A Design of Automatic Visualizations for Divide-and-Conquer Algorithms

J. Ángel Velázquez-Iturbide, Antonio Pérez-Carrasco, Jaime Urquiza-Fuentes
Departamento de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos, C/ Tulipán s/n, Móstoles 28933, Madrid, Spain

angel.velazquez@urjc.es

Abstract

The paper addresses the design of program visualizations adequate to represent divide-and-conquer algorithms. Firstly, we present the results of several surveys performed on the visualization of divide-and-conquer algorithms in the literature. Secondly, we make a proposal for three complementary, coordinated views of these algorithms. In summary, they are based an animation of the activation tree, an animation of the data structure, and a sequence of visualizations of the substructures, respectively.

1 Introduction

An informal distinction is commonly accepted between program visualization and algorithm animation. The former term describes external representations that are closely tight to program source code. The latter term refers to external representations of the abstract behaviour of a piece of program, typically an algorithm. Given the lower abstraction level of program visualizations, they are frequently generated automatically, whereas the higher abstraction level of algorithm animations forces human intervention.

As effort is one of the main reasons for instructors not to be using visualization software in education, it is worthwhile to further explore different directions for program visualization (Naps et al., 2003). We have addressed a line of research consisting in generating program visualizations based on their underlying algorithm design techniques, e.g. divide-and-conquer or backtracking. Consequently, a student who wants to understand and analyze the behaviour of an algorithm of a common design technique could generate expressive, automatic visualizations of the algorithm.

We have designed an implementation framework (Fernández-Muñoz et al., 2007) to develop several visualization systems, one per design technique. To illustrate the feasibility of this framework, a first system was implemented to visualize recursion, called SRec (Velázquez-Iturbide et al., 2008). Now, we are addressing the design of a visualization system for a proper algorithm design technique, namely divide-and-conquer.

The goal of this paper is to present the design of visualizations adequate for the divide-and-conquer technique. In the second section, we present the result of several studies we performed on the visualization of divide-and-conquer algorithms in the literature. The third section contains our proposal, consisting in three complementary, coordinated views. Finally, we summarize our conclusions and future work.

2 A Survey of Visualizations of Divide-and-Conquer Algorithms

In this section, we show the results of several studies we performed on visualizations of divide-and-conquer algorithms.

We assume that the definition of divide-and-conquer algorithms is well known, but we list here the terms used in the rest of the paper. A problem solved by divide-and-conquer is decomposed into subproblems. They are recursively solved, resulting in subsolutions whose combination gives place to the solution of the original problem.

Divide-and-conquer algorithms often traverse and manipulate a data structure. Each subproblem is constrained to a part of the structure, i.e. a substructure. We only deal here

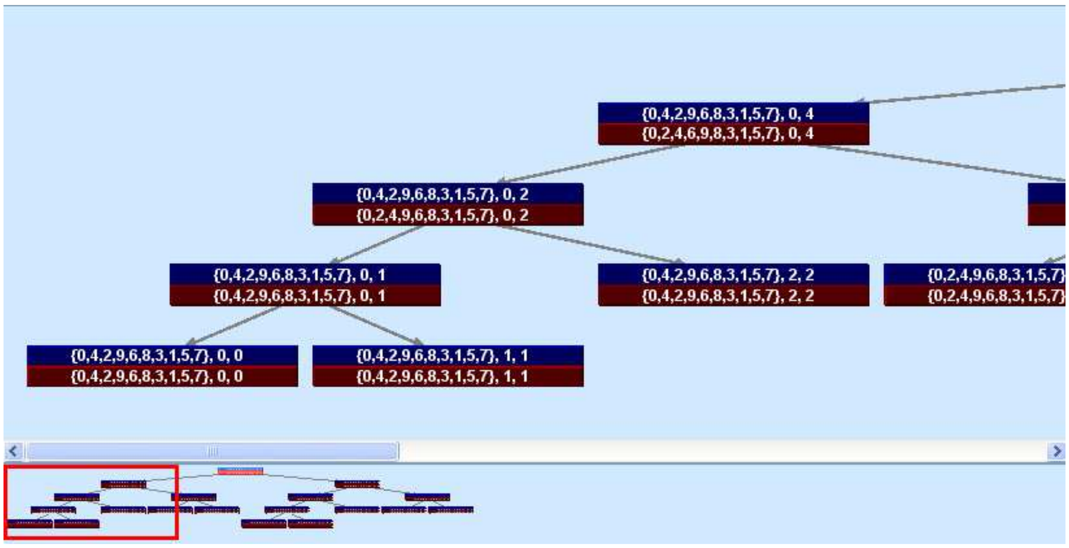


Figure 1: Activation tree for mergesort of {0,4,2,9,6,8,3,1,5,7} displaying array contents and indices

with one- and two-dimensional arrays, thus we often use the terms (sub)arrays, (sub)vectors and (sub)matrices. The most common and efficient way of delimiting subarrays is by using a range, defined with a lower and a higher index.

2.1 Visualizations of Recursion

Divide-and-conquer algorithms are a particular case of recursive algorithms. As a consequence, in a first approach, we tried to make use of visualizations for recursive algorithms. These visualizations are well known in CS: activation (or recursion) trees, the execution stack, traces, and multiple copies (of either code or variables). These visualizations are not equally effective for lineal and for multiple recursive algorithms. In particular, activation trees are more useful to display the behaviour of multiple recursive algorithms, e.g. divide-and-conquer algorithms.

Activation trees have limitations for divide-and-conquer algorithms. They are most effective for algorithms with a few, simple parameters or results. However, they are not as effective for larger data structures. The following two figures illustrate this inadequacy for mergesort. Both figures have been generated with SRec (Velázquez-Iturbide et al., 2008). The system allows the user to select the parameters or results to display; when a method does not return any value but produces side-effects, the original and final value of the parameters are displayed. It also has several facilities (zoom+panning, overview+detail) to handle large-scale activation trees. Finally, a user-defined colouring scheme can be used to differentiate input/output values, and the status of a call in the global process (executed, active or pending). In spite of all of these facilities, the resulting visualizations are not satisfactory.

Fig. 1 shows how the textual representation of arrays produces long nodes and therefore wide and shallow activation trees, difficult to browse and comprehend. In addition, the display of the complete array in every recursive call makes difficult identifying its corresponding subproblem and subsolution. Fig. 2 shows that omitting arrays from the nodes produces a more compact display, but changes of the array contents during the sorting process are not visible.

2.2 Visualizations in Algorithm Animations

Algorithm courses contain many divide-and-conquer algorithms, mainly mergesort and quicksort. Consequently, many creators of animation systems have developed animations for these algorithms. We have reviewed and analyzed the display of quicksort animations contained in

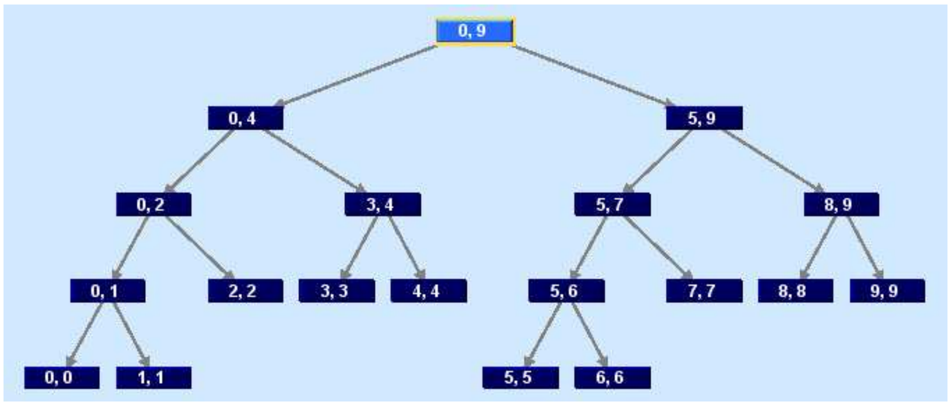


Figure 2: Activation tree for mergesort of {0,4,2,9,6,8,3,1,5,7} only displaying array indices

the reference books by Diehl (2007) and Stasko et al. (1998). Diehl (2007) shows one display (figs. 1.5 and 4.4) and Stasko et al. (1998) contains a higher number (pp. 41, 42, 91, 151, 158, 254, 374, 377, 379). This study reveals that some graphical representations are not useful because of several reasons:

- Some generic graphical representations are too poor. For instance, just displaying the data structure to manipulate is not expressive enough (see p. 42).
- A representation of a vector where each cell is displayed proportional to its size only is useful for certain problems (e.g. sorting). We find representations of cells as vertical bars (figs. 1.5 and 4.4), horizontal bars (pp. 374, 377) or in a "dots view" (pp. 41, 151, 379).

However, these animations also contain elements that can be successfully generalized to other divide-and-conquer algorithms:

- They use boxes to enclose the subarray handled by each recursive call (figs. 1.5 and 4.4).
- They classify the elements with respect to their status in the algorithm history by using shapes (pp. 41, 91, 158) or colours (figs. 1.5 and 4.4, pp. 91, 158, 374).
- The partition tree (pp. 41, 91, 158) is a variation of an activation tree that successfully combines recursion and vector representations. In summary, it consists of a tree isomorphic to an activation tree, where an element of the vector is displayed either as a node of the tree (when it is at its final position) or as a part of a subvector (when it has not been processed yet).

A different analysis of the visualization of recursive algorithms can be found at Stern and Naish (2002). They propose differentiating three kinds of algorithms, depending on how they handle a data structure (namely, algorithms that modify, traverse or construct it), rather than considering their recursion scheme. However, it is not obvious whether their visualizations can be generalized: the visualizations they propose contain vectors for the first class of algorithms, and trees for the other two classes. The visualization included in the article for the former class corresponds to a divide-and-conquer algorithm (namely, quicksort). It displays horizontally the vector and underlies it with horizontal bars that mirror recursive calls.

2.3 Visualizations in Textbooks

A comprehensive study on the visualization of divide-and-conquer algorithms must consider representations used by CS instructors. Consequently, we made a study of some of the most prestigious textbooks on design and analysis of algorithms (Fernández-Muñoz and Velázquez-Iturbide, 2006). The selection was necessarily arbitrary, but we consider it was representative of high-quality textbooks on algorithms (Aho et al., 1983; Alsuwaiyel, 1999; Baase and Gelderl, 1988; Brassard and Bratley, 1996; T.H. Cormen and Rivest, 2001; Gonnet and Baeza-Yates, 1991; Goodrich and Tamassia, 2001; Horowitz and Sahni, 1978; Johnsonbaugh and Schaefer, 2004; Levitin, 2003; Manber, 1989; Parberry, 1995; Sahni, 2000; Weiss, 1999)

We summarize our findings, after discarding visualizations specific of any problem:

- It is common to include a visualization illustrating the inductive definition of the recursive algorithm by displaying its elements: problem, subproblems, subresults, and result.
- It is common to include a visualization of the activation tree. There are many variants in its graphical representation:
  - Visualize either a single tree or two parallel trees, where the second one illustrates the auxiliary operation (e.g. partitioning in quicksort).
  - Display either the activation tree or a sequence of visualizations of the data structure. Notice that the latter is an implicit tree, since it corresponds to its traversal.
  - Display either the delimiting indices or the contents of subarrays.
  - Display either the original or the final values contained in a data structure.
  - Display the activation tree in an either ascending or descending layout. We also find the join display of both, representing the advance and return phases of recursion.
- It is common to include a visualization of the data structure, complemented with some representation of the partitioning performed by the divide-and-conquer algorithm:
  - By means of nested boxes enclosing subarrays.
  - As a sequence of successive states of the substructures handled by the successive calls. Each substructure is either aligned according to its delimiting indices or laid out isomorphic to the recursion tree.

3 A Proposal for Automatically Generated Visualizations of Divide-and-Conquer Algorithms

Based on the findings of the previous section, we propose to three (coordinated) views of the behaviour of divide-and-conquer algorithms. We have imposed an additional requirement on our visualizations: they must be applicable to both one- and two-dimensional arrays, i.e. vectors and matrices.

- An animation based on the activation tree. Each node is complemented with a visualization of the substructure it focuses on.
- An animation based on the data structure. It is complemented with a schematic diagram of its partitioning by the algorithm.
- A sequence of visualizations of the substructures.

We elaborate these views in the following subsections.

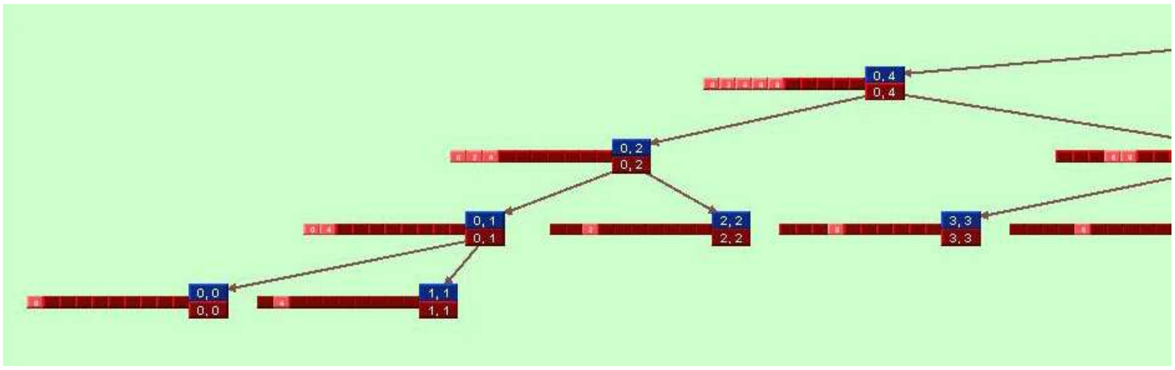


Figure 3: Visualization for mergesort of {0,4,2,9,6,8,3,1,5,7} based on the activation tree

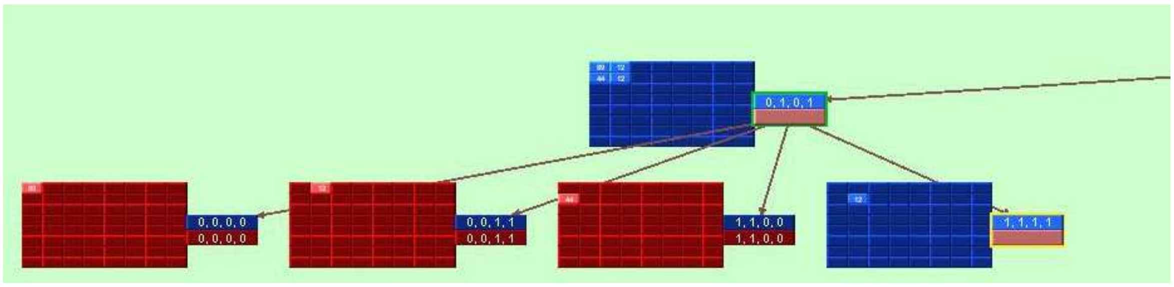


Figure 4: Visualization for transposition based on the activation tree

3.1 Animation Based on the Recursive Process

Fig. 3 shows an example of this view. Opposed to Fig. 1, each array is visualized once in each node. In addition, the application of a user-defined colouring scheme to the array allows to determine at a glance which subarray is the focus of the recursive call, as well as whether it is the subarray state at the entry or exit of the call. For the former issue, we recommend using different tones of the same colour, and for the second one, different colours. In the figures, the blue and red colours are respectively used to represent input and output values.

This view can be applied to matrices as well. Fig. 4 corresponds to a divide-and-conquer algorithm transposing a square matrix (an inefficient one!).

3.2 Animation Based on the Data Structure

This view provides a discrete animation of the successive states of the data structure to manipulate. It displays vectors and matrices in a conventional format. A set of bars is displayed below, that mirror the recursive process by underlying the subarray delimited by each recursive call. Fig. 5 shows an example of this view.

A colouring scheme is applied to the underlying bars, as well as to their associated subarrays. A first colour (red in Fig. 5) is used to mark recursive calls whose execution is over, as well as their corresponding subvectors. A second colour (blue in Fig. 5) is used for recursive calls whose execution is pending, as well as their corresponding subvectors. Tones of the two

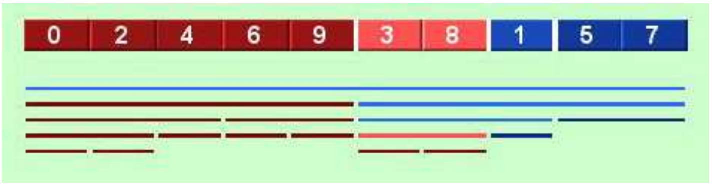
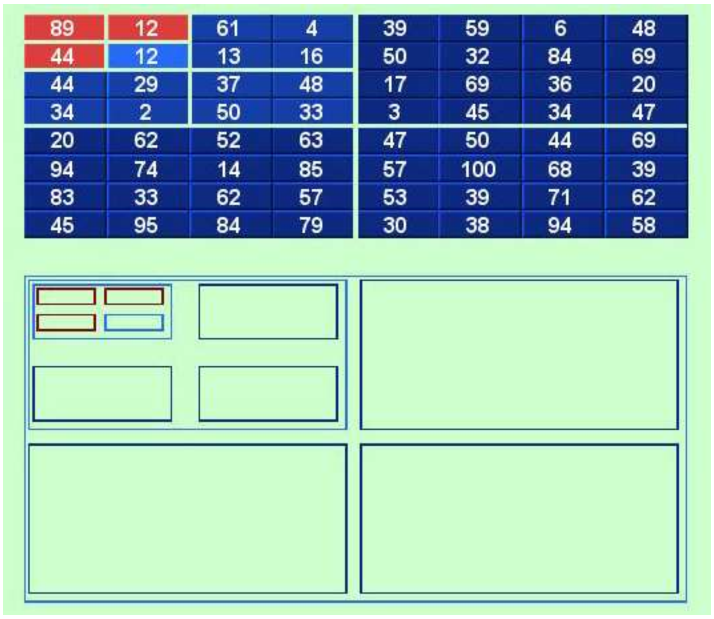


Figure 5: Visualization for mergesort of {0,4,2,9,6,8,3,1,5,7} based on the data structure



89	12	61	4	39	59	6	48
44	12	13	16	50	32	84	69
44	29	37	48	17	69	36	20
34	2	50	33	3	45	34	47
20	62	52	63	47	50	44	69
94	74	14	85	57	100	68	39
83	33	62	57	53	39	71	62
45	95	84	79	30	38	94	58

Figure 6: Visualization for transposition based on the data structure

colours are used to represent the distance of each call in the activation tree to the active call. The active call is always coloured light, and more distant nodes are coloured darker. In Fig. 5, the left part of the array is already sorted, being the active call focused at the subarray 3,8 and about to exit.

This view can be applied to matrices as well. The set of horizontal bars is replaced by a set of nested boxes cueing submatrices. Fig. 6 illustrates this for the algorithm to transpose a square matrix. Here, the algorithm only has completed three base cases and is focused on the fourth one.

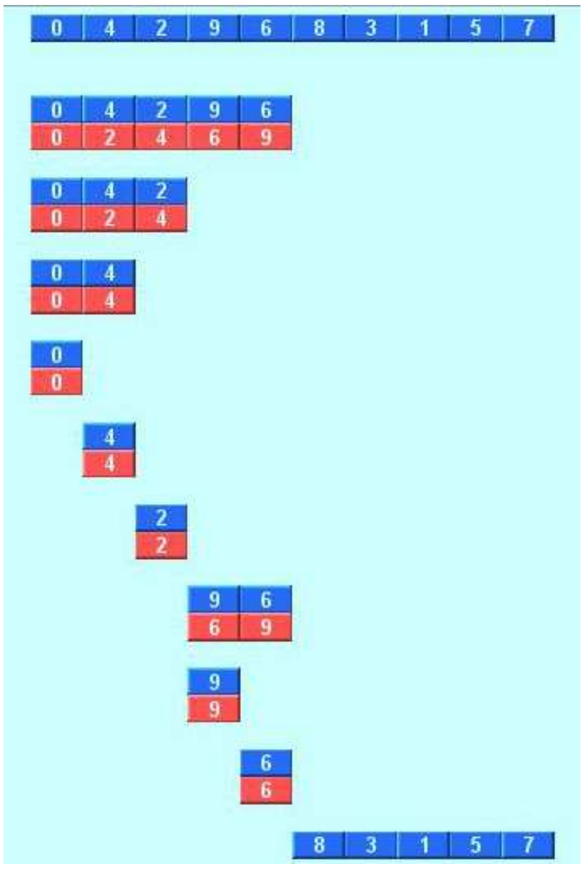
3.3 Sequence of Visualizations of the Data Structure

A third view displays a sequence of visualizations of the data structure, displayed top-down. Every time a recursive call is invoked, a new visualization of the array is displayed at the bottom of this view. In order to highlight the recursive process, each line only contains the subarray focused by its associated call, indented according to its delimiting indices.Every time a recursive call exits, a visualization of the resulting subarray is displayed below the original subarray displayed on call entry. Again, the use of colours allows differentiating them. Fig. 7a shows this view for the mergesort algorithm. The left part of the array is already sorted and a call has been made to sort its right half.

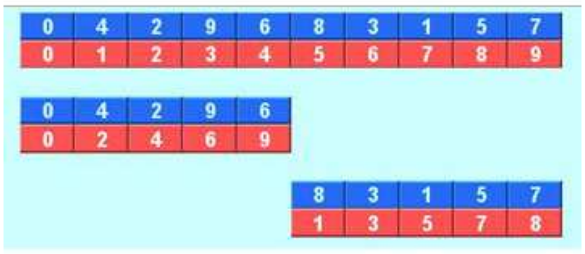
The main advantage of this view is that it allows generating a visualization that mirrors the inductive definition of the recursive algorithm. Fig. 7b illustrates this feature for mergesort. By selecting the animation control to jump over a recursive call and hiding the visualization of its underlying displays, the resulting display just consists of the original array, the two subarrays focused by the two recursive calls, the output subarrays of these recursive calls, and the final array.

4 Conclusions and Future Work

Custom visualizations for particular algorithms, as shown in algorithm animations, can be the most expressive. However, the effort necessary for manual generation of each particular animation is prohibitive. Therefore, we argue for using program visualizations as an alternative, effortless approach.



(a)



(b)

Figure 7: Full (a) and simplified (b) sequence of visualizations for mergesort of {0,4,2,9,6,8,3,1,5,7}

We have shown two results in this paper. Firstly, we have presented the results of examining visualizations of recursion as well as divide-and-conquer visualizations available at the literature. Secondly, we have proposed three program visualizations for divide-and-conquer algorithms. Two of them are respectively based on the animation of activation trees and of the data structure; both are mixed in the sense of displaying code and data elements. A third visualization is a sequence of substructures, and is capable of illustrating the inductive definition of the algorithm.

We have implemented a working prototype of the design presented here. However, more work is necessary to become a fully operational system. Usability evaluations performed by experts (i.e. instructors) and in sessions with students are important to assess their validity, as described in (Velázquez-Iturbide et al., 2008) for the SRec system.

References

V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison Wesley, 1983.

M.H. Alsuwaiyel. *Algorithms Design Techniques and Analysis.* World Scientific, 1999.

S. Baase and A. Van Gelderl. *Computer Algorithms.* Addison Wesley, 1988.

G. Brassard and P. Bratley. *Fundamentals of Algorithmics.* Prentice-Hall, 1996.

S. Diehl. *Software Visualization.* Springer-Verlag, 2007.

L. Fernández-Muñoz, A. Pérez-Carrasco, J.Á. Velázquez-Iturbide, and J. Urquiza-Fuentes. A framework for the automatic generation of algorithm animations based on design techniques. In E. Duval, R. Klamma, and M. Wolpers, editors, *Creating New Learning Experiences on a Global Scale - EC-TEL 2007*, volume 4753 of *LNCS*, pages 475–480, 2007.

L. Fernández-Muñoz and J.Á. Velázquez-Iturbide. A study on the visualization of algorithm design techniques (in spanish). In C. Bravo M.Á. Redondo and M. Ortega, editors, *VII Congreso Internacional de Interacción Persona-Ordenador*, pages 315–324, 2006.

G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures in Pascal and C.* Addison-Wesley, 2nd edition, 1991.

M.T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java.* John Wiley & Sons, 2nd edition, 2001.

E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms.* Pitman, 1978.

R. Johnsonbaugh and M. Schaefer. *Algorithms.* Pearson Prentice Hall, 2004.

A. Levitin. *The Design and Analysis of Algorithms.* Addison-Wesley, 2003.

U. Manber. *Introduction to Algorithms.* Addison-Wesley, 1989.

T.L. Naps, G. Roessling, 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. *SIGCSE Bulletin*, 35(2):131–152, 2003.

I. Parberry. *Problems on Algorithms.* Prentice Hall, 1995.

S. Sahni. *Data Structures, Algorithms and Applications in Java.* McGraw-Hill, 2000.

J. Stasko, J. Domingue, M.H. Brown, and B.A. Price, editors. *Software Visualization.* The MIT Press, 1998.

L. Stern and L. Naish. Visual representations for recursive algorithms. In *33th SIGCSE Technical Symposium on Science Education, SIGCSE 2002*, pages 196–200, 2002.

C.E. Leiserson T.H. Cormen and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

J.Á. Velázquez-Iturbide, A. Pérez-Carrasco, and J. Urquiza-Fuentes. Srec: An animation system of recursion for algorithm courses. In *13rd Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE 2008*, page In press, 2008.

M.A. Weiss. *Data Structures and Algorithm Analysis in Java.* Addison-Wesley, 1999.