndex("7-i");
}
```

The setLoopControlVariable, swapElements, incrementLoopControlVariable, and finalStep functions define the steps of the algorithm animation. The return value of these functions are used to control the animation depending on the animation mode:

- If the return value is -1, or the function does not have any return value ("undefined" value): the animation will stop (in case the animated step was started using the >> or > button), or will wait few milliseconds defined by the speed scrollbar on the right side of the control panel (in case the animated step was started with the Play button).
- If the return value is 0: the animation will continue automatically with the next step, without any stop or delay.
- If the return value is a positive integer: the animation will stop (in case the animated step was started using the > button), or will wait few milliseconds defined by the return value (in case the animated step was started with the >> or Play button).

The checkLoopControlVariable function defines the condition of a repeat loop. Its return can be a boolean value. If this function returns a true value, some of the previous steps will be repeated, in another case the animation will continue with the next steps.

Finally, we need to define the order of the steps, e.g.

```
stage.setSteps([
        [
          setLoopControlVariable,
          swapElements,
          incrementLoopControlVariable
        ], checkLoopControlVariable,
        finalStep
]);
```

In this example, the setLoopControlvariable, swapElements, and incrementLoopControlVariable functions are used to animate the algorithm. This sequence of steps can be iterated depending on the return value of the checkLoopControlVariable function. Finally, the finalStep function is called one time to remove the index variable from the canvas of the animation when the algorithm ends.

Some interactive animations developed with the inalan Javascript library can be found on the web pages: http://anim.ide.sk/basic_algorithms.php, http://anim.ide.sk/sorting_algorithms_1.php, http://anim.ide.sk/sorting_algorithms_2.php.

3. Pedagogical experiment

Using our interactive animations we conducted a pedagogical experiment during the academic year 2015/16. The goal was to determine if our interactive animations help students to comprehend the sorting algorithms easier than comprehend them with static pictures representing the main steps of the algorithms. In the experiment were involved 71 first-year computer science students from J. Selye University, Komárno, Slovakia. They were divided randomly into four study groups during the

„Algorithmization and programming” course. Before the experiment, all students filled out a pre-test, in which we tried to assess students’ previous knowledge. Next, to learn sorting algorithms, two of the study groups (35 students) used interactive animations, and the other two study groups (36 students) used static pictures containing the main steps of the algorithms (see Figure 2). The experiment took five weeks (5 × 1.5 hours) based on the following schedule:

- 1st week: pre-test, simple exchange sort, bubblesort,
- 2nd week: improved bubblesort, insertion sort, improved insertion sort,
- 3rd week: selection sort – minsort, maxsort,
- 4th week: quicksort,
- 5th week: mergesort.

Figure 2. *Screenshot of the interactive animation of bubblesort algorithm, and a static picture containing the main steps of the bubblesort algorithm*

The interactive animations and graphical representations were completed with the teacher’s explanation in voice. After the explanations, students had time to experiment with the animations, or think over the steps of the algorithms represented by static images. They could ask questions for clarification of the steps of the algorithms. Explaining the algorithms and answering the questions usually took longer in the study groups where static pictures were used. When students did not have any more questions, they were asked to fill out a test related to the given sorting algorithm.

4. Results

After the experiment, we examined the acquired results. The mean numbers, standard deviation, and medians of the results in percentage are shown in Table 1.

Table 1. *Mean numbers, standard deviations, and medians of the results of the test in percentage*

	Results of students who used INTERACTIVE ANIMATIONS				**Results of students who used STATIC PICTURES**			
	N	Mean (%)	Std. Dev. (%)	Median (%)	N	Mean (%)	Std. Dev. (%)	Median (%)
PRETEST	34	59,56	23,501	58	33	53,24	24,276	50
SIMPLESORT	34	73,32	15,338	75	33	62,73	17,424	63
BUBBLESORT	34	88,09	14,154	88	33	80,55	16,216	88
BUBBLESORT2	31	77,97	16,353	75	34	52,12	25,156	50
INSERTSORT	31	87,26	12,223	88	34	81,12	13,906	88
INSERTSORT2	31	67,65	19,818	71	34	48,35	26,722	43
MINSORT	32	83,81	15,126	89	27	70,89	21,684	78
MAXSORT	32	84,94	16,679	88	27	78,89	18,352	88
QUICKSORT	30	83,77	15,110	82	23	71,35	15,683	73
MERGESORT	28	70,25	20,323	73	19	51,74	22,896	64

Shapio-Wilk tests showed that there is not normal distribution in the data, except the tests that measured the understanding of the mergesort algorithm. For this reason, we used independent-samples t-test to evaluate the results of the tests related to the mergesort algorithm, and Mann-Whitney U tests to evaluate the results of other tests.

The Mann-Whitney U test showed that the distribution of the scores in pre-test for both groups were similar, as assessed by visual inspection. Median score for students in the first group (58) and students in the second group (50) was not statistically significantly different, $U = 468$, $z = -1.173$, $p = 0.241$. The result of this test shows that students in both groups had similar previous knowledge.

Table 2 shows the results of Mann-Whitney U tests for different sorting algorithms.

Table 2. *Results of Mann-Whitney U tests for different sorting algorithms*

Sorting algorithm	**Is the distributon of points similar in groups?**	**Students using INTERACTÍVE ANIMÁTIONS**	**Students using STATIC PICTURES**	**Result**
Simplesort	no	mean rank = 40.19	mean rank = 27.62	$U = 350$, $z = -2.715$, $p = 0.007$
Bubblesort	no	mean rank = 38.76	mean rank = 29.09	$U = 399$, $z = -2.115$, $p = 0.034$
Bubblesort2	no	mean rank = 43.29	mean rank = 23.62	$U = 208$, $z = -4.245$, $p < 0.0005$
Insertsort	no	mean rank = 37.03	mean rank = 29.32	$U = 402$, $z = -1.713$, $p = 0.087$
Insertsort2	no	mean rank = 40.05	mean rank = 26.57	$U = 308$, $z = -2.913$, $p = 0.004$
Minsort	no	mean rank = 34.75	mean rank = 24.37	$U = 280$, $z = -2.365$, $p = 0.018$
Maxsort	no	mean rank = 32.91	mean rank = 26.56	$U = 339$, $z = -1.467$, $p = 0.142$
Quicksort	no	mean rank = 32.37	mean rank = 20.00	$U = 184$, $z = -2.964$, $p = 0.003$

Distributions of the scores for students using interactive animations and students using static pictures were not similar, as assessed by visual inspection. In most cases, except the insertion sort and maxsort algorithms, the scores for students using interactive animations were statistically significantly higher than for students using static pictures. For insertion sort and maxsort algorithms, there were not statistically significant difference between the groups.

To evaluate the results of the tests related to the mergesort algorithm we used independent-samples t-test (see Table 3).

Table 3. *Results of independent-samples t-test for mergesort algorithm*

Group Statistics					
	GROUP	N	Mean	Std. Deviation	Std. Error Mean
MERGESORT	animation	28	70,25	20,323	3,841
	graphics	19	51,74	22,896	5,253

Independent Samples Test									
MERGESORT	Levene's Test for Equality of Variances		t-test for Equality of Means						
								95% Confidence Interval of the Difference	
	F	Sig.	t	df	Sig. (2-tailed)	Mean Diff.	Std. Error Diff.	Lower	Upper
Equal variances assumed	0,384	0,539	2,912	45	0,006	18,513	6,357	5,709	31,318
Equal variances not assumed			2,845	35,607	0,007	18,513	6,507	5,311	31,715

There were no outliners in the data, as assessed by inspection of a boxplot. Scores for both groups were normally distributed, as assessed by Shapiro-Wilk's test ($p>0.05$), and there was homogeneity of variance, as assessed by Levene's test for equality of variances ($p=0.539$). Students who used interactive animations to learn mergesort algorithm reached higher scores in test (mean: 70.25 ± 20.32) than students who used static graphics (mean: 51.74 ± 22.90), statistically significant difference 18.51 (95% CI, 5.71 to 31.32), $t(45) = 2.912$, $p=0.006$.

5. Conclusion

In conclusion, students who learned sorting algorithms using our interactive micro-level animations got better results on tests than students who used static images. The results were statistically significant for seven sorting algorithms, for two algorithms there was no statistically significant difference between the groups.

Our experiences showed that our interactive micro-level animations can be used as helpful didactic tools for teaching and learning algorithms on one-dimensional arrays. Students can change the values of elements even run-time, so they can easily observe the behavior of different parts of the algorithms on different data sets. The ability to move forward and backward anytime in the algorithms, using different animating modes, gives even more possibilities to experience with the interactive animations.

References

[1] Young, J.R., *Homework? What Homework? Students Seem to Be Spending Less Time Studying Than They Used To.* The Chronicle of Higher Education, 2002. **49**(15), A35-A37.

[2] Grissom, S., M.F. McNally, & Naps, T. (2003). *Algorithm visualization in CS education: comparing levels of student engagement*. in *Proceedings of the 2003 ACM symposium on Software visualization*. San Diego, California: ACM.

[3] Hansen, S., Narayanan, N.H. & Hegarty, M. (2002). *Designing educationally effective algorithm visualizations.* Journal of Visual Languages and Computing, **13**(3), 291-317.

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

[5] 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.

[6] 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): p. 265-284.

[7] Byrne, M.D., Catrambone, R. & Stasko, J.T. (1999). *Evaluating animations as student aids in learning computer algorithms.* Computers & Education, **33**(4), 253-278.

[8] Kann, C., Lindeman, R.W. & Heller, R. (1997). *Integrating algorithm animation into a learning environment.* Computers & Education, **28**(4), 223-228.

[9] Mayer, R.E. (2009). *Multimedia Learning* (second ed.). New York, USA: Cambridge University Press. 304.

[10] Bernát, P. (2014). *The Methods and Goals of Teaching Sorting Algorithms in Public Education.* Acta Didactica Napocensia, **7**(2), 1-10.

[11] Fleischer, R. & Kucera, L. (2002). *Algorithm animation for teaching.* Software Visualization, **2269**, 113-128.

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

[13] Naps, T. L., et al. (2002). *Exploring the role of visualization and engagement in computer science education.* SIGCSE Bull., **35**(2), 131-152.

[14] Végh, L. (2011). *Animations in Teaching Algorithms and Programming (Animácie vo vyučovaní algoritmov a programovania).* in *Nové technologie ve vzdělávání*. 2011. Olomouc, CZ: Palacký University, Olomouc.

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

Author

Ladislav Végh, J. Selye University, Komárno, Slovakia, e-mail: vegh34@gmail.com