Tools for Web-Based Sorting Animation

Herbert L. Dershem
Department of Computer Science
Hope College
Holland, MI 49422-9000
dershem@cs.hope.edu

Peter Brummund
Department of Computer Science
Taylor University
Upland, IN
brummund@taylor.edu

Abstract

There is a long and rich tradition of sort algorithm animations that have been used effectively in instruction. This paper describes a new tool that combines the best of this tradition with the advantages of being in the form of an applet for use on the World Wide Web and of animating the code of the algorithm in concert with the animation of the data. In addition, this tool facilitates student-designed animations that are useful for the debugging of student-written sort algorithms. Recursive sort algorithms are made more accessible through the use of a special feature of the tool that animates the recursion.

1. Introduction

Animation of sort algorithms has a long history in computer science education, starting with the Sorting Out Sorting video in 1981 [1]. This video introduced the idea of representing data elements in a sort by bars with length of the bar corresponding to the sort key value of the element. It also used highlighting to indicate elements being compared and concluded with a race of the nine algorithms examined.

Sorting Out Sorting set such a high standard that as algorithm animation systems such as GAIGS [2], Tango [3], and Balsa [4] were developed, they focused on animations of algorithms other than sorting, although sorting animations were developed in all of these. None of these presented a significant improvement over Sorting Out Sorting beyond their flexibility to work on user-provided data.

The advent of the World Wide Web and Java applets has opened up a new era for algorithm animations, permitting remote interactive access to such animations with platform independence and an ability to link animations to text, sound, and video[5]. Most algorithm animation systems are presently either on the web or in the process of migrating there.

Algorithm animation can be coupled with code animation by including a separate window or frame where the code for the algorithm is found. Animators using this technique highlight the line of code being executed in the algorithm simultaneously with the algorithm animation activity. This permits students to see how the execution of program code corresponds to the steps of the animation. This coupling with code animation has been implemented in JCAT[6] using pseudocode, ZStep 95[7] using LISP, and in AdaVision[8] using Ada.

Systems of algorithm animation that facilitate user-designed animations are a more recent innovation. This approach has recently been advocated as pedagogically advantageous by Stasko[9].

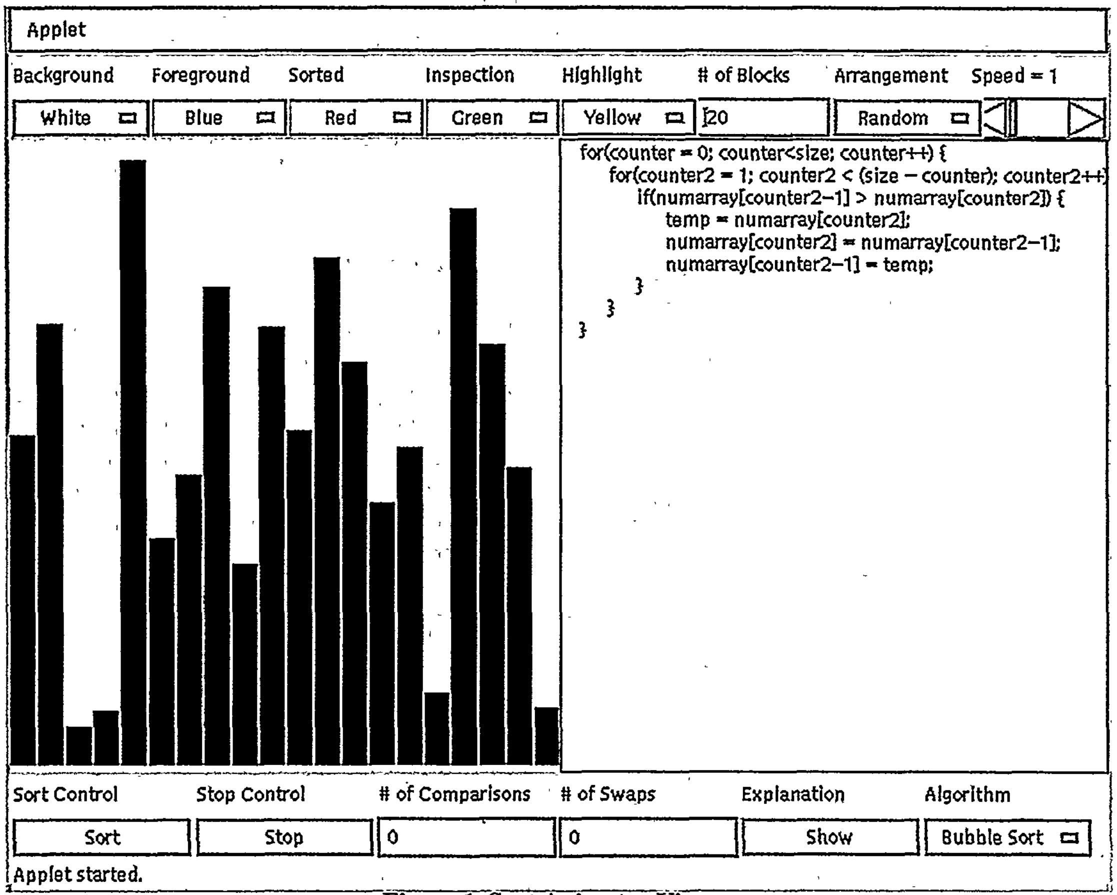
The project described in this paper adds the recent advances of web access, code coupling, and user design to the proven features of the Sorting Out Sorting video, resulting in an effective tool for teaching and learning sort algorithms. In addition, an animation technique is added to enhance student learning of how recursion occurs in sorting algorithms.

2. The Sort Animator Applet

The Sort Animator is a Java applet that provides a split window showing the sort animation in the left frame and the code animation in the right frame. The basic structure of Sort Animator is illustrated in Figure 1.

The vertical bars in the left frame represent the elements to be sorted with the horizontal coordinate showing the position or index of the element in the sort list and the vertical coordinate representing the magnitude of the element's sort key. As the algorithm executes, the bars move as data movement occurs in the sort list. The line of code being executed at each point in time is highlighted inside the right frame.

The user has many controls over this animation, all of which can be modified at any point of the algorithm's execution. The user may specify five different colors to the sort animator, using buttons in the upper left portion of the window. These specify the background color for the sort animation, three bar colors, and the color used to highlight lines of code. The bar colors are for bars that are presently being inspected


Applet

Background	Foreground	Sorted	Inspection	Highlight	# of Blocks	Arrangement	Speed = 1
White	Blue	Red	Green	Yellow	20	Random	

```
for(counter = 0; counter<size; counter++) {
   for(counter2 = 1; counter2 < (size - counter); counter2++
      if(numarray[counter2-1] > numarray[counter2]) {
         temp = numarray[counter2];
         numarray[counter2] = numarray[counter2-1];
         numarray[counter2-1] = temp;
      }
   }
}
```

Sort Control	Stop Control	# of Comparisons	# of Swaps	Explanation	Algorithm
Sort	Stop	0	0	Show	Bubble Sort

Applet started.

Figure 1. Sort Animator View

by the algorithm, those that are already in their sorted position, and those that fit into neither of these categories. These four colors may be modified at any time during the animation without disrupting the algorithm.

The user may also specify the number of elements being sorted. This number may be as large as 900. The arrangement of the data can be random, ascending, or descending, depending on the user's selection. Changing the number of blocks or the arrangement results in an immediate halt of the present sort execution and a restart with the new parameter in force.

The speed control is in the upper right corner of the window and manages the speed of the animation. A slide bar determines the speed of the algorithm. If the speed is set to zero, the algorithm comes to a temporary halt. By manipulating this bar appropriately the user can observe the algorithm in detail by stepping through under mouse control or by running the algorithm at a very slow speed. Once the user has grasped the general approach of the algorithm, a faster speed can be selected for completion of the algorithm. High speed animation may also enhance algorithm understanding by giving a better overview of the sorting strategy.

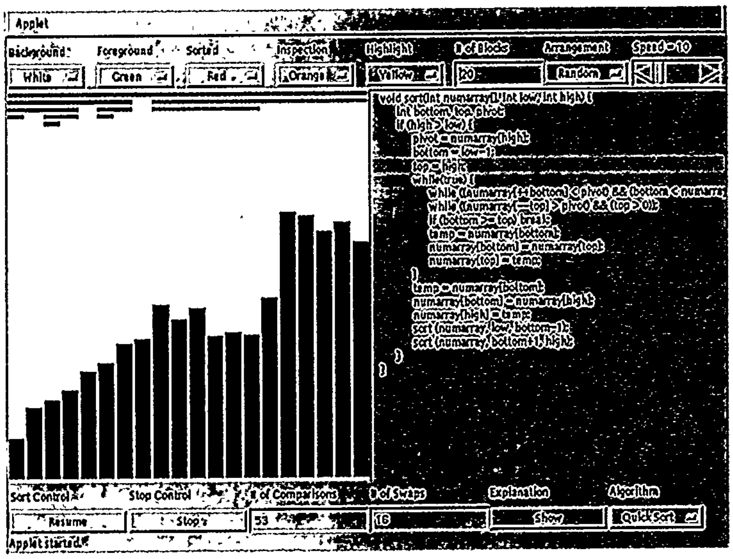
The execution of the algorithm is controlled by the two buttons in the lower left portion of the window. The "Sort" button begins the sort animation and its label changes to "Pause" when the animation is active. The "Stop" button halts the sort with no possibility of resuming the interrupted sort, although another sort can be initiated following "Stop" button selection.

When the explanation button is selected, a window appears that contains a text-based description or explanation of the sort algorithm. The Algorithm button permits the selection of a sort algorithm. Presently there are six algorithms provided although more will be added in the near future.

A running display of the number of comparisons and swaps is shown at the bottom of the window. The code for the algorithm is in the right frame. The lines of code are highlighted as they are executed by the animation. The code currently provided with the Sort Animator is in Java, although this need not be the case. Code in any language could be used to express the algorithms, including pseudo-code.

3. Animation of Recursion

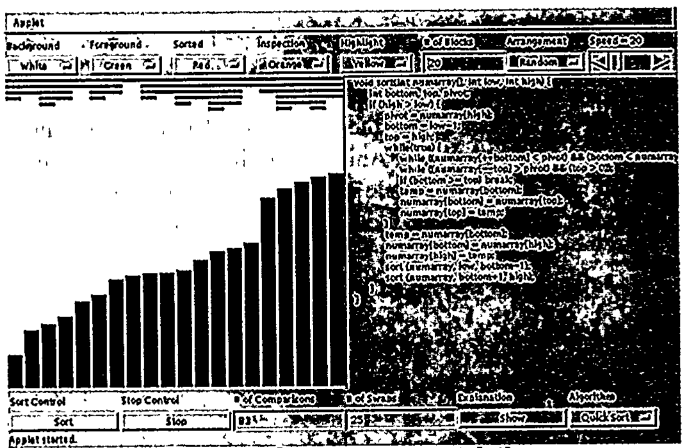
A useful feature of the Sort Animator is its manner of displaying recursive sort algorithms. A collection of horizontal bars is added at the top of the left frame to indicate the level of recursion and the part of the sort list that is sorted by each recursive call. The vertical position of the bar in used to indicate the level of recursion with the top bar drawn when the initial call is made, a bar underneath the first bar at the second level drawn when the first recursive call is made, and each succeeding call resulting in a bar drawn one level further down. The horizontal extent of each bar will exactly cover the portion of the sort list that is assigned to the corresponding recursive call.



```
void sort(int numarray[], int low, int high) {
    int bottom, top, pivot;
    if (high > low) {
        pivot = numarray[high];
        bottom = low-1;
        top = high;
        while(true) {
            while (numarray[++bottom] < pivot && (bottom < numarray
            while (numarray[--top] > pivot && (top > 0));
            if (bottom >= top) break;
            temp = numarray[bottom];
            numarray[bottom] = numarray[top];
            numarray[top] = temp;
        }
        temp = numarray[bottom];
        numarray[bottom] = numarray[high];
        numarray[high] = temp;
        sort (numarray, low, bottom-1);
        sort (numarray, bottom+1, high);
    }
}
```

Figure 2. Quicksort animation in progress

Figure 2 shows a quicksort animation in progress. Each horizontal bar at the top of the left frame represents a recursive call that has been made. The top bar is the original sort call on the entire list. The two shorter bars at the second level are the two recursive calls made from the first activation of procedure sort. The left of those two is darker indicating it has already been completed. At the third level, the two darkened bars to the left indicate that those have been completed while the lighter colored bar on the right is the sort range that is currently active. The remaining bars at levels four and five indicate further completed sort calls. In Figure 2, 6 calls have been completed and 3 are still active. At any time, the lowest lighter-colored bar represents the range of the active sort.



```
void sort(int numarray[], int low, int high) {
    int bottom, top, pivot;
    if (high > low) {
        pivot = numarray[high];
        bottom = low-1;
        top = high;
        while(true) {
            while (numarray[++bottom] < pivot && (bottom < numarray
            while (numarray[--top] > pivot && (top > 0));
            if (bottom >= top) break;
            temp = numarray[bottom];
            numarray[bottom] = numarray[top];
            numarray[top] = temp;
        }
        temp = numarray[bottom];
        numarray[bottom] = numarray[high];
        numarray[high] = temp;
        sort (numarray, low, bottom-1);
        sort (numarray, bottom+1, high);
    }
}
```

Figure 3. Quicksort at completion

This technique for illustrating recursion has the added advantage that it provides a history of the sort as well. Figure 3 shows the completion of the sort that was in progress in Figure 2. The horizontal bars reflect the history of the quicksort with the partitions at each level clearly identified. The pivot element at each level is also identified by the position of the space between the bars.

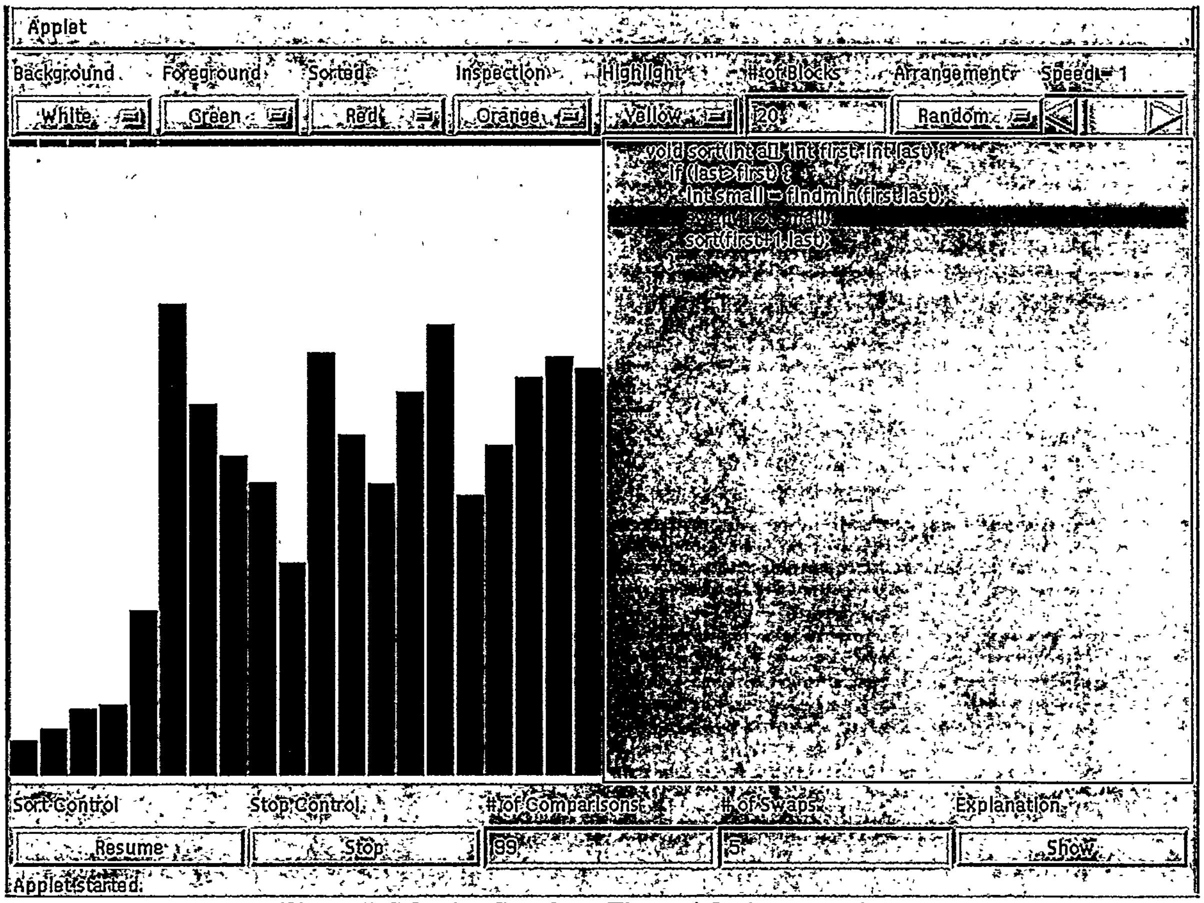
4. Sort Animation Builder

The Sort Animation Builder uses the same animation environment as the Animator. The sort algorithm is provided by the user along with special instructions to direct the animation. To illustrate the simplicity of building an animation, an example is shown in Figure 4. This is the code that the user would provide for a simple recursive selection sort with the animation control statements accompanied by explanatory comments.

```
import java.awt.*;

class UserThread extends SortThread {
  static int level = 0;

  public UserThread(SortClass temp) {
    super(temp);
  }

  int findmin(int first, int last) {
    int minloc = first;
    int i;
    for (i=first+1; i<=last; i++) {
      if (greater(minloc,i))   // greater compares elements at minloc and i
        minloc=i;
    }
    return minloc;
  }

  void sort(int first, int last) {
    drawBar(first,last,0,sorter.swapcolor);  // draws horizontal bar
    skipStatements(0,1);         // highlights statements 0 and 1 with delay
    if (last>first) {
      int small = findmin(first,last);
      setColorAt(small,sorter.swapcolor); // Sets color of element at small
      skipStatements(2,2);                // highlights statement 2 with de
      skipStatements(3,3);                // highlights statement 3 with de
      highlight(3);                       // resets highlight on statement
      swap(first,small);                  // swaps elements first and small
      markSorted(first);                  // colors first element sorted
      dehighlight(3);                     // dehighlights statement 3
      skipStatements(4,4);                // highlights statement 4 with de
      drawBar(first,last,0,sorter.backcolor); // draws hor bar from first t
                                          //     last at depth 0 with backc
      drawBar(first,first,0,sorter.sortedcolor); // draws hor bar from firs
                                          // first at depth 0 with sorted c
      sort(first+1,last);
    }
    else {
      markSorted(first);                  // colors first element sorted
      drawBar(first,first,0,sorter.sortedcolor); // draws hor bar from firs
                                          // first at depth 0 with sorted c
    }
    skipStatements(5,6);          // highlights statements 5 and 6 with de
  }

  public void runmain() {
    sort(0, getSize()-1);
  }
}
```

Figure 4. Sort Animator Code for Selection Sort

To build a sort, the user provides a `runmain()` function in the `UserThread` class. This class inherits all of the animation functions from its parent class, `SortThread`. The functions `setColorAt`, `color`, `greater`, and `swap` control the animation of the data elements, the function `drawBar` controls the horizontal bars that illustrate recursion, and the functions `skipStatements`, `highlight`, and `dehighlight` control the code animation. Other functions are available beyond those used in this illustration. Figure 5 shows a mid-sort window generated by the code in



Background White
Foreground Green
Sorted Red
Inspection Orange
Highlight Yellow
#of Blocks 20
Arrangement Random
Speed = 1

```
void sort(int a[], int first, int last) {
    if (last > first) {
        int small = findmin(first, last);
        swap(first, small);
        sort(first+1, last);
    }
}
```

Sort Control Resume
Stop Control Stop
#of Comparisons 99
#of Swaps 5
Explanation Show

Figure 5. Selection Sort from Figure 4 during execution

Figure 4. The code used for the code animation in the right frame is provided by the user in a separate file.

5. Conclusions

The Sort Animator and Sort Animation Builder provide a number of significant benefits to students learning sort algorithms and to their teachers as well. These tools are Java applets that can be run over the World Wide Web, making them accessible to students in a variety of settings. This includes the closed laboratory, the open laboratory, and on their own computer systems.

The animation of the algorithm's code along with its visual animation gives the student an explanation of each step as it is being carried out. This enhances the understanding of both the code and the algorithm as well as how they correspond to each other. The Sort Animation Builder permits the teacher and the student to implement code and algorithm animations themselves

These tools can be used in a variety of ways in courses where sort algorithms are discussed. First, they are valuable for classroom demonstrations. The instructor, while describing the sort algorithm, can step through its execution, describing the role of each pertinent step and addressing student questions. The Sort Animation Builder permits the instructor to implement algorithms or code from the textbook or algorithms of the instructor's own choosing.

A second use of these tools is in student laboratory exercises. In such exercises, students can be asked to do comparative, empirical analysis of algorithms, aided by the Sort Animator's view of the count of key operations. Also, since the animations are run as Java threads, it is possible for the students to conduct sort races to observe the relative efficiency of sorts. Another tool sometimes used in labs is to have the students observe the sort animation without the accompanying code, and identify from the animation the sort algorithm under observation from among those studied in class. It is also possible, within a lab setting, to have students observe the different behavior of the sort algorithms when the sort list is in order before the sort algorithm begins.

A final use of these tools is to have students design their own animations. This approach has been recommended as an effective learning tool by Stasko [9], and the Sort Animation Builder gives the student a natural, easy-to-learn way to implement such animations. As students debug their algorithm and code, they will find the algorithm/code animation helpful in this process as it provides two of the three visualization approaches explored by Baecker et al[10].

The tools described here are available through a web site developed by the second author. This site contains links to an extensive collection of web-based animations of many different algorithms, including sort algorithms. It also contains a link to the Sort Animator and the Sort Animation Builder. The web site is located at
`www.cs.hope.edu/~alganim/ccaa/ccaa.html`

Acknowledgments

The authors would like to thank James Vanderhyde, who contributed many helpful ideas to this project. This work was partially funded by the National Science Foundation, Grant Number CDA-9423943-03.

References

[1] Baecker, R.M. *Sorting out Sorting,* color/sound film, University of Toronto, 1981. Distributed by Morgan Kaufmann, San Francisco.

[2] Naps, T.L. Algorithm Visualization in Computer Science Laboratories, *Proceedings of the Twenty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, 22, 1 (Feb. 1990), 105-110.

[3] Stasko, J.T. Tango: A framework and system for algorithm animation. *IEEE Computer*, 23, 9 (Sep. 1990), 39-44.

[4] Brown, M.H. Exploring algorithms using BALSA-II. *IEEE Computer*, 21, 5 (May 1988), 14-36.

[5] Naps, T.L. Algorithm visualization served off the World Wide Web: why and how. *Proceedings SIGCSE/SIGCUE '96*, Barcelona, Spain (June 1996), pp. 66-71.

[6] Brown, M.H. and Najork, M.A. Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom. *Digital Systems Research Center Research Report 142*, 1996.

[7] Ungar, D., Lieberman, H. and Fry, C. Debugging and the Experience of Immediacy. *Communications of the ACM* 40, 4 (Apr. 1997), 38-43.

[8] Dershem, H.L., Barth, W.L., Bowsher, C.J., and Brown, D.P. Data Structures with Ada Packages, Laboratories, and Animations. *Proceedings of the First Australasian Conference on Computer Science Education*, Sydney, Australia, (June 1996), pp. 32-38.

[9 Stasko, J.T. Using Student-Build Algorithm Animations as Learning Aids. *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, 29, 1 (Mar. 1997), 25-29.

[10] Baecker, R., DiGiano, C., and Marcus, A. Software Visualization for Debugging, *Communications of the ACM* 40, 4 (Apr. 1997), 44-54.