PERGAMON

Computers & Education 31 (1998) 89–103

COMPUTERS & EDUCATION

A low-tech, hands-on approach to teaching sorting algorithms to working students

J. Geller[a,*], R. Dios[b]

[a]*Department of Computer and Information Sciences, New Jersey Institute of Technology, Newark, NJ 07102, USA*
[b]*Department of Mathematics, New Jersey Institute of Technology, Newark, NJ 07102, USA*

Abstract

This work focuses upon identifying the educational effects of "activity oriented" instructional techniques. We seek to determine which instructional methods produce enhanced learning and comprehension. Specifically we discuss the problem of learning *sorting algorithms*, a major topic in every Computer Science curriculum. We present a "low-tech hands-on" teaching method for sorting algorithms. We stress that there is no need to introduce the World-Wide Web or other high technology tools into this scenario. Primary targets for our teaching approach are part-time students that have little time for homework assignments, because they are supporting families and/or have full time jobs. In this paper we also report the results of a statistical evaluation of our approach. The application of our "hands-on" technique to teaching sorting algorithms produces a dramatic improvement of students' test scores.

1. Introduction

One of the core subjects in introductory Data Structures and Algorithms classes is sorting. The students are exposed to a battery of increasingly complicated sorting techniques. The current state-of-the-art in teaching algorithms in general and sorting algorithms in particular stresses the use of graphical and interactive methods (Maxim & Elenbogen, 1987; Scanlan, 1987; Stone, 1989; Sachdev, Nagpal & Adu, 1990). In the HotJava browser (HOTJAVA, 1995) to the World-Wide Web (WWW) three sorts (Quicksort, Bubble Sort, and Bi-directional Bubble Sort) are actually built in as demonstrations, and students appear to like those very much. Still "chalk and talk" (Transformation of Learning, 1994) is the primary way of communicating this kind of knowledge to students, except in distance education (Adams, Barker, Gal-Ezer,

* Corresponding author. Tel.: (973) 596-3383, e-mail: geller@homer.njit.edu.

0360-1315/98/$19.00
PII: S0360-1315(98)00021-9

Lawhead, Maly et al., 1997). We are still a couple of years away from classrooms where every student can run a Web browser during class to understand a sorting algorithm. Indeed, we are even a couple of years away from every teacher having Web access in the classroom where he is teaching.

Looking at results of Recker and Pirolli (1992) no clear picture emerges whether we even want to offload teaching to the Web. As they report "...higher ability subjects using the hypertext environment improved and made significantly less errors when programming new concepts while lower ability subjects did not improve and made errors. Meanwhile, subjects using the control environment did not show this ability-based difference (p. 382)". In other words, the Web approach to teaching seems to widen the gap between good and bad students. As a consequence of such results we have turned to a decidedly low-tech approach towards teaching sorting algorithms, based on the maxim of "learning by doing".

The *purpose* of this article is to investigate the claim that low-tech approaches to teaching computer science concepts can result in considerable improvements of students' scores. This challenges the current wisdom that high-tech tools such as the World-Wide Web might be necessary to improve computer science education. A *100 level* college course for computer science majors is used as the basis for this investigation. We are specifically interested in applying our techniques to courses in which a majority of the registered students also have substantial work and family care commitments. These students need to make optimal use of class time, since they have very little time for studying and homework assignments.

In Section 2 all the concepts of computer algorithms that are necessary for an understanding of this paper are explained in detail. Section 3 shows in detail how we have implemented Learning by Doing for Sorting Algorithms into a class room setting. In Section 4 we respond to possible criticisms, e.g., that modern Web-based approaches are necessarily better than our approach. Section 5 shows how the experimental data of students was statistically evaluated. Section 6 contains brief conclusions.

2. Sorting algorithms and their complexity

As mentioned before, one of the core subjects in introductory Data Structures and Algorithms (Carrano, 1995) is sorting. The students are exposed to increasingly complicated sorting techniques. In the beginning comes usually Selection Sort, which is easy to understand, but relatively slow. Then they advance to more complicated but faster sorts.

The speed of an algorithm is described by the "big-Oh notation". It is assumed that an algorithm A receives input of the size N. For a sort that would mean that there are N numbers to be sorted. If we then say that algorithm A has the time complexity O(N), that means that the runtime of A is limited by c*N, where c is a constant. The big-Oh notation assumes the "worst possible data" and does not exclude the possibility that the program might run faster on some data. In addition, a finite number of exceptions may be allowed. A simple sort such as Selection Sort has a complexity $O(N^2)$. That means that it takes less than $c*N^2$ run time for all but finitely many N. Selection Sort has a complexity of $O(N^2)$ under all circumstances even if the numbers are already sorted!

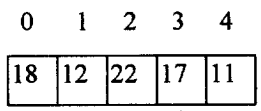

Fig. 1. Initial unsorted array of numbers with position numbers.

Students then advance to better $O(N^2)$ algorithms, such as Bubble Sort and Insertion Sort. These two algorithms have O(N) best case behavior, even though they remain $O(N^2)$ in the worst case. That means that Bubble Sort runs in O(N) time in the special case where the data happens to be already sorted. In the worst case, however, Bubble Sort still needs $O(N^2)$ time. Both Bubble Sort and Insertion Sort are more difficult to understand than Selection Sort.

2.1. Bubble sort

To make this paper more concrete and self contained we will demonstrate the functioning of Bubble Sort with an example. Assume that the following numbers are given:

18 12 22 17 11

These numbers are stored in a way so that they are directly accessible by position numbers. Graphically, we express this as shown in Fig. 1. The number 18 is at position 0, the number 12 is at position 1, etc. In later following figures we will omit the position numbers.

This kind of structure is basic in computer science and is called *array of integers*. What we would like to get as a result of running the Bubble Sort algorithm is shown in Fig. 2.

That is, we want the same array with the numbers sorted in increasing order. The basic idea of Bubble Sort is that we compare two consecutive numbers. If they are already in correct *relative* order, we leave them alone. If the second number is smaller than the first number then we exchange those two numbers. In our case, 18 and 12 are in wrong relative order, so we exchange them. We mark the two numbers that we are comparing by arrows, as shown in Fig. 3.

This "compare and exchange step" is repeated once through the whole array. The numbers 18 and 22 are already in correct order. The numbers 17 and 22 are not in correct order, so we swap them. Finally, 22 and 11 are not in correct order, so we swap them. These steps are all shown in Fig. 4.

At this stage, we are guaranteed that the largest number in the array is at the last position. Furthermore, that means that the largest number is at the *absolutely* correct position and never needs to be moved again. We may therefore ignore it for the rest of this analysis. (For a proof of these facts, the reader is referred to any of a large number of Data Structures and

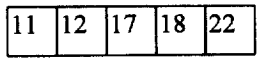

Fig. 2. Desired result: the sorted array.

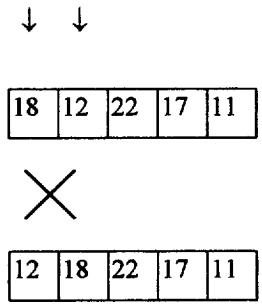

Fig. 3. Initial sorting step: 18 and 12 are "swapped".

Algorithms text books.) Ignoring the last number means that we now repeat the same process on the following subarray (Fig. 5).

There is one more twist to Bubble Sort. If it is possible to traverse through the array once, without ever swapping any two numbers, then the array is obviously already sorted. Checking for this is easy, and that makes Bubble Sort an O(N) sort in the *best* case. It should now also be clear where the name Bubble Sort is coming from. If we draw the array vertically, the large "heavy" numbers are sinking down to the bottom, the "light" small numbers are bubbling up to the top.

Finally, after studying Selection Sort, Bubble Sort and Insertion Sort, students are introduced to Merge Sort and Quicksort which are quite difficult to understand, but which have a very good average complexity of O(N log N). Merge Sort even has a worst case complexity of O(N log N). In the theory of algorithms this is considered to be the best possible complexity for a sorting algorithm based on comparisons.

2.2. Quicksort

Even though Quicksort is algorithmically much more complex than most of the common sorting algorithms, we have found it easy to extend our methodology to it. Quicksort consists

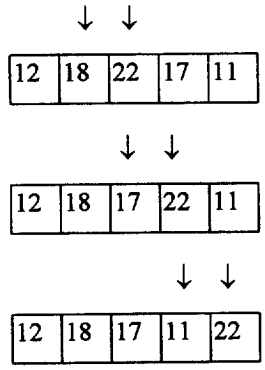

Fig. 4. End of first run through the array.

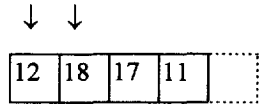

Fig. 5. Beginning of the second run through the (sub)array.

of two parts (Carrano, 1995). One part is Quicksort proper which is very short and generally considered to be simple, as it consists mostly of two recursive calls to Quicksort and one additional call to a function named Partition. All the conceptual difficulties for students are hidden in Partition.

Partition takes an array of numbers and rearranges it such that after processing every number there will be three subarrays of numbers called S1, Pivot, and S2. Pivot contains a single number. S1 consists of all the numbers of the original array that are smaller than Pivot. S2 consists of all the numbers of the original array that are larger than Pivot. These three subarrays are arranged in the order S1–Pivot–S2. Let us assume that the array in Fig. 6 shows the initial data passed to Partition. It is a common assumption to make the first number in the array (here 18) the Pivot (Carrano, 1995).

Fig. 7 shows the results of the partitioning process. S1 consists of the numbers 13 and 12. S2 consists of the numbers 24 and 22. The Pivot, 18, comes after S1 and before S2. The "trick" of Quicksort is that now the Pivot is at the correct position, that is, at the position where it will be eventually in the sorted array (Carrano, 1995). Now S1 and S2 can be sorted by the two recursive calls to Quicksort that were previously mentioned.

It is not our intention to teach Quicksort in this paper, and therefore we will not go through every single step of how Partition transforms Fig. 6 into Fig. 7. However, it was necessary to introduce the basics of Quicksort, so that it becomes clear that our methodology (described below) applies equally to simple sorting algorithms and complex sorting algorithms.

3. Our approach to learning by doing

The claims made in this paper are (1) that learning by doing can be integrated into a normal class room setting of teaching sorting; (2) that learning by doing results in enhanced learning and comprehension; and (3) that high technology tools such as the World-Wide Web are not necessary to improve class room performance. As this last point goes against the currently accepted wisdom, we will devote Section 4 to refute possible criticisms of our approach.

Claim (1) was proven by doing it. The students of a "normal" CIS 114 class at NJIT were asked to bring small index cards to class. Then they had to draw an "array" of empty boxes onto a sheet, and write numbers onto the index cards. The sorting algorithms were simulated

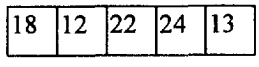

Fig. 6. The initial array for Partition.

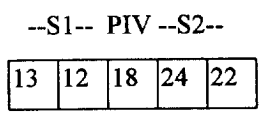

Fig. 7. The result of applying Partintion to the array of Fig. 6.

by placing the unsorted numbers into the array of boxes, and then step by step performing the sorting steps shown on the blackboard. We will now describe in more detail the setup that was used.

3.1. Lecture situation

The teaching of sorting algorithms started immediately after the midterms had been graded. All students were divided into two groups, A and B, based on their midterm grades. The division was performed in such a way that every student belonged to one of the two groups, that the group sizes were identical, and that the sums of midterm points of both groups were nearly identical. (By coincidence, we came up with two groups that indeed had identical point sums.) There were 13 students in each group, and the total number of points on the midterm in each group was 762. The reason for this division was to create an unbiased distribution of the students.

Over the course of two weeks, six sorts, Radix Sort and the five sorts mentioned above, were taught as follows. All students recorded one example of each sort in their notebooks. However, students of group A also hand-simulated Selection Sort, Insertion Sort, and Merge Sort with their paper arrays. This exercise was interleaved with their recording of the sort examples in their notebooks.

Students of group B would simulate Bubble Sort, Quicksort, and Radix Sort with their paper arrays, again interleaved with recording corresponding examples in their notebooks. In this way, both groups of students were exposed to the "lecture-only" learning situation for three sorts, and to the "learning by doing situation" for three other sorts.

3.2. Exam

A week before the final exam, students were informed that sorting would be a major issue on their finals. When the exams were designed, they consisted of nine questions. Six of them were sorting questions, one for each sort. In every one of these questions students were given an array of numbers and had to draw, step by step, how this array would change in the course of the sorting algorithm, similar to Figs. 1–7. The other three questions were from other topics, and they were identical for all students.

Both groups, A and B, were internally subdivided into four subgroups, Aa, Ab, Ac, Ad, and Ba, Bb, Bc, Bd. Exams of Aa were identical to exams of Ba, Ab = Bb, etc. The only distinction was a marking on the corner, stating that this was group Aa, Ab, etc.

The sorting questions of subgroups a, b, c, and d were also identical to each other, however, they were counterbalanced in the order of presentation. In order to eliminate serial effects, the orders of sorts were counter balanced. Students of subgroup "a" would see the three sorts that

they learned by doing first, and the three other sorts afterwards. Students of subgroup b would see the three sorts that they learned the classical way first, and the three other sorts afterwards. Students of group c had the sorts interleaved in one pattern, students of group d had them interleaved in an opposing pattern.

At the exam all students of group A showed up. Unfortunately, two students of group B had dropped the class. The exam was given "closed book". Every student received 9 xeroxed pages, with one question on each page, and appropriate space to fill in the answers on the same page.

As there was no questionnaire attached to the exam, and as variant exams are commonly given to avoid cheating, the students were not informed about the nature of the experiment, except that they would have to sit in a way that no two people of the same group would sit next to each other.

3.3. Evaluation

Exams were graded by the instructor of the class, with no particular order of the exams. First all Selection Sorts were graded. Then all Bubble Sorts were graded, etc. Penalties (in points) and their reasons were recorded and referred to throughout the grading process, independently for each sort.

The exams were graded by a second independent instructor who had taught the class in question several times. This instructor was not given any specific instructions as how to grade them. He was only given vague information about the purpose of the experiment, namely that a new teaching technique was experimentally evaluated. He was asked to be "uncommonly careful and consistent about grading". The results of the grading process are included in Appendix A of this paper.

4. Alternatives to the hands-on approach

4.1. Why we don't need high technology alternatives

In this section, we would like to address possible criticisms of our approach. It is very obvious that we are "swimming against the stream". At the current state of the art one could use the World-Wide Web in at least two different ways to create programs to help students learn sorting algorithms.[1] One possible way would be to graphically simulate the sorting process and let students watch. The problem with this approach is that it is passive. After starting the process, the student can watch, or he can get distracted and lose a step, in the end deriving very little benefit out of the process.

[1] We are quite familiar with Web-based tools, as we are involved in a research project that connects databases to the Web. We are also fully in favor of using the Web for distributing materials to teachers (Tucker, Rada, Roberts & Wegner, 1997) and using the Web in classes where it is relevant to the content, such as multimedia (Heller, Fox, Adams & Vides, 1997).

To overcome this passivity, an interactive extension of the above Web program may be used. Here, the student has to move numbers around with a mouse, and he gets all the benefits of our hands-on approach. This is actually a very good learning method. However, we still prefer the index card approach. We will now explain why, and we hope that the field is "broad enough" to accept our approach, too.

At the current state of networking there are few classrooms in few universities where every student has access to his own computer. This kind of access is required for real hands-on interactive manipulation. That means that the students will have to do this manipulation as part of their homework assignments. But there are two reasons why we would not want students to learn such important concepts in a homework situation, as opposed to a classroom setting.

First of all, we don't want to assign too much homework. Second of all, students are underusing the classroom experience. We will now elaborate on these two points. We don't want to assign too much homework, because the primary targets for our approach are working students which are typically part-time students with full-time jobs. Many of these students also have to support families, both financially and with their time.

Now what do we mean by "the students are underusing the classroom experience?" Most of the students that we know are trying to listen and to concentrate. However, often they fail, which is especially understandable in night classes where they might have already worked a 9 h workday before coming to class. As a result, many students go home, without acquiring the important concepts in class, so they have to do it at home. But at home they have no time, might be distracted by a crying baby, etc., which brings us back to our first point. Thus, we would like the students to make maximum use of the classroom experience.

In our experiments we have found that manipulating index cards in class is an excellent way of overcoming some of the problems mentioned above. The physical involvement of having to manipulate the index cards helps them to focus their attention. It literally makes it harder for the students to "drift away with their thoughts". If the students manage to acquire the important concepts in class then they don't need to start a Web browser and work on those concepts "on their own time". Therefore, until there is a computer on every desk in every class room, we suggest using index card manipulation to help students focus on the learning process.

4.2. Why the hands-on approach is better than a code walk-through

After we have argued against the high-tech solution to teaching sorting algorithms, one wonders whether our hands-on approach is really necessary. Maybe a simple code walk-through would be sufficient to teach sorting algorithms? In order to reject this notion, Section 5 shows experimental results that compare the normal teaching method with our hands-on teaching method. The hands-on method was found to be superior.

But why is the hands-on approach better than a simple code walk-through? The answer echoes some of the arguments of the previous subsection. In a normal lecture situation the instructor does the walk-through, and it is up to the attention level of the student whether he follows or not. In the hands-on approach a student is forced to maintain a minimum amount of attention, otherwise he cannot advance from array configuration to array configuration.

4.3. Other issues

There are two more issues that we would like to address:

1. Does this approach also work for complex sorting algorithms such as Quicksort? The answer to this question is that in our experience it does work. Even though we did not describe all details of Quicksort, it should be clear that nothing more complicated than two indices (arrows) and the same exchange (swap) operation that we previously used in Bubble Sort are necessary. Students can simulate the two indices by pointing with two pencils to the appropriate index cards. Swaps are performed by exchanging index cards. In summary, our methodology is (was!) easily applied to Quicksort and other complex sorts.
2. Does this approach slow down the teacher and cut down the amount of material that can be covered in class? The answer to this question is an unequivocal yes. However, whether this is unacceptable is a question of teaching philosophy. We feel that the classroom should be used to truly acquire concepts. We are willing to accept a small loss in quantity if we can improve the quality of the classroom learning process. Professors that disagree with this philosophy will not find our method useful.

5. Hypotheses and statistical evaluation

Our basic hypothesis is that students from group A would score better on Selection Sort, Insertion Sort and Merge Sort than students from group B. On the other hand, student from group B would score better on the three other sorts which they had experienced in a hands-on fashion.

This claim was evaluated by tabulating the individual points of each sort for each group three times: for grader I, for grader II, and for an average computed from grader I and grader II. In addition, we used the points given on the non-sorting exam questions (7, 8, 9) for the purpose of normalizing the data in an additional evaluation phase. The basic assumption of this phase is that the total score on questions 7–9 should be the same in both groups, as both groups had identical midterm grades. Assume, however, that for unknowable reasons students of group A prepared better than students of group B for the final. We then determined a scale factor that would reduce the total score of group A on questions 7–9 to the same value as the total score of group B on questions 7–9. By applying the same scale factor to the points on questions 1–6 of group A, we could exercise some control over such a shift in preparation. With this normalized data we now repeated the three evaluation steps described above, i.e., for each teacher and for an average of both teachers.

5.1. Statistical discussion of the students' grade scores

A Two-Way Analysis of Variance (ANOVA) was performed on the two separate data sets provided by the two graders. In both cases, there was a strong significant difference in grades for the two distinct modes of instruction (see Tables 1 and 2: $P = 0.00082$, $P = 0.00083$).

Table 1
Analysis of variance table for Grader 1

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	*F* Ratio	Prob(*F*) (*P*-value)
Replications	10	1721	172	4.8	0.000056
Factor A	1	437	437	12.1	0.00082
Factor B	8	3540	442	12.3	0.00005
Interaction AB	8	127	16	0.44	0.895
Error	170	6129	36		
Total	197	12,000			

In addition, there was also a very strong significant difference on exam performance by the students in both groups from one sorting procedure to the next (see Tables 1 and 2: $P = 0.00005$, $P = 0.000061$). The level of interaction was very high for both experimental variables: "mode of instruction" versus "sorting procedure". The first grader had the highest level of interaction for the variables (see Table 1: $P = 0.895$) indicating that his methodology for assigning grades was highly dependent upon the sorting procedure being tested. The data from the second grader also indicated significant interaction (see Table 2: $P = 0.484$). The data on the students' grades further indicate that the first grader was somewhat more versatile in assigning grades, hence showing diversity in his evaluation of the variety of solution techniques/approaches which are possible within the two modes of instruction being tested for statistical differences. Otherwise, the graders produced highly similar results. The values of the correlation coefficient linking sorting procedure to mode of instruction are 54.3% and 39.4%, respectively. There was also a strong significant difference between one student's performance to the next within their own class. This is true for both graders. The following tables provide the statistical information for the ANOVA, as well as descriptive estimates of the central tendency and dispersion parameters of both data sets. We see as an example that the average grade over all tests is much higher for the first mode of instruction ("hands-on") than for mode 2 (see Tables 3 and 4). Additionally, we can see by perusing the mean values of the test scores that the most difficult problems to master are those numbered 7, 9 and 8 (see Tables 5 and 6). We also observe that there is far more variation between grades for the "easier" sorting procedures which possess the largest mean test scores.

Table 2
Analysis of variance table for Grader 2

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	*F* Ratio	Prob(*F*) (*P*-value)
Replications	10	2126	213	3.3	0.00063
Factor A	1	768	768	12.1	0.00083
Factor B	8	2642	331	5.2	0.000061
Interaction AB	8	479	60	0.9	0.484
Error	170	11,000	64		
Total	197	17,000			

Table 3
Summary of the population statistics for Factor A (mode of instruction)

Factor A	Count	Mean	Standard Deviation	Minimum	Maximum
Mode 1	99	15.848	1.927	0	25
Mode 2	99	12.879	1.708	0	25

Table 4
Grader 2. Summary of population statistics for Factor A (mode of instruction)

Factor A	Count	Mean	Standard Deviation	Minimum	Maximum
Mode 1	99	16.263	0.881	0	25
Mode 2	99	12.323	0.938	0	25

Table 5
Grader 1. Summary of the population statistics for Factor B (sorting procedure)

Factor B	Count	Mean	Standard Deviation	Minimum	Maximum
1	22	19.182	4.692	5	25
2	22	20.545	4.883	10	25
3	22	13.773	3.581	5	25
4	22	18.500	4.501	7	25
5	22	12.409	3.328	0	25
6	22	16.136	4.657	0	25
7	22	7.591	2.458	0	15
8	22	11.000	2.952	0	20
9	22	10.136	2.457	2	15

Table 6
Grader 2. Summary of population statistics for Factor B (sorting procedure

Factor B	Count	Mean	Standard Deviation	Minimum	Maximum
1	22	14.955	2.624	0	25
2	22	19.773	1.903	0	25
3	22	16.591	1.872	0	25
4	22	18.227	1.685	0	25
5	22	15.000	1.918	0	25
6	22	15.227	2.579	0	25
7	22	8.500	1.237	0	15
8	22	10.455	1.182	0	20
9	22	9.909	0.775	4	15

5.2. Final comments

During the experimental situation and evaluation of the exams it became clear that there is at least one problem with our hypothesis. We are assuming that students either learn the mechanics of a sort, or they fail to learn it. In reality, even though we achieved quite good results in our statistical evaluation, students might fail because they attach a wrong label to a mechanism.

For instance, in some cases students produced a perfect example of a Bubble Sort, but they did it in the question that required a Selection Sort. Those students might or might not have acquired the mechanics of Selection Sort, but they were missing the correct link between the name and the mechanism. Our "learning by doing" approach does not help in creating this link. Given this problem we were actually surprised by the excellent results of our statistical analysis. One aspect of future work in this area is to try to separate out failure due to mislabeling the mechanism and failure due to misunderstanding the mechanism.

6. Conclusions

In this paper we have argued that not all solutions to improved learning have to be high-tech. We have shown an approach to the teaching of sorting algorithms that is hands-on but requires only normal writing materials. Our approach is primarily targeted at students that have to minimize the time they can spend on homework assignments because they have full time jobs and often also family care obligations.

The statistical analysis of the data verifies our contention that the mode of instruction utilized to teach a variety of sorting procedures can have a vastly different and powerful effect upon the level of learning for a given student. The six different sorting procedures are clearly very different both systematically and from a cognitive viewpoint. The "hands-on" method of instruction produces higher test scores indicating improved comprehension and a greater depth of understanding. This is true for all sorting methods.

Even though we are pleased with the results described in this paper, it is obvious to us that it defines the beginning of a research program, and not the end. We hope to give impetus to other studies that perform a rigorous three-way comparison between "talk and chalk" teaching, Web-based teaching, and hands-on teaching as defined in this paper. Great care has to be taken to eliminate the personal biases of the instructors from such studies. Statistical evaluations of exam data need to be performed, to decide whether differences between grade results are statistically significant, or not. We hope that our study leads the way in this respect. Comparisons between night-class sections and sections of full-time students with the same instructor need to be designed in a way that student performance is measured, eliminating the influence of the 9 h workday the instructor has "survived" when he starts teaching the night class.

Down the line, it would be desirable for instructors to record their grades for every exam problem in a computerized database. Error codes for commonly occurring student mistakes should be assigned to wrong answers and stored in the same database. This would make it possible to perform longitudinal studies and to reevaluate previous results with other statistical tools. We are looking forward to responses from other researchers in the field.

Appendix A. Test Data

Key:

S	=	Selection Sort
B	=	Bubble Sort
I	=	Insertion Sort
M	=	Merge Sort
Q	=	Quick Sort
R	=	Radix Sort
#	=	Student Number

Grader 1: *Group A*:

Table 7

#	S	B	I	M	Q	R	7	8	9
2	25	25	10	23	17	20	15	15	15
3	25	25	25	23	25	25	15	15	15
4	25	25	10	25	13	25	2	10	6
5	15	25	10	10	5	5	4	10	11
6	25	25	25	10	21	25	6	15	12
7	25	25	20	13	13	25	15	5	12
10	25	17	25	25	25	25	10	10	12
13	15	25	10	10	5	25	0	15	9
16	10	10	10	23	13	0	0	10	11
18	25	10	10	19	13	25	11	15	9
19	25	25	10	25	13	5	13	15	15
21	13	17	10	16	25	25	5	10	12
22	25	25	20	25	13	25	9	5	12

Group B:

Table 8

#	S	B	I	M	Q	R	7	8	9
1	17	25	10	21	13	20	0	5	9
8	15	25	10	19	13	5	2	12	12
9	13	10	10	7	5	5	10	0	9
11	25	10	10	23	13	25	0	15	6
12	7	17	10	25	13	25	8	20	6
14	25	25	20	23	5	5	15	15	12
15	25	25	25	25	17	25	15	15	15
17	25	25	18	10	13	25	15	15	9
20	10	18	10	25	13	5	7	0	11
23	15	10	10	10	5	5	4	10	5
24	5	25	5	13	0	5	0	0	2

Grader 2:

Please note that two problems were accidentally not graded. These are marked with "?".

Group A:

Table 9

#	S	B	I	M	Q	R	7	8	9
2	25	25	10	25	15	25	15	15	15
3	25	25	25	23	25	25	15	15	15
4	25	20	15	25	20	25	3	10	4
5	0	25	0	0	0	0	4	10	7
6	25	25	25	22	25	25	11	15	12
7	25	20	25	10	20	25	15	5	12
10	25	25	25	23	25	25	15	5	12
13	0	20	20	10	5	?	2	15	8
16	0	0	15	25	0	0	4	10	11
18	25	25	10	23	20	25	12	15	9
19	25	25	15	10	20	5	14	15	15
21	0	25	10	?	25	25	2	10	12
22	25	25	25	25	20	25	2	0	12

Group B:

Table 10

#	S	B	I	M	Q	R	7	8	9
1	5	15	5	20	15	25	1	5	12
8	0	25	20	20	20	5	1	10	10
9	0	20	0	5	0	0	10	5	9
11	25	0	10	10	10	25	2	15	6
12	0	25	15	25	20	25	7	20	8
14	24	0	20	25	0	0	15	15	12
15	25	25	25	25	20	25	15	15	15
17	25	25	25	10	15	25	15	15	6
20	0	10	5	25	20	0	6	5	7
23	0	25	25	10	20	0	5	10	5
24	0	25	25	15	0	0	0	0	4

References

Adams, E. S., Barker, K., Gal-Ezer, J., Lawhead, P., Maly, K., Miller, J. E., & Thomas, P. (1997). Distance education: promise and reality. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (pp. 369–370). San Jose, CA.

Carrano, F. M. (1995). *Data Abstraction and Problem Solving with C + + : Walls and Mirrors.* Redwood City, CA: The Benjamin/ Cummings Publishing Company.

Heller, R., Fox, E., Adams, W. J., & Vides, G. M. (1997). Defining multimedia courses within a computer science education. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (pp. 379–380). San Jose, CA.

The HOTJAVA browser: a white paper. [http://java.sun.com/java.sun.com/1.0alpha3/doc/overview/hotjava/index.html], 1995.

Maxim, B. R., & Elenbogen, B. S. (1987). Teaching programming algorithms aided by computer graphics. *ACM SIGCSE Bulletin, 19*(4), 297–301.

Recker, M. M., & Pirolli, P. (1992). Student strategies for learning programming from a computational environment. In *Intelligent Tutoring Systems: Second International Conference* (pp. 382–394). Berlin: Springer Verlag.

Sachdev, M. S., Nagpal, M., & Adu, T. (1990). Interactive software for evaluating and teaching digital relaying algorithms. *IEEE Transactions on Power Systems, 5*, 346–352.

Scanlan, D. (1987). Data-structures students may prefer to learn algorithms using graphical methods. *ACM SIGCSE Bulletin, 19*, 302–307.

Stone, D. C. (1989). Using cumulative graphics traces in the visualization of sorting algorithms. *ACM SIGCSE Bulletin, 21*(4), 37–42.

A transformation of learning: use of the NII for education and lifelong learning. [http://iitfcat.nist.gov:94/doc/Education.html], 1994.

Tucker, A. B., Rada, R., Roberts, E., & Wegner, P. (1997). Strategic directions in computer science education. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (pp. 371–372). San Jose, CA.