ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/224954355

ViSA: Visualization of Sorting Algorithms

Conference Paper · May 2012

CITATIONS
4

READS
2,472

2 authors, including:

Tihomir Orehovački
Juraj Dobrila University of Pula
126 PUBLICATIONS **671** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

eLogoped ("e-Speech-Language Therapist") View project

All content following this page was uploaded by Tihomir Orehovački on 08 October 2014.

The user has requested enhancement of the downloaded file.

ViSA: Visualization of Sorting Algorithms

I. Reif and T. Orehovacki
University of Zagreb, Faculty of Organization and Informatics, Varazdin, Croatia
{ivan.reif, tihomir.orehovacki}@foi.hr

Abstract - Algorithm analysis and design is a great challenge for both computer and information science students. Fear of programming, lack of interest and the abstract nature of programming concepts are main causes of the high dropout and failure rates in introductory programming courses. With an aim to motivate and help students, a number of researchers have proposed various tools. Although it has been reported that some of these tools have a positive impact on acquiring programming skills, the problem still remains essentially unresolved. This paper describes ViSA, a tool for visualization of sorting algorithms. ViSA is an easy-to-set-up and fully automatic visualization system with step-by-step explanations and comparison of sorting algorithms. Design principles and technical structure of the visualization system as well as its practical implications and educational benefits are presented and discussed.

I. INTRODUCTION

Programming courses are a fundamental part of both computer and information science curricula. Their purpose is to introduce basic programming concepts (e.g. control structures, aggregation mechanisms, sorting algorithms, etc.) to students, thus helping them to acquire programming skills. This goal is often very difficult to achieve and according to some studies (e.g. [28]), introductory programming courses have relatively high dropout and failure rates. Nevertheless, a tremendous variation in pass, fail, abort and skip rates were reported by Bennedsen and Caspersen [2]. It should be noted that the higher average passing rate was established at colleges (88%) than at universities (66%). The most common reason for failing a course was the abstract nature of programming concepts since this was the first course where students were confronted with programming. According to Radošević et al. [1], the main problems in learning programming are: lack of previous knowledge, problem-solving algorithms, fear of programming, and the perception of difficulty of programming language syntax. They also discovered that problem-solving algorithms are perceived as a much greater obstacle than programming language syntax itself. Regarding the problems related to the C++, programming language syntax, research on compilation behavior [20] has reported that most common mistakes made by novice programmers are: use of undeclared variable, defining variable within iteration, wrong operator in the logical expression, etc.

The objective of novice programmers is to implement the abstract process of algorithm execution in a way or language that will be understandable to the computer. However, if a programmer wants to 'explain' something to the computer, he or she first has to understand it perfectly. Namely, it is very difficult to understand and learn complex algorithms such as iterations, recursions or sorting algorithms only by watching code lines or a flow chart. On the contrary, if students can control their own data sets and see the whole process of algorithm execution, they are able to draw conclusions from the resulting data or in course of the algorithm visualization.

The aforementioned claims have motivated researchers to develop tools that will simplify and facilitate the learning of programming concepts, thus helping students to acquire both programming knowledge and skills. With an aim to engage students in learning activities, the implementation of a visualization technology within educational process has been proposed [8][27]. The earliest examples were slideshow presentations or simple web interfaces with basic structures and operations [3], [5]. Later, more advanced tools such as DrScheme [29] or Robolab [30] were introduced. However, a mere development and deployment of visualization tools is not enough to attain an objective of learning complex algorithms. Hansen et al. [6] suggested that visualization technology should involve students in activities such as the construction of their own input data sets and interaction with animation, which will eventually result in their own predictions and logical conclusions.

With an aim to facilitate the understanding, analysis and design of sorting algorithms, a tool for their visualization called ViSA is presented in this paper. Using ViSA, students are guided in the learning process by a detailed step-by-step explanation. The remainder of the paper is structured as follows. Section 2 provides a brief literature review. In section 3, specific features and examples of ViSA use are described. Section 4 presents the results of a pilot study on perceived usefulness, perceived ease of use, behavioral intention, and attitude towards use of ViSA. Concluding remarks and future work directions are provided in the last section.

II. RELATED WORK

A gap between the way in which algorithms are taught and the students' learning style is one of the main reasons why introductory programming courses are perceived as hard. Namely, while the majority of people are visual types [7], teaching, to a great extent, is verbal. Visual learning is a type of learning where people prefer using images, pictures, colors, and maps to organize information in order to comprehend complex concepts. This is the main reason why algorithm visualization (AV) tools are implemented in learning process. In literature there are a lot of AV tools such as Alice [9], Software Visualization

System [7], Jcat [10], Guido von Robot [11], Samba Algorithm Animation System [12], JHAVÉ [4], etc. Most of them contain a set of control options for the user and can run in a step-by-step mode. On the other hand, only some of them allow the user to adjust the speed of animation (e.g. Samba [12]). Moreover, the feature allowing the user to going backwards a certain number of steps as in DDD [13] is relatively uncommon.

There are two types of AV: animation and simulation tools. Algorithma 99 [14] is an animation tool that enables the user to enter algorithms in a predefined pseudo-code language which can then be visualized. Such a method of animation requires the knowledge of setting up the pseudo code and linking it with control elements which then execute the animation. Systems like JAWAA [15] and JSamba [4] simulate the algorithm provided in a special syntax that needs to be learned prior to starting using the tools.

There are several animation tools based on interpreting source code, such as Zstep95 [16]. However, without precognition related to system expressions and their contribution to the animation, user cannot make the decision whether to use the details (animation controls) or an expression. On the other hand, if all elements are enabled and shown during animation, some key parts needed for the understanding of algorithm could be overlooked.

The KAMI system [17] is an animation tool that relies on debugging data and generates a slide-by-slide display illustrating the code execution. Slide by slide animations are not controlled by the user or reviewed by the instructor so they might present excessive additional information. Jeliot [18] reads Java source code and automatically generates visualizations. This tool does not offer the explanation of any step executed during animation and is limited only to java source code.

BlueJ [19] is one of the first systems developed to teach introductory object oriented programming. The key feature of the system is the static visualization of a class structure as a UML diagram. Furthermore, it allows the learner to interact with the objects by creating them, calling their methods, and inspecting their state with easy-to-use menus and dialogs. However, it does not provide any dynamic visualization of the program.

Simulation tools are more user-friendly and useful in teaching programming. One of them is Alice [9], a general purpose simulation tool aimed for teaching any kind of programming concepts. However, before it can be used for teaching specific algorithms, the user has to learn its specific syntax. In addition, it does not allow any interaction with animation during programming. According to Hansen et al. [6] the use of Alice produces a great deal of time wasted on learning a new syntax instead of learning the algorithms at hand. For example, to create a simple iteration, a user would have to know how to use the syntax, set up 3D models, and implement the triggers that interact with the models. Setting up a tool like Alice would also require intense preparations on the host computer such as 3D driver installation, etc.

A similar tool aimed for teaching programming and algorithms is Guido von Robot. It is not 3D based, but instead uses its own programming syntax which is entirely written in Python. When working with Guido von Robot [11], similar problems occur. Although the knowledge of triggers and animation is not required here, a more complex syntax is still used. Guido von Robot is a simulation tool made of two basic parts: pointer, and array of dots. The pointer represents the current position of data in the algorithm while the array of dots represents the path which the data take. This is why the user has to integrate the pointer move commands inside the code itself. The user also needs to know what the code does in order to integrate the data for moving the cursor correctly, which makes the process of learning a new algorithm much more difficult than necessary.

While the aforementioned tools are aimed for learning programming in general, the tools described below are intended for sorting algorithms visualization. Jcat [10] is a web based simulation tool developed in the Java programming language. Unlike the afore-mentioned tools, it enables sorting algorithm visualization and provides information related to the steps taken during simulation. However, it has a number of drawbacks. Users are not allowed to customize the array size or elements value, which constrains interaction. Furthermore, users cannot control the speed or flow of animation and the information that is shown during the animation does not explain why certain elements are swapped or compared. Executing web based applications also implies several potential problems such as missing add-ons that are required for code execution.

Animal [21] animations include a linked sequence of animation steps operated by the pause, stop, play and rewind buttons just like any other visualization tool. The main window is sectionalized in three segments. A control tool bar at the top of the windows allows adjusting the display speed and magnification. A tool bar at the bottom contains control elements for triggering the animation display and the last part is the pseudo-code area. Animal provides several different control mechanisms for selecting the current animation step: a control tool bar, a list of labeled animation steps, a slide ruler, and text field for direct input. The main advantage of this tool is its support to rewinding an animation or skipping to a certain step. However, it has serious problems with screen resolution. Namely, data containers, graphs, and pseudo-code will change their positions or completely cover the animation control if the screen resolution is changed. Additionally, this tool does not provide any information about the steps that are taken during the animation.

Software Visualization System [7] is very similar to ViSA. It allows users to select one of the sorting algorithms and, subsequently, to fill in the array data and run the animation. The main window is made of four panels. The first panel presents the execution of the algorithm in an array. Along with the execution panel, the interface also contains a pseudo-code panel, where users can learn the algorithm during code creation. Beneath the pseudo-code panel the execution status messages panel is located which shows the steps during the animation. Next to it is a panel which shows variables in use. Although this

kind of window organization does not explain steps in which specific elements are swapped or compared, it shows the user full algorithm information. Nevertheless, it does not support more than seven elements in the array. We believe that customization of the numbers of elements in an array is very important because some algorithms work better with larger arrays than small ones. Given that Software Visualization System does not support changing the size of an array, it is not really possible to compare two different sorting algorithms. Finally, the lack of multi-threading, which means that only one algorithm can be executed at the same time, presents the greatest disadvantage of this tool.

III. VISUALIZATION OF SORTING ALGORITHMS (VISA)

The purpose of ViSA is to provide an easy to use learning interface with as much as possible feedback information presented to the user through explanation. As mentioned earlier, a good AV tool has to allow a quick integration, where one-click setup is enough to have the software ready for use and enable users to construct their own input data sets and interact with the visualization. ViSA installation is powered by the Microsoft click-one install method [22], where the software automatically downloads all the necessary drivers and after the installation it runs entirely by itself.

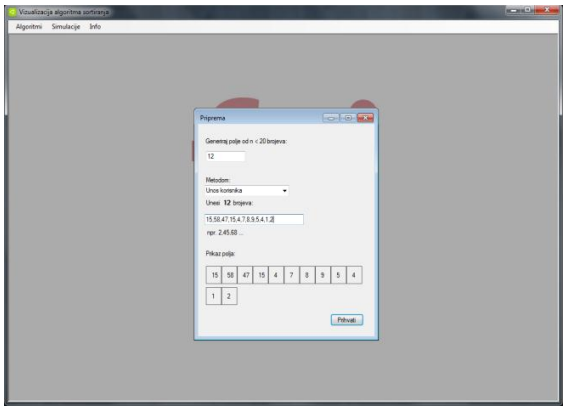

Figure 1. ViSA data set construction

ViSA supports well known sorting algorithms such as Cocktail Sort, Comb Sort, Heap Sort, Insertion Sort, Merge Sort, Quick Sort, Selection Sort, and Shell Sort. After choosing an algorithm, the user is prompted with the data set construction form. The form elements (textboxes, labels, array preview) are dynamic, thus allowing the user to make changes at any time without preventing the software to receive the correct data input. In case of an error committed by the user, he or she will receive a warning about the wrongly entered or inaccurately formatted data. The user can select among a range of methods for generating random numbers such as pure random, big values at the end, small values at the end, or almost sorted. If the user wants to enter specific values, he or she will simply enter the numbers in the empty textbox delimited with a coma and an array preview will be generated. When the user has finished with setting up the data set, ViSA runs a final data format test and allows the user to proceed. This form of input generation is constructed in a way to be forwarded to the visualization, allowing the students to learn whether their data set drives the animation in the anticipated manner. The animation sequence is generated dynamically, as presented in the code block 1. After each comparison or swap, the information about the elements is added to the Listbox queue.

```
    Do

        blnSwapped = False

        For i = iMin To iMax

        ListBox1.Items.Add((i.ToString + "|" + (i
+ 1).ToString).ToString())

              If Array(i) > Array(i + 1) Then

        ListBox1.Items.Add((i.ToString + ">" + (i
+ 1).ToString).ToString())

                           varSwap = Array(i)

                           Array(i) = Array(i + 1)

                           Array(i + 1) = varSwap

                           blnSwapped = True

                      End If

                Next

     ListBox1.Items.Add(((iMax  +  1).ToString  +
"+" + (iMax).ToString).ToString())

                iMax = iMax - 1

    Loop Until Not blnSwapped
```

Code block 1. Bubble Sort

After the algorithm is finished, the actions for the visualization are executed from the Listbox one by one. This is the reason why it is very easy to add more algorithm examples to the ViSA repository. If an algorithm is added, any of the control elements can be activated by inserting the appropriate syntax, as shown in code block 2.

```
ListBox1.Items.Add((i.ToString + ">" + (i +
1).ToString).ToString())
```

Code block 2. Control elements syntax

The function Listbox1.Items.add receives three parameters: (i) *i* that represents the first element, (ii) string ">" that represents a comparison action, and (iii) *i+1*, a variable that contains the second element.

Figure 2. ViSA main window

The ViSA main animation (presented in Figure 2) window is composed of several standard controls: (i) oval shapes that represent elements in an array, (ii) slider for controlling the animation speed, (iii) start and stop button, (iv) information about complexity of the algorithm including the number of swaps and comparisons that are executed during the animation, and (v) the panel on the right side that contains explanations of each step that is taken during the visualization. The animation flow can be controlled by the user with the next button that will either execute one step at the time or fully automatically. Additional controls are added to each sorting algorithm for better understanding of the algorithm (e.g. a data structure-max tree for Heap sort or a divided sub-array in Merge Sort). After each action (swap, comparison, interval readjustment or pivot update) an appropriate explanation is shown in the bottom right area. The animation playback has two modes: automatic and step-by-step mode. In any of these two modes the user can pause the animation and read the description of the current action. The most appropriate mode for novice programmers is the step-by-step mode, where the user is able to see every action during the visualization and read the corresponding explanation.

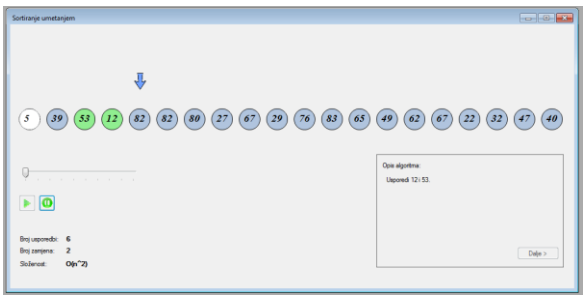

Figure 3. Array controls

For easier understanding of the actions that are happening inside the array, additional controls (shown in Figure 3) have been implemented. Controls like pointer (arrow) represent the end of the sorted part of the array (in the case of Insertion sort) or pivot (in the case of Quick sort). If necessary, each control can be used for any other algorithm, to either provide a more detailed explanation or support the visualization. Colors like green and white are used to flag the status of the element in the array. Green informs the user that two elements are being compared or swapped, while white notifies the user that the element has been sorted and is positioned at its final place inside the array. The last control is the interval marker. With the interval marker we can visualize the size of a sub-array in Shell Sort or intervals that are being sorted in Quick Sort. All the controls are dynamically created, which means that one or more controls can be used at the same time. Using the aforementioned controls provides the user with a full visualization of actions that are happening inside the array, making them easy to remember.

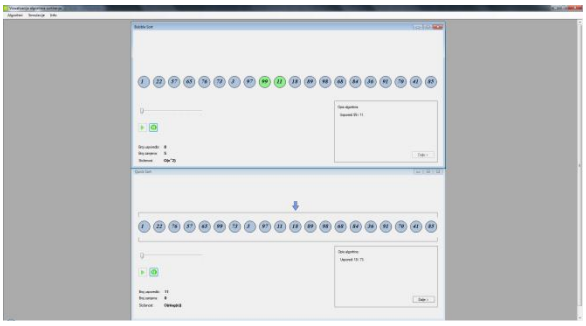

Figure 4. Simultaneous animation

Figure 4 illustrates one of ViSA's special features – simultaneous visualization of selected sorting algorithms. This feature allows the user to compare the speed and complexity of the sorting algorithms that have been selected. The user can sort the same array with any algorithm from the list or choose a new data set for a particular algorithm. If all algorithms are executed for the same initial array, it is possible to run a real time speed test. In that case, the visualization window is minimized to the bottom part of the form container, enabling the user to access any of the running algorithms. Given that some algorithms perform more comparisons at the beginning and finish with a quick series of swap actions, users can compare the number of actions for each algorithm in real time. In order to enable as much user interaction as possible and stimulate users' logical conclusions, ViSA is supplemented with algorithm analysis (Figure 5).

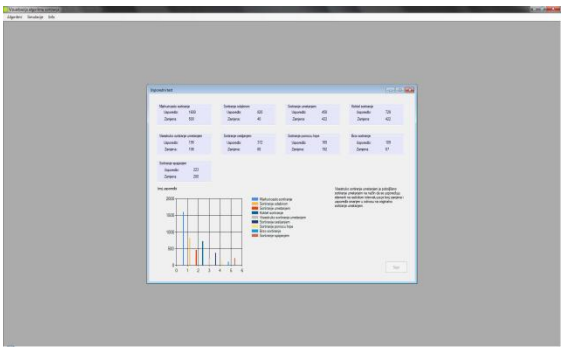

Figure 5. Algorithm analysis

Algorithm analysis allows the user to select a data set, executes all the sorting algorithms for the same array, resulting in a simple benchmark of sorting algorithms shown as a graph. The columns in such a graph represent the efficiency of sorting algorithms. By observing analysis results, the user can easily make a conclusion on the complexity of the compared sorting algorithms. The benchmark test can be run to a maximum of the integer array size, thus providing a very accurate method of comparing algorithms. The input data set can be constructed in the same manner as the visualization, but by using different array setups algorithm methods can be fully analyzed since some algorithms work faster or perform fewer element swaps if the array is formed in a certain way. For example, the Quick Sort algorithm will

not be executed in the same manner every time since the pivot element is randomly selected and the algorithm has to go through the whole array even though the array might already have been sorted. On the other hand, Bubble Sort is quite an old and slow algorithm that will still perform very well on an almost sorted array. An additional feature of algorithm analysis is that when the user drags a mouse pointer over a particular algorithm, a detailed explanation of the algorithm as well as its comparison with similar algorithms is provided.

IV. MODELLING THE ACCEPTANCE OF VISA

With an aim to investigate the perceived ease of use, perceived usefulness, behavioral intention, and attitude towards using ViSA, a pilot study was conducted. A total of 76 information science students participated in the study. The sample was composed of 65 male (85.5%) and 11 female (14.5%) students. The majority of them (94.7%) were enrolled in the second year of study. Data were collected with an online questionnaire survey. The average age of students was 20.39 (SD = 0.994). Before taking part in the survey, students had used ViSA during one lab-based session. The responses were modulated on a five point Likert scale (1-strongly agree, 5-strongly disagree). The Cronbach's alpha values ranged from .668 to .808, thus indicating sufficient reliability of the scale for exploratory research [23][26]. In order to ensure content validity, the questionnaire items were adopted from previous research on this topic [24][25].

As can be observed from the responses to survey items which are presented in Table 1 (Appendix A.), the majority of students perceived ViSA as a useful and easy to use tool aimed for learning sorting algorithms. For instance, in case of the survey item "Using ViSA would enable me to learn sorting algorithms much faster than learning from textbooks." 90.8% of the students responded with "Strongly agree" or "Mostly agree". Furthermore, 82.9% of the participants responded in the same way to the survey item "Using ViSA would help me to understand sorting algorithms much easier than during classes." Also, in case of the survey item "My interaction with ViSA would be clear and understandable." 98.7% of the students responded with "Strongly agree" or "Mostly agree". Finally, as many as 96.1% of the participants responded positively to the following statement: "It would be easy for me to become skillful at using ViSA." The aforementioned results indicate a positive attitude towards ViSA as well as the behavioral intention regarding its use.

V. CONLUSION

The purpose of our research was to develop an educational tool that would engage students in the learning process, helping them to acquire knowledge about well-known sorting algorithms. Visualization of sorting algorithms (ViSA) offers a full range of functionalities such as data set entry and animation control as well as the explanation and detailed algorithm analysis. Using ViSA, the learning curve and time spent learning and understanding should be significantly reduced. ViSA creates a bridge between students and algorithms in a manner that removes the need for writing the programming code, thus reducing the fear of programming. Once they have familiarized themselves with the algorithm, students would access programming with much more confidence. According to the results of our pilot study, ViSA is perceived as an educational tool whose usefulness and ease of use contribute to a positive attitude and behavioral intention regarding its use.

In our future work, we plan to extend ViSA with additional pedagogical features such as a pop-up form that would prompt the user with a question regarding a specific algorithm. Supplementing ViSA with such features may enhance its pedagogical usability and stimulate students' creative and logical thinking. Given that all forms and controls are created dynamically, we intend to supplement ViSA with visualization of other programming concepts such as data structures. Finally, we will also expand our research efforts related to the acceptance of ViSA.

REFERENCES

[1] D. Radošević, T. Orehovački, and A. Lovrenčić, "Verificator: Educational Tool for Learning Programming", Informatics in Education, vol. 8, no. 2, 2009, pp. 261-280.

[2] J. Bennedsen, and M. E. Caspersen, "Failure Rates in Introductory Programming", ACM SIGCSE Bulletin, vol. 39, no. 2, 2007, pp. 32-36.

[3] S. Al-Imamy, J. Alizadeh, and M.A. Nour, "On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process", In Proceedings of JITE, 2006, pp.271-283.

[4] T. Naps, J. Eagan, and L. Norton, "JHAVÉ: An Environment to Actively Engage Students in Web-based Algorithm Visualizations", In Proceedings of the 31st ACM SIGCSE Technical Symposium on Computer Science Education, Austin:ACM, 2000. pp. 109-113.

[5] A. Gomes, and A. J. Mendes, "Learning to program – difficulties and solutions". In: Proceedings of the International Conference on Engineering Education. Coimbra, Portugal, 2007 http://icee2007.dei.uc.pt/proceedings/papers/411.pdf

[6] S. Hansen, N. H. Narayanan, and M. Hegarty, "Designing Educationally Effective Algorithm Visualizations", Journal of Visual Languages and Computing, vol. 13, no. 3, 2002, pp. 291-317.

[7] G. P. Waldheim, "Understanding How Students Understand", Engineering Education, vol. 77, no. 5, 1987, pp. 306-308.

[8] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. 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", In Working group reports from ITiCSE on Innovation and technology in computer science education, Aarhus: ACM, 2002, pp. 131-152.

[9] M. Conway, S. Audia, T. Burnette, D. Cosgrove, and K. Christiansen, "Alice: lessons learned from building a 3D system for novices", In Proceedings of the SIGCHI conference on Human factors in computing systems, The Hague: ACM, 2000, pp. 486 – 493.

[10] A. W. Lawrence, "Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding. PhD thesis, Department of computer Science, Georgia Institute of Technology, 1993, http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA275135&Location=U2&doc=GetTRDoc.pdf

[11] Guido von Robot, http://gvr.sourceforge.net/

[12] J. Stasko, "Samba algorithm Animation System", http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html

[13] A. Zeller, "Animating data structures in DDD", In Proceedings of the SIGCSE/SIGCUE Program Visualization Workshop, 2000, Porvoo: ACM, pp. 69-78.

[14] A. I. Concepcion, N. Leach, and A. Knight, "Algorithma 99: an experiment in reusability & component based software engineering", ACM SIGCSE Bulletin, vol. 32, no. 1, 2000, pp.. 162-166.

[15] W. C. Pierson, and S. H. Rodger, "Web-based animation od data structures using JAWAA", ACM SIGCSE Bulletin, vol. 30, no. 1, 1998, pp. 267-271.

[16] H. Liberman, and C. Fry, "Zstep 95: A reversible, animated source code stepper", In Software Visualization--Programming as a Multimedia Experience, 1998, pp. 277-292.

[17] C. D. Hundhausen, and S. A. Douglas, "Low-Fidelity Algorithm Visualization", Journal of Visual Languages and Compunting 2002, vol. 13, no. 5, pp. 449-470.

[18] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing Programs with Jeliot 3, In Proceedings of the working conference on Advanced visual interfaces, Gallipoli: ACM, 2004. pp. 373-376.

[19] D. J. Barnes, and M. Kolling, "Objects First with Java: A Practical Introduction Using BlueJ", Prentice Hall; 2 edition, 2004.

[20] D. Radosevic, and T. Orehovacki, "An Analysis of Novice Compilation Behavior using Verificator", In Proceedings of the 33rd International Conference on Information Technology Interfaces (ITI), Cavtat: IEEE, 2011. pp. 325–330.

[21] G. Rößling, and B. Freisleben, "ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation", Journal of Visual Languages & Computing, vol. 13, no. 3, 2002, pp. 341-354.

[22] J. C. Bradley, A. C. Millspaugh, "Advanced Programming Using Visual Basic .NET", Mcgraw-Hill, 2nd edition, 2003.

[23] L. J. Cronbach, "Coefficient Alpha and the Internal Structure of Tests", Psychometrika, vol. 16, no. 3, 1951, pp. 297-334.

[24] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology", MIS Quarterly, vol. 13, no. 3, 1989, pp. 319-340.

[25] M. Gong, Y. Xu, Y. Yu, "An Enhanced Technology Acceptance Model for Web-Based Learning", Journal of Information Systems Education vol. 15, no. 4, 2004, pp. 365-374.

[26] J. C. Nunnally, "Psychometric Theory", Second Edition, McGraw Hill, New York, 1978.

[27] W. Dann, S. Cooper, and R. Pausch, "Using Visualization To Teach Novices Recursion", In Proceedings of the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Canterbury, England, 2001, pp. 109-112.

[28] M. Guzdial, and E. Soloway, "Log on education: teaching the Nintendo generation to program", Communications of the ACM, vol. 45, no. 4, 2002, pp. 17-21.

[29] R. B. Findler, C. Flanagan, M. Flatt, S. Krishnamurthi, and M. Felleisen, "DrScheme: A pedagogic programming environment for scheme", Lecture Notes in Computer Science, vol. 1292, 1997, pp. 369-388.

[30] B. Erwin, M. Cyr, and C. Rogers, "LEGO Engineer and RoboLab: Teaching Engineering with LabVIEW from Kindergarten to Graduate School", International Journal of Engineering Education, vol. 16, no. 3, 2000, pp. 181-192.

APPENDIX A.

TABLE I. SUMMARY OF MEASUREMENT SCALES

Items	**Mean**	**SD**
Perceived Usefulness (Cronbach's α = 0.668)		
Using ViSA would enhance my efficiency in understanding of sorting algorithms.	1.41	0.570
Using ViSA would enhance my effectiveness in understanding of sorting algorithms.	1.55	0.551
Using ViSA would enable me to learn sorting algorithms much faster than learning from textbooks.	1.61	0.750
Using ViSA would help me to understand sorting algorithms much more easily than during classes.	1.89	0.741
I would find ViSA useful for learning sorting algorithms.	1.37	0.538
Perceived Ease of Use (Cronbach's α = 0.808)		
Learning to operate ViSA would be easy for me.	1.36	0.534
I would find it easy to get ViSA to do what I want it to do.	1.64	0.725
My interaction with ViSA would be clear and understandable.	1.37	0.512
I would find ViSA to be flexible to interact with.	1.96	0.720
It would be easy for me to become skillful at using ViSA.	1.45	0.575
I would find ViSA easy to use	1.39	0.591
Attitude Towards Use (Cronbach's α = 0.777)		
I think it would be very useful to use ViSA in programming related courses.	1.43	0.574
In my opinion, it would be very desirable to use ViSA in programming related courses.	1.42	0.572
I think it would be very wise if students used ViSA for learning sorting algorithms.	1.43	0.618
Behavioral Intention (Cronbach's α = 0.807)		
I would recommend ViSA to my colleagues.	1.49	0.702
I would recommend ViSA to novice programmers.	1.33	0.661
I intend to use ViSA for learning and recapitulation of sorting algorithms.	2.16	0.994
To the extent possible, I would use ViSA as part of courses related to the programming.	1.67	0.806
When I need to make a comparison and analysis of sorting algorithms, I intend to use ViSA.	1.96	0.855

View publication stats