Student Monks – Teaching Recursion in an IS or CS Programming Course Using the Towers of Hanoi

Alan C. Benander
Barbara A. Benander
Computer and Information Science Department
Cleveland State University
Cleveland, Ohio, 44023, USA
abenande@cis.csuohio.edu

ABSTRACT

Educators have been using the Towers of Hanoi problem for many years as an example of a problem that has a very elegant recursive solution. However, the elegance and conciseness of this solution can make it difficult for students to understand the amount of computer time required in the execution of this solution. And, like many recursive computer programs, students often find it difficult to follow a trace of the solution. Research in computer education has shown that active learning exercises achieve positive educational results. In line with this research, an active learning exercise was employed in the classroom to assist students in gaining a better understanding of the recursive solution to the Towers of Hanoi problem. This demonstration can be used in an introductory IS or CS programming class, independent of the language used. The demonstration involves using student volunteers, who, in the demonstration, are referred to as "monks", a reference to the original problem that had monks moving the golden rings in the Towers of Hanoi. An anonymous student survey revealed that students felt strongly that the demonstration helped them better understand recursion, and that the demonstration was a good use of class time. In addition, an analysis of a small sample of students' computer programs following the demonstration, suggests that there may be pedagogical benefits to use of the student monk demonstration.

Keywords: Teaching Recursion, Active Learning, Computer Education, Towers of Hanoi

1. INTRODUCTION

1.1 Recursion and Related Research

Recursion is a programming technique that involves a procedure/method/function calling itself either directly or indirectly through another procedure/method/function (Deitel and Deitel, 2005). The use of recursion has long been viewed as being fundamental to computer programming (McCracken, 1987). Today, recursion is typically found in most IS and CS curricula, in the latter part of the first programming course. Historically, programming instructors have had difficulty in explaining recursion to their students (Levy and Lapidot, 2000; Wiedenbeck, 1988), and researchers have attested to the difficulty that most students have in fully understanding recursion (Anderson, 1976; Henderson and Romero, 1989; Levy and Lapidot, 2000; Pirolli and Anderson, 1985; Turbak, Royden, Stephan, and Herbst, 1999; Wiedenbeck, 1988). It has been contended that the difficulty in understanding recursion stems from unfamiliarity with recursive activities, and that the mind, while able to deal with iteration, has difficulty in dealing with recursion (Anderson, Pirolli, and Farrell, 1988). It has also been proposed that recursion is difficult because it lacks everyday analogues (Pirolli and Anderson, 1985).

Research (Gotschi, Sanders, and Galpin, 2003) has also found that that many first-year students have difficulty tracing recursive programs. This finding is consistent with results from a survey given in Section 4 in which students indicated that recursion is a difficult topic. A trace of a recursive program is defined as the "representation of the flow of control and the calculation of the solution of a recursive program" (Gotschi et al., 2003, pp.346). Ability to correctly trace a recursive program is essential in order to determine its computational time complexity. The computational time complexity of a program is the time that a program needs to execute, as a function of the input size. For example, the standard bubble sort program's time complexity is a quadratic function of n, where n is the number of elements to sort. Even in an age of increasing clock speeds and bus widths, time complexity is a very important consideration when assessing program performance. For example, suppose a program has time complexity $f(n) = n$, a linear function of the size, n, of its input. If this program were to run on a processor that is 10 times as fast, it could process a problem with 10 times greater size. In contrast to this, suppose a program has time complexity of $f(n) = n^2$, a quadratic function of the size, n, of its input. If this program were to run on a processor that is 10 times as fast, it could process a problem with only 3.16 times greater size. Aho, Hopcroft, and Ullman (1974) give an analysis showing the importance of the time complexity of an algorithm.

While recursive programming is often difficult for students to master, it can often be an elegant approach to solving certain problems that present themselves in database and data warehouse programming applications. In these types of applications, data and their indexes are stored in tree-like structures that can often best be traversed using recursive techniques. Indeed, recursion is quite useful in computer programs that use recursively defined data structures, such as linked lists and trees. The operations of traversal, retrieval, and searching through linked lists and trees, can be defined recursively, and thus lend themselves to a natural recursive programming implementation. A (non-empty) linked list, for example, can be defined (informally) recursively; i.e., in terms of itself, in the following way. A non-empty linked list of size n consists of a single node if n = 1; and for n > 1, consists of a linked list of size n - 1 with an additional node attached.

Figure 1 illustrates a linked list. Note that a linked list will have a “pointer” (denoted by “L” in Fig. 1) to the first item, which allows for traversal of the linked list.

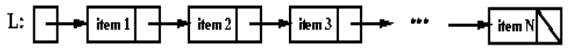

Figure 1. Linked List

Typically, recursion is introduced to IS and CS students only after they have been taught to use iterative constructs, such as for-loops and while-loops. While any recursive code can be written iteratively, it is often easier to write recursive solutions for certain programming problems. Recursive programming, and the differences between recursive and iterative programming, have been studied by various researchers (Benander, A., Benander, B., and Sang, 2000; Benander, A., Benander, B., and Pu, 1996; Danvy, 2002; Kessler and Anderson, 1986; Sinha and Vessey, 1992). A variety of studies have been performed comparing the use of recursion with iteration in computer programming. For example, Benander et al. (1996) reported on an empirical study of comprehension of iterative and recursive code which showed that beginning programmers were able to correctly comprehend certain recursive code involving linked lists faster than the iterative version. Benander et al. (2000) also conducted an empirical study showing that for certain small segments of code, subjects in an experiment were able to more successfully locate a bug in a recursive version of that code than in an iterative version. In another study, Wiedenbeck (1989) concluded in a study of novice Pascal programmers, that comprehension of an iterative program was not affected by prior experience with the recursive version of the same program, and that comprehension of the recursive version was only weakly affected by prior experience with the iterative version.

Over the years, there has also been much research on various pedagogical approaches to the teaching of recursion (Bruce, Danyluk, and Murtagh, 2005; Ford, 1984; Henderson and Romero, 1989; Kruse, 1982; Tung, Chang, Wong, and Jehng, 2001; Turbak et al., 1999; Wiedenbeck, 1989; Wu, Dale, and Bethel, 1998). For example, Bruce et al. (2005) recommend teaching recursion before teaching arrays, while Turbak et al. (1999) recommend the uncommon approach of teaching recursion before teaching iterative techniques. Ford (1984) outlines the use of the principle of mathematical induction in arguing the correctness of a recursive algorithm. Kruse (1982) suggests the use of tree diagrams to illustrate recursion and explain its implementation. Wu et al. (1998) concluded that concrete conceptual models were better than abstract conceptual models in teaching recursion to novice programmers. Finally, the choice of a certain functional programming language, Standard ML, has been recommended as a tool for teaching recursion by Henderson and Romero (1989). A summary of this past research on pedagogical approaches to teaching recursion is given below in Table 1.

Recommendation for Teaching Recursion	**Source of Recommendation**
Teach recursion before arrays	Bruce, Danyluk and Murtagh, 2005
Use visual representations	Tung, Chang, Wong, and Jehng, 2001
Teach recursion before iteration	Turbak, Royden, Stephan and Herbst, 1999
Use concrete conceptual models	Wu, Dale, and Bethel, 1998
Use Standard ML programming language	Henderson and Romero, 1989
Learning from examples	Wiedenbeck, 1989
Use math induction	Ford, 1984
Use tree diagrams	Kruse, 1982

Table 1. Summary of Past Research on Pedagogical Approaches to Teaching Recursion

However, none of the approaches found in the research literature, or in computer programming textbooks, have suggested the use of human subjects in a demonstration of recursion. Everyone can relate to message passing among humans, and active learning exercises that augment passive lectures have been shown to have positive educational results (Cassel, 2002; Depradine and Gay, 2004; Massey, Brown, and Johnston, 2005; McConnell, 1996; Umble, M. and Umble, E., 2004; Walker, 2004). A summary of this research is found in Table 2.

Motivated by this research, the authors have used an in-class demonstration for teaching recursion that uses student volunteers to illustrate the recursive solution to a classical computer programming problem involving the Towers of Hanoi puzzle. This classroom demonstration is being proposed as an aid to teaching recursion in an introductory computer programming class for IS or CS students, independent of the programming language being used.

1.2 The Towers of Hanoi Puzzle

The Towers of Hanoi is a classical example of a problem that has a very difficult iterative solution, yet a relatively simple recursive one. The Towers of Hanoi problem has been

Active Learning Examples (E.g.) & Results (R)	**Source**
E.g.: Use of crossword puzzle and Jeopardy game **R:** Student feedback very positive	Massey, Brown and Johnston, 2005
E.g.: Playing cards in decision tree demo **R:** Positive student results	Umble, M. and Umble, E. 2004
E.g.: Active-learning computer lab sessions **R:** Positive student feedback, success on exams	Walker, 2004
E.g.: Interactive IDE for writing programs **R:** Fewer coding mistakes on tests	Depradine and Gay, 2003
E.g.: Student role playing in network demo **R:** Aided understanding of routing algorithm	Cassell, 2002
E.g.: Props/experiments/group exercises **R:** Better exam performance	McConnell, 1996

Table 2. Summary of Active Learning Exercises That Augment Passive Lectures

studied extensively by different educators (Anderson and Douglas, 2001; Birtwistle, 1985; Mayer and Perkins, 1984; Maziar, 1985; Noyes and Garland, 2003; Sapir, 2004). The actual Towers of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. The apocryphal story of the Towers of Hanoi is a narrative involving a task givento monks and the eventual end of our planet. According to a discussion of the history of the Towers of Hanoi, found at http://hanoitower.mkolar.org/HThistory.html, the Towers of Hanoi puzzle is probably not of ancient Indian origin, as many believe, but instead was probably invented "from scratch" by Lucas. The website also contends that there is no written record of the puzzle prior to 1883. The task given to the monks was to move 64 golden rings of varying diameter from one peg to another, following certain rules. When the monks finish the task, the world will end, according to the story.

Initially, all 64 rings are placed on one of 3 pegs, referred to as the "source peg", in order of decreasing diameter from bottom to top. All 64 rings are to be moved to one of the other two pegs, designated as the "destination peg" (Figure 2). The other peg can be used as a "temporary peg" in moving the rings. The rules for moving the rings are: i) only one ring may be moved at a time, and ii) a larger ring may not be moved on top of a smaller ring. With these rules, if the monks move 1 ring per second, it is estimated that our solar system will have been out of existence before the task is finished. Assuming one move per second, and no wrong moves, it would take approximately 590 billion years to complete.

The short, elegant, recursive solution in pseudocode is given in Figure 3. The 3 steps in this pseudocode solution to the Towers of Hanoi can be explained in non-programming terms as: 1) Move N-1 rings from the "Source" peg to the "Temp" peg; 2) Move one ring (the last and largest one) from the "Source" peg to the "Dest" peg; 3) Move the N-1 rings from the "Temp" peg to the "Dest" peg.

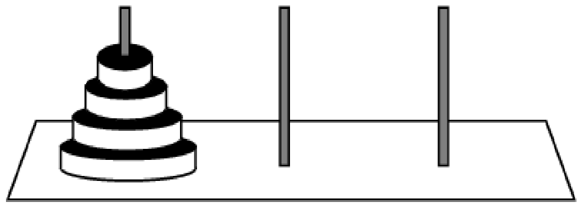

Figure 2. The Towers of Hanoi with 4 rings

As in the development of any recursive programming solution, it should be emphasized to students that: i) their solution should "work" for the "base case"; i.e., for Towers of Hanoi, the case where there is 1 ring (N = 1); ii) each recursive call should reduce the size of the problem; and iii) assuming that the solution "works" for the N - 1 case (N – 1 rings in the Towers problem), then it works for the Nth case (N rings in the Towers problem). This last condition (condition iii) for a recursive solution often requires a "leap of faith" by the students since it is often difficult for students to accept this "inductive condition". These three conditions form the basis of a "three-question method" (Dale, 2003) that programmers should use when attempting to verify the correctness of a recursive solution.

```
Procedure Towers (N, Source, Temp, Dest) {
 if N <= 0 exit;
 else{
        Towers (N-1, Source, Dest, Temp);
    Move ring from Source to Dest;
        Towers(N-1, Temp, Source, Dest);
  }
}
```

Figure 3. Recursive Solution to the Towers of Hanoi

A Java applet illustrating the Towers of Hanoi puzzle can be found at www.cut-the-not.org/recurrence/hanoi.shtml, and can be useful for both teachers and students as an introduction to the solution of the puzzle. The applet shows rings being moved from one peg to another, but does not illustrate the many recursive calls being executed in the solution, nor does it lend itself to an easy trace of the solution. The next two sections discuss an active learning approach to help students in an introductory IS or CS programming course trace the recursive solution to the Towers of Hanoi, and to also help them understand its time complexity.

2. PRE-DEMONSTRATION STUDENT MONK TRAINING

One class period before the actual in-class demonstration, the Towers of Hanoi puzzle can be briefly introduced to all the students in the class. During that in-class introduction, the instructor asks for volunteers to help demonstrate the puzzle at the next classroom meeting. Experience has shown students to be eager to volunteer as "student monks". Also, those students not chosen as volunteers showed no outward signs of disappointment. Since the student monks are to be trained by the instructor outside of class, it may not be feasible to have more than 5 "student monk" volunteers. To help students better understand the recursive solution found in Figure 3, a demonstration prop is used in a training session. This same prop will be used in the actual in-class demonstration. The prop consists of several wooden rings (painted gold for an added authentic touch) and 3 pegs, such as those pictured in Figure 2.

The actual out-of-class training of the student monks should take no more than 30 minutes. The training session begins by labeling the pegs, A, B, and C, left to right. During the actual in-class demonstration, these pegs will be labeled in the same way. The 1-ring case is easily demonstrated by moving the ring from peg A to peg C. The 1-ring student monk is trained to move a ring from peg A to peg C whenever he/she is called with Towers(1, A, B, C). It must be made clear to the 1-ring monk that A, B, and C are actual arguments in the call, and that they must be substituted for the parameters Source, Temp, and Dest, respectively, in the code of the Towers procedure. After moving the ring, the 1-ring monk is trained to tell the student who called him/her, that he/she is finished.

The 2-ring student monk, when called with Towers (2, A, B, C), is taught to do the following: call the 1-ring monk with Towers (1, A, C, B); wait for the 1-ring monk to tell him/her that he/she is finished; when told by the 1-ring monk that he/she is finished, move a ring from peg A to peg C; call the one-ring monk with Towers (1, B, A, C); wait for the 1-ring monk to tell him/her that he/she is finished; when told by the 1-ring monk that he/she is finished, tell the student who called him/her (i.e., the 3-ring monk) that he/she is finished.

Similarly, 3-ring, 4-ring, and 5-ring student monks can be trained, depending on the number of rings to be used in the classroom demonstration. The task that requires the most emphasis in the student training session is the proper substitution of parameters in the Towers procedure itself with the arguments in the call to the Towers procedure. From experience, this is one of the biggest pitfalls for students, and failure to perform this task properly can nullify the benefits of the demonstration. Also, based on experience, it is best to train the student monks outside of class. In-class training of volunteer student monks can be tedious and time consuming, and detract from the lesson at hand. Indeed, when one of the authors first used this demonstration, training was done in an impromptu fashion, in class, without proper understanding by the student monks. The result was comedic, but not at all effective as a learning exercise.

Also, in the first few minutes of the training session held outside of the class, the "dramatic" details (e.g. stepping forward, moving back, "shouting" out the call) of the demonstration need not be stressed. The basic activity of each student monk should be taught first. Detailed training instructions, found in Appendix A ("Training the 4 Student Monks"), should be given to the student monks after the training session in order to reinforce the roles of the different monks. These written instructions in Appendix A can be used by the student monks to study, if they wish, prior to the actual demonstration. Finally, it is suggested that the student monks meet with the instructor 10 or 15 minutes before the actual in-class demonstration, to do a quick rehearsal of the demonstration.

3. THE IN-CLASS STUDENT MONK DEMONSTRATION

For the actual classroom demonstration, the use of 4 rings is suggested. The solution to the 4-ring puzzle is non-trivial, involving 15 moves of the rings on the prop. This is enough moves to sufficiently demonstrate the complexity of the recursive solution. Yet, if the student monks have been properly trained, the demonstration using 4 rings should not take an excessive amount of classroom time, normally less than 15 minutes. The instructor choosing to employ this method of demonstration should note that use of each additional student monk in this demonstration requires twice as many moves of the rings. Also, of course, the use of more rings and more student monks increases the chances of a student monk failing to properly execute the task. Finally, the use of more than 4 rings (and 4 student monks) probably does not provide significant additional educational benefits.

The 4-ring student monk demonstration begins with placement of 4 rings on the leftmost peg of the prop. The 4 trained student monks line up next to each other in order, with the 1-ring monk next to the 2-ring monk, etc., in the front of the classroom, each with a sheet of paper reminding them of their tasks to perform when called. Also, on the sheet of paper, the student mark should keep track of the number of rings he/she moved. The prop with the rings and the pegs is placed on the desk in front of the room. The instructor labels the pegs A, B, C, left to right, and calls the 4-ring student monk with Towers (4, A, B, C). The 4-ring monk then steps forward and makes the call audibly, "Towers (3, A, C, B)". The 4-ring monk then steps back and waits for the 3-ring monk to finish. The demonstration is more effective when each student monk steps forward when making a call, and steps back in line when waiting.

The rest of the students can see how relatively long the 4-ring monk waits before being told by the 3-ring monk that he/she is finished. After the 3-ring monk is finished from the initial call made by the 4-ring monk, the 4-ring monk moves a single ring, and makes another call to the 3-ring monk. It becomes clear to the students that, at that time, the problem is only half-solved. It is also clear to the students in the class that the 1-ring monk moves twice as many rings as the 2-ring monk, the 2-ring monk moves twice as many rings as the 3-ring monk, and that the 4-ring monk moves actually only 1 ring. In practice, the demonstration with the 4 student monks is more effective if preceded by a 3-student monk demonstration. When the final ring is moved, each student monk marks on the blackboard the number of rings that he/she moved, and the numbers for each monk are summed, producing a total number of moves. Students can be asked to "guess" the formula for the number of moves needed to solve the N ring problem. This can be followed by a formal proof using mathematical induction or recurrence relations, if appropriate for the background and type of class being taught. The www.cut-the-knot.org/recurrence/hanoi.shtml, website has a formal proof that the number of moves needed to solve the N-ring puzzle is 2^N-1 for $N \geq 1$. Appendix B

("Walkthrough of 4-Ring Demonstration") contains a complete walkthrough of a 4-ring monk demonstration.

4. STUDENT SURVEY AND PROGRAMMING ASSIGNMENT RESULTS

4.1 The Survey

In an introductory, undergraduate computer and information science Java programming course, the Towers of Hanoi student monk demonstration was given approximately 2 weeks before the final exam. The instructor of this class was one of the authors. Shortly after final grades were submitted, an anonymous survey was given to the students to obtain their opinions regarding the benefits of the demonstration. The survey and directions for completing it were sent via email to the students by an assistant of the instructor. The survey was conducted after completion of the course to help eliminate bias in answering the questions. For this same reason, the survey was sent by the assistant, and not by the instructor, using the assistant's email account. In the emails to the students, the assistant assured the students of anonymity, which was preserved by the assistant who gave responses without names or email addresses to the instructor of the course.

A copy of the survey, with the directions that were given to the students, is given in Appendix C. The survey contained 6 questions. On the first 4 questions, students were asked their opinions related to their classroom experience with the demonstration. The 5th question asked their general opinion of using active participation in the classroom to demonstrate programming concepts. The 6th question asked them about their perception of the difficulty of learning recursion. Eleven of the 12 students in the class completed the survey. The one student who did not complete the survey replied to the assistant that he/she was not in attendance on the day of the Towers of Hanoi demonstration.

Survey Questions 1 = Strongly Agree … 5 = Strongly Disagree	**Mean**	**Std. Dev.**
Question 1. "The Towers of Hanoi Monks demonstration helped me to understand the overhead involved in using recursive code (e.g., the many recursive calls that are actually made in a recursive solution)."	2.00	0.63
Question 2. "The Towers of Hanoi Monks demonstration was more helpful to me in understanding the solution to the Towers of Hanoi problem than a classroom lecture alone would have been."	1.64	0.67
Question 3. "I paid more attention to the Towers of Hanoi Monks demonstration than I would have paid to a simple class lecture from the instructor explaining the Towers of Hanoi."	2.18	0.75
Question 4. "The Towers of Hanoi Monks demonstration was a good use of class time."	1.64	0.81

Table 3. Student Opinion of the Student Monk Demonstration (n = 11)

Mean responses and standard deviations to the first 4 survey questions are given in Table 3. These responses certainly indicate an overall perceived effectiveness of the student monk demonstration in terms of helping the students learn of the overhead involved in recursion, and about the solution to the Towers of Hanoi problem (questions 1 and 2). The survey also indicates that the students had an overall positive experience with the demonstration (questions 3 and 4).

Mean responses and standard deviations for Questions 5 and 6 are given in Tables 4 and 5, respectively. The strongest opinion from the students in the survey came from question 5, where there was strong agreement with the statement that "involving of students in active participation in the classroom helps them to better understand programming concepts." Indeed, 7 of 11 respondents indicated that they "strongly agreed" with that statement.

Survey Question 1 = Strongly Agree …. 5 = Strongly Disagree	**Mean**	**Std. Dev.**
Question 5. "Involving students in active participation in the classroom helps them to better understand programming concepts."	1.45	0.69

Table 4. Student Opinion of Usefulness of Active Participation (n = 11)

The responses to question 6 **(**Table 5 below) support other research (Anderson, 1976; Henderson and Romero, 1989; Levy and Lapidot, 2000; Pirolli and Anderson, 1985; Turbak et al., 1999; Wiedenbeck, 1988) indicating the difficulty that students have in learning recursion. The students in this survey reported that they did indeed find the topic of recursion to be difficult. Not a single respondent indicated that recursion was an easy or very easy topic

Survey Question 1 = Very Difficult …. 5 = Very Easy	**Mea**	**Std Dev**
Question 6. "How difficult, in general, did you find the topic of recursion?"	2.27	0.65

Table 5. Student Opinion of Difficulty of Recursion (n = 11)

It is also notable that not a single student who participated in the survey strongly disagreed (response = 5) or disagreed (response = 4) with any of the statements 1 through 5 in the survey. Also, among the 55 total responses to these 5 questions, only 9 were neutral (neither agreeing nor disagreeing). The other 46 responses were in agreement or strong agreement with the statements. This strongly suggests an overall positive experience with the student monk demonstration, and the use of active learning in the classroom, among the 11 student respondents.

4.2 Programming Assignment Results

In addition to the survey from students in the introductory programming class in which the student monk demonstration was given, there were some data, obtained from the same class, that suggest that the students may have benefited from the demonstration in regard to their ability to solve a recursive programming problem. Shortly after the student monk demonstration, the students in the class were given an assignment consisting of two parts. Each part required students to code a programming solution to a problem. In Part 1 of the assignment, the problem involved reading several strings from a file and determining which strings were palindromes. It was required that this problem be solved recursively. In Part 2 of the assignment, the programming problem involved calculating the remaining balance owed on a loan after each monthly payment was made, using a graphical user interface (GUI) that included a button that was to be pressed after a monthly payment was made. Prior to receiving this two-part assignment, students had received 2 and one-half class lectures on event driven programming and GUI, 2 class lectures on recursion, and had experienced the student monk demonstration. Each of the two parts of the assignment was equally weighted in terms of grade on the assignment. The recursive programming problem in Part 1 was more difficult in terms of developing a solution to the problem. The solution to the GUI problem in Part 2 was more straightforward (the formula for computing remaining balance was given to the students as part of the problem statement from the textbook used in the class). The GUI requirements were minimal, requiring only a button and an output label. While the solution to the GUI problem was easier, it involved more coding because of the requirement of creating a graphical user interface. Both parts of the assignment were graded by the instructor of the class.

It is interesting to compare the performance of the students on these 2 problems, because it does allow for comparison among the same group of students, who were at the same level of programming experience, and who were taught by the same instructor. Two students in the class of 12 had not turned in the previous assignment, and were obviously falling behind in their coursework. Those 2 students never did turn in any part of this two-part assignment. Among the remaining 10 students, 8 attempted Part 1, the recursive programming problem, and 10 attempted Part 2**,** the GUI programming problem. Seven of the 8 students who attempted Part 1, the recursive programming problem, received full credit. Only 4 of 10 students who attempted Part 2, the GUI problem, received full credit (Figure 4). Among the 8 students who attempted the recursive programming problem, the mean score was 93.3%. Among the 10 students who attempted the GUI problem, the mean score was 81.3% (Figure 5). Also, it was noted that among all students who attempted both problems, all scored higher, or the same, on the recursive programming problem. Using all 12 students in the class for comparison, 4 students received 0% for the recursive programming problem, and 2 students received 0% for the GUI problem, because no attempt was made by these students to solve those problems. Including all these scores of 0%, the mean score for the recursive programming problem was 62.3%, and the mean score for the GUI problem, was 67.8% (Figure 5).

From the authors' many years of experience in teaching programming courses, and Java in particular, it wasn't surprising to see more (however, only 2 more) students attempt the GUI problem than the recursive problem on the assignment. Most programming students are fascinated with writing a program that can produce a visually appealing windows interface that allows for interaction by the user of the program. On the other hand, in the past, students in a first course in programming have been observed to express uneasiness at having to solve recursive problems, perhaps due to students' perception of recursion as being difficult. Indeed, in the class in which the demonstration was conducted, student response to Question 6 of the survey (Table 5) indicates that the students in this class perceived recursion as difficult. However, in this small sample, those students who did attempt the recursive programming problem did perform very well, and 8 of the 10 students who had been keeping up with the work in the class did at least attempt to solve the recursive programming problem.

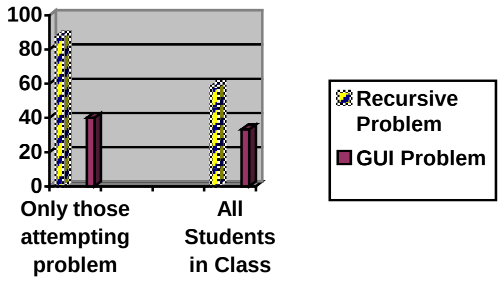

Figure 4. Percent of Students Receiving Full Credit on Problem

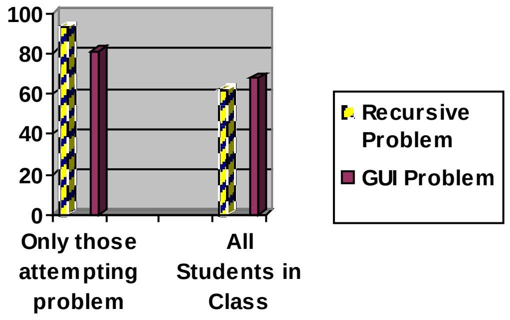

Figure 5. Mean scores on programming problems

The instructor of the class expected that the scores on the GUI problem would have been higher relative to the scores on the recursive programming problem. The reasons for this expectation included the fact that i) the GUI problem was a more straightforward, non-recursive problem, with the needed formula having been supplied in the problem statement from the textbook; and ii) students generally enjoy working on graphical user interface programs. Perhaps the student monk demonstration helped students understand recursion better, which in turn, may have helped students solve the recursive programming problem in the assignment.

5. SUMMARY AND CONCLUSIONS

A form of active learning that uses "student monk" volunteers in a classroom demonstration of the recursive Towers of Hanoi solution is proposed as a way to help instructors teach recursion in an introductory IS or CS programming class, independent of the programming language used. This proposal is made after a literature review of research on teaching recursion failed to reveal any studies or proposals that included an active learning approach using human subjects in a demonstration of recursion.

The pedagogical benefits of active learning in the classroom have been shown (Cassel, 2002; Depradine and Gay, 2004; Massey et al., 2005; McConnell, 1996; Umble and Umble, 2004; Walker, 2004). Using student monk volunteers in the demonstration of the Towers of Hanoi recursive solution allows students to visualize the amount of work that is actually done in a single recursive call. During the demonstration, it was observed that the students participating in this demonstration, and the students who watched the demonstration in the classroom, thoroughly enjoyed the experience. Similar enthusiastic reaction from students during active learning exercises has been observed by others who have done research in active learning activities (Cassell, 2002; Massey et al., 2005; Umble, M. and Umble, E., 2004; Walker, 2004).

A survey was completed anonymously by students who had experienced and participated in the Towers of Hanoi student monk demonstration. The survey results, found in Section 4, indicate an overall perceived effectiveness of the student monk demonstration in helping the students understand the overhead involved in recursion. Also, according to survey results, the demonstration helped students understand the recursive solution to the Towers of Hanoi problem. Results from the survey also indicate that the students believe that active participation helps them to better understand programming concepts. This student perception of the pedagogical benefits of active participation is in line with reported results found in previous research (Cassel, 2002; Depradine and Gay, 2004; Massey, et al., 2005; McConnell, 1996; Umble, M. and Umble, E., 2004; Walker, 2004).

An analysis of a student programming assignment, given in Section 4, also suggests benefits of using the student monk demonstration. The programming assignment consisted of two parts -- a recursive programming problem, and a non-recursive programming problem that entailed creation of a graphical user interface (GUI). Results of this analysis showed a mean score of 93.3% on the recursion problem for those who attempted it, compared to a mean score of 81.3% on the GUI problem for those who attempted it. Also, among all students who attempted both parts (recursion part and GUI part), all scored higher, or the same, on the recursive programming problem. One possible reason for this level of student performance on the recursion program is use of the in-class student monk demonstration. It may be that the student demonstration created more interest in programming recursive solutions, and it may have helped students to better understand recursion.

While the student performance on the recursion programming problem is suggestive of pedagogical benefits associated with the student monk demonstration, the size of the class was small. In a future study using a large introductory programming class, the class could be divided into two groups, each group having an equal number of students. The student monk demonstration would be given to only one of the groups. During the short period of time that the demonstration was being given to that group, the other group would not be in attendance. In this way, all students will have had the same instructor, the same programming background, the same lecture material, and the same programming learning experiences, except of course, for the Towers of Hanoi student monk demonstration. A carefully designed instrument of measurement would include questions on recursion that would be given to both groups of the class. Differences in measurements between the two groups of the class could then be statistically analyzed for significance.

The Towers of Hanoi student monk demonstration is but one of many examples where active learning may be used in an information systems or computer science class. There are many different opportunities for instructors to use student volunteers in an active learning demonstration, as was done with the student monk demonstration. One such example would have students lining up in a row in the front of the classroom, each holding a shoe box, representing a memory location. Assignment statement execution could be illustrated by having a data value, written on an index card, placed in the appropriate shoe box. In an object-oriented programming (OOP) class, to illustrate the concept of an object, a demonstration would entail using students, each one representing an object and holding an index card with field names and method (function) names written on it. Using this model, various features of OOP can be demonstrated by developing appropriate activities. As a final example, a linked list (Figure 1) could be demonstrated by using students, each one representing a node in the list. Each student would hold an index card representing the data stored in the node, and an arrow pointing to the next student node in the list. It would seem that limits on the use of such active learning activities are set only by an instructor's imagination. Computer programming, in particular, lends itself well to the use of such activities, because of the plethora of abstract concepts that can be demonstrated through the use of active learning exercises.

It must be noted, however, that most active learning exercises to be used in the classroom will require a good deal of preparation on the part of the instructor, and possibly on the part of the students as well, depending on the nature of the activity. As reported in Section 2, lack of preparation for the active learning exercise can lead to an unsuccessful learning experience. In particular, it is important to note that for the Towers of Hanoi student monk demonstration, a well-planned training session of the student monks is necessary for an effective presentation. Also, of course, the instructor who explains the demonstration to the student monk volunteers must be ready to answer any questions regarding the presentation. Detailed directions (Appendix A) should be given to each student monk volunteer explaining his or her particular role in the demonstration. When proper training occurs, the use of student monks in a classroom demonstration is proposed as an excellent way to demonstrate the recursive solution to the classical Towers of Hanoi problem.

6. REFERENCES

Aho, A., Hopcroft, E., Ullman, J. (1974) The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts.

Anderson, J., Douglass, S. (2001) “Tower of Hanoi: Evidence for the Cost of Goal Retrieval”, Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol.27, No. 6, pp. 1331-1346.

Anderson, J. R. (1976) Language, Memory, and Thought, Lawrence Erlbaum Associates, Hillsdale, NJ.

Anderson, J.R., Pirolli, P., and Farrell, R. (1988) Learning to Program in Recursive Functions, in The Nature of Expertise, Lawrence Erlbaum Associates, Hillsdale, NJ.

Benander, A., Benander, B., Pu, H. (1996) “Recursion vs. Iteration: An Empirical Study of Comprehension”, Journal of Systems and Software, Vol. 32, No. 1, pp. 73-82.

Benander, A., Benander, B., Sang, J. (2000) “An Empirical Analysis of Debugging Performance – Differences Between Iterative and Recursive Constructs”, Journal of Systems and Software, Vol. 54, No. 1, pp. 17-28.

Birtwistle, G. (1985) “The coroutines of Hanoi, SIGPLAN Notices, Vol. 20, No. 1, pp 9-10.

Bruce, K., Danyluk, A. Murtagh, T. (2005) “Why Structural Recursion Should Be Taught Before Arrays in CS1”, ACM SIGCSE Bulletin, Vol. 37, No. 1, pp. 246-250.

Cassell, L. (2002) “Very Active Learning of Network Protocol”, ACM SIGCSE Bulletin, Vol. 34, No. 3, p. 195.

Dale, N. (2003) C++ Plus Data Structures, 3rd Edition, Jones and Bartlett, Sudbury, Massachusetts.

Danvy, O. (2002) “There and Back Again”, ACM SIGPLAN Notices, Vol. 36, No. 9. pp. 230-234.

Deitel, H. M., Deitel, P.J. (2005) Java How to Program, 6th Edition, Prentice Hall, Upper Saddle River, New Jersey.

Depradine, C., Gay, G. (2004) “Active Participation of Integrated Development Environments in the Teaching of Object-Oriented Programming”, *Computers and Education*, Vol. 43, No. 3, pp. 291-298.

Ford, G. (1984) “An Implementation-Independent Approach to Teaching Recursion”, ACM SIGCSE Bulletin, Vol. 16, No. 1, pp. 213-216.

Gotschi, T., Sanders, I., and Galpin, V. (2003) “Mental Models of Recursion”, ACM SIGSCE Bulletin, Proceedings of the 34th SIGSCSE Technical Symposium on Computer Science Education, Vol. 35, No. 1, pp. 346-350.

Henderson, P. B. and Romero, F. J. (1989) “Teaching Recursion as a Problem-Solving Tool Using Standard ML”, ACM SIGSCE Bulletin, Proceedings of the 20th SIGSCE Technical Symposium on Computer Science Education, Vol. 21, No. 1, pp. 17-31.

Kessler, C. and Anderson, J. (1986) “Learning Flow of Control: Recursive and Iterative Procedures”, Human-Computer Interaction, Vol. 2, No. 2, pp. 135-166.

Kruse, R. (1982) “On Teaching Recursion”, ACM SIGCSE Bulletin, Vol. 14, No. 1, pp. 92-96.

Levy, D., Lapidot, T. (2000) “Recursively Speaking: Analyzing Students’ Discourse of Recursive Phenomena”, ACM SIGSCE Bulletin, Proceedings of the 31st SIGSCE Technical Symposium on Computer Science Education, Vol. 32, No. 1, pp. 315-319.

Massey, A., Brown, S., Johnston, A. (2005) “It’s All Fun and Games…Until Students Learn”, *Journal of Information Systems Education*, Vol. 16, No. 1, pp. 9-14.

Mayer, H. and Perkins, D. (1984) “Towers of Hanoi Revisited: A Nonrecursive Surprise”, SIGPLAN Notices, Vol. 19, No. 2, pp. 80-84.

Maziar, S. (1985) “Solution of the Towers of Hanoi Problem Using a Binary Tree”, SIGPLAN Notices, Vol. 20, No. 5, pp. 16-20.

McConnell, J. (1996) “Active and Group Learning Techniques and Their Use in Graphics Education”, Computer & Graphics, Vol. 20, No. 1, pp. 177-180.

McCracken, D. (1987) “Ruminations on Computer Science Curricula”, Communications of the ACM, Vol. 30, No. 1, pp. 3-5.

Noyes, J., Garland, K. (2003) “Solving the Tower of Hanoi: Does Mode of Presentation Matter?”, Computers in Human Behavior, Vol. 19, No. 4, pp. 579-592.

Pirolli, P. L., Anderson, J. R. (1985) “The Role of Learning from Examples in the Acquisition of Recursive Programming Skills”, Canadian Journal of Psychology, Vol. 39, No. 2, pp. 240-272.

Sapir, A. (2004) “The Tower of Hanoi with Forbidden Moves”, The Computer Journal, Vol. 47, No. 1, pp. 20-24.

Sinha, A. and Vessey, I. (1992) “Cognitive Fit: An Empirical Study of Recursion and Iteration”, IEEE Transactions on Software Engineering, Vol. 18, pp. 368-379.

Tung, S., Chang, C., Wong, W., Jehng, J. (2001) “Visual Representations for Recursion”, International Journal of Human-Computer Studies, pp. 285-300.

Turbak, F., Royden, C., Stephan, J., and Herbst, J. (1999) “Teaching Recursion Before Loops in CS1”, Journal of Computing in Small Colleges, Vol. 14, No. 4, pp. 86-101.

Umble, M., Umble, E. (2004) “Using Active Learning to Transform the Monte Hall Problem into an Invaluable Classroom Exercise”, Decision Sciences Journal of Innovative Education, Vol. 2, No. 2, pp. 213-217.

Walker, G.N. (2004) “Experimentation in the Computer Programming Lab”, SIGCSE Bulletin, Volume 35, No. 34, pp. 69-72.

Wiedenbeck, S. (1988) “Learning Recursion as a Concept and as a Programming Technique”, ACM SIGSCE Bulletin, Proceedings of the19th SIGSCE Technical Symposium on Computer Science Education, Vol. 20, No. 1, pp. 275-278.

Wiedenbeck, S. (1989) “Learning Iteration and Recursion from Examples”, International Journal of Man-Machine Studies, Vol. 30, No. 1, pp. 1-22.

Wu, C., Dale, N., Bethel, L. (1998) “Conceptual Models and Cognitive Learning Styles in Teaching Recursion”, ACM SIGCSE Bulletin, Vol. 30, No. 1, pp. 292-296.

AUTHOR BIOGRAPHIES

Dr. Alan Benander is a Professor in the Computer and

Information Science Department at Cleveland State University. He teaches a variety of courses in CIS, including programming, databases and analysis of algorithms. His research interests are in the areas of software engineering and computer education.

Dr. Barbara Benander is a Professor in the Computer and

Information Science Department at Cleveland State University, where she teaches various CIS courses, including mobile computing and distributed application development. Her research areas include software metrics and the use of active learning in the classroom.

APPENDIX A
TRAINING THE 4 STUDENT MONKS

I. Setup and General Training Instructions

During the training session, (and in the actual demonstration) 4 Student Monks, facing their classmates, will stand behind the Towers of Hanoi prop which is in the front of the classroom.

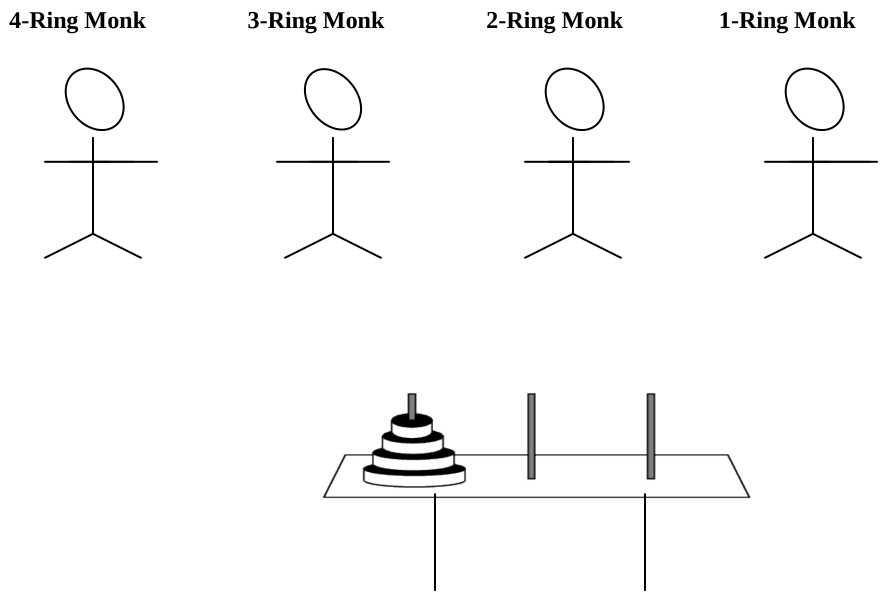

Table with Prop in Front of Classroom
4 Student Monks Standing Behind Table
4 Rings Initially on Peg A to be moved to Peg C
Diagram A. Initial Setup for 4-Ring Demonstration

All monks will have a pad of paper and pencil. On the top half of the paper they will mark letters corresponding to arguments with which they are called. On the bottom half they will make a mark every time they move a ring. The 4-ring monk will be called (once only) by the instructor, e.g., with the instructor saying, "Towers (4, A, B, C)". The intent of this call is to (eventually) have all the 4 rings moved from Peg A to Peg C. The 3-ring monk will only be called by the 4-ring monk (the 3-ring monk will be called twice by the 4-ring monk during the entire demonstration). The 2-ring monk will only be called by the 3-ring monk (the 2-ring monk will be called a total of 4 times by the 3-ring monk during the entire demonstration). The 1-ring monk will only be called by the 2-ring monk (the 1-ring monk will be called a total of 8 times by the 2-ring monk during the entire demonstration). Whenever a monk makes a call, the monk takes one step forward, makes the call, steps back in line, and waits for the monk that was called to shout, "Finished!"

II. Detailed Training Instructions for Each of the 4 Monks

Training Instructions Given to the 4-Ring Monk

You will only be called by the instructor. For this demonstration, initially, all the 4-rings will be placed on Peg A, and are to be (eventually) all moved to Peg C. When the instructor shouts, "Towers (4, A, B, C)", jot the letters A, B, C on your paper to help you remember the order of the arguments. Then you step forward and make a call to the 3-ring monk by switching the order of the last 2 letters on your sheet, keeping the first letter in the first position. So, in this example, you would shout, "Towers (3, A, C, B)". Notice that you simply switch the order of the last 2 letters on your paper when you make this first call to the 3-ring monk. After calling the 3-ring monk, take a step back, and wait for the 3-ring monk to shout, "Finished!" When that happens (it may take a few minutes), you step forward to the table and move a ring from the peg with the first letter on your paper, to the peg with the last letter on your paper (Peg A to Peg C, in this example demonstration). After moving the ring, make a mark on the bottom half of your paper (to keep track of number of rings you move), and make a call to the 3-ring monk by switching the order of the first 2 letters on your sheet. In this example, since you had written, A, B, C on your paper, you will shout out the call, "Towers(3, B, A, C)." After making this call, step back in line. When the 3-ring monk eventually shouts, “Finished!”, you step forward and shout “Finished”, also. Then step back in line, and cross out the letters on your sheet, since you are finished with your call from the instructor.

Training Instructions Given to the 3-Ring Monk

You will only be called by the 4-ring monk. You will be called twice by the 4-ring monk in this 4-ring demonstration. For this demonstration, you will initially be called by the 4-ring monk with the call, “Towers (3, A, C, B).” Whenever you are called by the 4-ring monk, jot down the order of the letters to help you remember the order of the arguments. So, in this first call from the 4-ring monk, you will jot down A, C, B. Then you step forward and make a call to the 2-ring monk by switching the order of the last 2 letters on your sheet, keeping the first letter in the first position. So, in this example, you would shout, “Towers (2, A, B, C).” Notice that you simply switch the order of the last 2 letters on your paper when you make this first call to the 2-ring monk. After making this first call to the 2-ring monk, take a step back, and wait for the 2-ring monk to shout, “Finished!” When that happens (it may take a minute or so), you step forward and move a ring from the peg with the first letter on your paper, to the peg with the last letter on your paper (Peg A to Peg B, in this example). After moving the ring, make a mark on the bottom half of your paper (to keep track of number of rings you move), make a call to the 2-ring monk by switching the order of the first 2 letters on your sheet. In this example, since you had written, A, C, B on your paper, you will shout out the call, “Towers(2, C, A, B).” When the 2-ring monk shouts, “Finished!”, you shout “Finished”, also. Cross out the letters on your sheet, since you are finished with your call from the 4-ring monk.

Remember, during the entire 4-ring demonstration, you will be called twice by the 4-ring monk – follow the same procedure outlined above each time you are called by the 4-ring monk: you should write letters on your sheet in the order they were given to you in the call, step forward and call the 2-ring monk by switching the order of the last 2 letters on your sheet, step back in line, wait for the 2-ring monk to shout, “Finished!”, then step forward and move a ring – from peg with 1st letter on sheet, to peg with last letter on your sheet, make a mark on the bottom half of your sheet—indicating another ring moved by you, call the 2-ring monk again by switching the order of the first 2 letters on your sheet, step back in line, and wait for the 2-ring monk to shout “Finished!” When this occurs, step forward, and shout “Finished!”, step back in line, then cross out (or erase) the letters on your sheet.

Training Instructions Given to the 2-Ring Monk

You will only be called by the 3-ring monk. You will be called a total of 4 times by the 3-ring monk in this 4-ring demonstration. For this demonstration, the first time you are ever called by the 3-ring monk it will be with the call “Towers (2, A, B, C).” Whenever you are called by the 3-ring monk, jot down the order of the letters to help you remember the order of the arguments. So, in this very first call from the 3-ring monk, you will jot down A, B, C. Then you step forward and make a call to the 1-ring monk by switching the order of the last 2 letters on your sheet, keeping the first letter in the first position. So, in this example, you would shout, “Towers (1, A, C, B).” Notice that you simply switch the order of the last 2 letters on your paper when you make this first call to the 1-ring monk. After making this first call to the 1-ring monk, take a step back, and wait for the 1-ring monk to shout, “Finished!” When that happens (it should only take a few seconds), you step forward and move a ring from the peg with the first letter on your paper, to the peg with the last letter on your paper (Peg A to Peg C, in this example). After moving the ring, make a mark on the bottom half of your paper (to keep track of number of rings you move), make a call to the 1-ring monk by switching the order of the first 2 letters on your sheet. In this example, since you had written, A, B, C on your paper, you will shout out the call, “Towers(1, B, A, C).” When the 1-ring monk shouts, “Finished!”, you shout “Finished”, also. Cross out the letters on your sheet, since you are finished with your call from the 3-ring monk.

Remember, during the entire 4-ring demonstration, you will be called 4 times by the 3-ring monk – follow the same procedure outlined above each time you are called by the 3-ring monk: you should write letters on your sheet in the order they were given to you in the call, step forward and call the 1-ring monk by switching the order of the last 2 letters on your sheet, step back in line, wait for the 1-ring monk to shout, “Finished!”, then step forward and move a ring – from peg with 1st letter on sheet, to peg with last letter on your sheet, make a mark on the bottom half of your sheet—indicating another ring moved by you, call the 1-ring monk again by switching the order of the first 2 letters on your sheet, step back in line, and wait for the 1-ring monk to shout “Finished!” When this occurs, step forward, and shout “Finished!”, step back in line, then cross out (or erase) the letters on your sheet.

Training Instructions Given to the 1-Ring Monk

You may have the easiest task of all the monks. But you will also be doing the most ring moves of any monk. You will only be called by the 2-ring monk. You will be called a total of 8 times by the 2-ring monk in this 4-ring demonstration. Whenever you are called by the 2-ring monk, step forward and move a ring from the peg having the 1st letter in the call to the peg with the last letter in the call. Make a mark on your paper whenever you move a ring to keep track of the number of rings that you move. After moving a ring, simply shout “Finished!” and move back in line. For example, the very first time that the 2-ring monk calls you, it will be with the call, “Towers(1,A, C, B).” When you hear this call you will step forward, move a ring from Peg A to Peg B, make a mark on your paper, shout “Finished!” and step back in line.

APPENDIX B
WALKTHROUGH OF 4-RING DEMONSTRATION

See Appendix A for initial setup of students and prop, and for training of student monks.

1. Instructor shouts, “Towers (4, A, B, C).”
2. 4-ring monk marks letters: ABC on paper, steps forward and calls, “Towers(3, A, C, B)” and then steps back in line.
3. 3-ring monk marks letters ACB on paper, steps forward and calls, “Towers(2, A, B, C)” and then steps back in line.
4. 2-ring monk marks letters ABC on paper, steps forward and calls, “Towers(1, A, C, B)” and then steps back in line.
5. 1-ring monk steps forward and **moves ring from Peg A to Peg B**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
6. 2-ring monk steps forward and **moves a ring from Peg A to Peg C**, makes a mark on the sheet for counting rings moved, calls “Towers(1, B, A, C)”, and steps back in line.
7. 1-ring monk steps forward and **moves ring from Peg B to Peg C**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
8. 2-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
9. 3-ring monk steps forward and **moves a ring from Peg A to Peg B**, makes a mark on the sheet for counting rings moved, calls “Towers(2, C, A, B)”, and steps back in line.
10. 2-ring monk marks letters CAB on paper, steps forward and calls, “Towers(1, C, B, A)” and then steps back in line.
11. 1-ring monk steps forward and **moves ring from Peg C to Peg A**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
12. 2-ring monk steps forward and **moves a ring from Peg C to Peg B**, makes a mark on the sheet for counting rings moved, calls “Towers(1, A, C, B)”, and steps back in line.
13. 1-ring monk steps forward and **moves ring from Peg A to Peg B**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
14. 2-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
15. 3-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
16. 4-ring monk steps forward and **moves a ring from Peg A to Peg C**, makes a mark on the sheet for counting rings moved, calls “Towers(3, B, A, C)”, and steps back in line.
17. 3-ring monk marks letters BAC on paper, steps forward and calls, “Towers(2, B, C, A)” and then steps back in line.
18. 2-ring monk marks letters BCA on paper, steps forward and calls, “Towers(1, B, A, C)” and then steps back in line.
19. 1-ring monk steps forward and **moves ring from Peg B to Peg C**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
20. 2-ring monk steps forward and **moves a ring from Peg B to Peg A**, makes a mark on the sheet for counting rings moved, calls “Towers(1, C, B, A)”, and steps back in line.
21. 1-ring monk steps forward and **moves ring from Peg C to Peg A**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
22. 2-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
23. 3-ring monk steps forward and **moves a ring from Peg B to Peg C**, makes a mark on the sheet for counting rings moved, calls “Towers(2, A, B, C)”, and steps back in line.
24. 2-ring monk marks letters ABC on paper, steps forward and calls, “Towers(1, A, C, B)” and then steps back in line.
25. 1-ring monk steps forward and **moves ring from Peg A to Peg B**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
26. 2-ring monk steps forward and **moves a ring from Peg A to Peg C**, makes a mark on the sheet for counting rings moved, calls “Towers(1, B, A, C)”, and steps back in line.
27. 1-ring monk steps forward and **moves ring from Peg B to Peg C**, makes a mark on the sheet for counting rings moved, shouts “Finished!”, and then steps back in line.
28. 2-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
29. 3-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.
30. 4-ring monk steps forward, shouts “Finished!”, steps back in line, and crosses out letters on sheet.

Instructor, proclaims, “Great work, Monks!”

APPENDIX C
STUDENT SURVEY OF STUDENT MONK DEMONSTRATION

Recall the demonstration involving the 4 student monks that was used to demonstrate the recursive programming solution to the Towers of Hanoi problem. Please answer the following 6 questions. This survey should take approximately 5 minutes of your time. Do not put your name on this survey. Thank you!

In order to be able to type your answers into the survey, you must click on the “Reply button” – you will then be able to type your answers directly into the survey. After you have completed the survey, click on the “Send” button to send it.

1. Do you agree or disagree with the following statement?

“The Towers of Hanoi Monks demonstration helped me to understand the overhead involved in using recursive code (e.g., the many recursive calls that are actually made in a recursive solution)”.

1. Strongly Agree 2. Agree 3. Neither agree nor disagree 4. Disagree 5. Strongly disagree

1, 2, 3, 4 or 5? ____

2. Do you agree or disagree with the following statement?

“The Towers of Hanoi Monks demonstration was more helpful to me in understanding the solution to the Towers of Hanoi problem than a classroom lecture alone would have been.”

1. Strongly Agree 2. Agree 3. Neither agree nor disagree 4. Disagree 5. Strongly disagree

1, 2, 3, 4 or 5? ____

3. Do you agree or disagree with the following statement?

“I paid more attention to the Towers of Hanoi Monks demonstration than I would have paid to a simple class lecture from the instructor explaining the Towers of Hanoi.”

1. Strongly Agree 2. Agree 3. Neither agree nor disagree 4. Disagree 5. Strongly disagree

1, 2, 3, 4 or 5? ____

4. Do you agree or disagree with the following statement?

“The Towers of Hanoi Monks demonstration was a good use of class time.”

1. Strongly Agree 2. Agree 3. Neither agree nor disagree 4. Disagree 5. Strongly disagree

1, 2, 3, 4 or 5? ____

5. Do you agree or disagree with the following statement?

“Involving students in active participation in the classroom helps them to better understand programming concepts.”

1. Strongly Agree 2. Agree 3. Neither agree nor disagree 4. Disagree 5. Strongly disagree

1, 2, 3, 4 or 5? ____

6. On a scale of 1 (very difficult) to 5 (very easy) how difficult, in general, did you find the topic of recursion? _______

Information Systems & Computing
Academic Professionals

EDSIG
Serving Information Systems Educators

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096