

# Commonsense Computing: using student sorting abilities to improve instruction

Tzu-Yi Chen  
Computer Science Dept.  
Pomona College  
Claremont, CA 91711 USA  
tzuyi@cs.pomona.edu

Gary Lewandowski  
Mathematics and Computer  
Science Dept.  
Xavier University  
Cincinnati, OH 45207 USA  
lewandow@cs.xu.edu

Robert McCartney  
Computer Science and  
Engineering Dept.  
University of Connecticut  
Storrs, CT 06269 USA  
robert@cse.uconn.edu

Kate Sanders  
Mathematics and Computer Science Dept.  
Rhode Island College  
Providence, RI 02908 USA  
KSanders@ric.edu

Beth Simon  
Computer Science and Engineering Dept.  
Univ. of California San Diego  
La Jolla, CA 92093 USA  
esimon@cs.ucsd.edu

## ABSTRACT

We examine students' commonsense understanding of computer science concepts before they receive any formal instruction in the field. For this study, we asked students on the first day of a CS1 class to describe in English how they would arrange a set of numbers in ascending, sorted order; we then repeated the experiment asking students to sort a list of dates (in mm/dd/yyyy format).

We found that a majority of students described a coherent algorithm; some described versions of insertion or selection sort, but many gave unexpected algorithms. We also found significant differences between responses given for sorting numbers versus dates. Based on our analysis of the data we suggest that beginning-programming instructors more explicitly discuss data types, begin loop instruction with post-test loops, assist students in recognizing implicit conditional and iteration use in natural language solutions to problems, and recognize that novices and experts focus on different aspects of the problem in even basic problem solving tasks.

## Categories and Subject Descriptors

K.3.2 [Computer Science Education]: Introductory Programming—*abstract programming concepts*

## General Terms

Algorithms, Human Factors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

## Keywords

CS1, preconceptions, resources, naïve, beginner, constructivism, sorting

## 1. INTRODUCTION

This paper reports on the second in a series of projects investigating “commonsense computing”: what students know about computing concepts before having formal instruction.

Recent studies of computing students indicate that introductory students lack certain skills: both their ability to write programs [14], and their ability to read and trace code [13] are well below what we might expect. Ben-David Kolikant [3] found that students apply a sense of “mostly correct” to their programs, suggesting they do not even know what it would mean for their programs to work! These results are independent of the programming language and paradigm of instruction.

On the other hand, studies have also demonstrated that students have considerable ability to reason about computer science topics. Ben-David Kolikant [2] found students could solve problems requiring concurrency; Gibson and O’Kelly [9] found pre-college students could solve a variety of search problems and beginning computer-science students could prove results about the Towers-of-Hanoi problem.

This disconnect between demonstrated programming knowledge and demonstrated reasoning skills suggests students have considerable knowledge that we, as instructors, can leverage to teach computer science more effectively. To leverage students’ existing knowledge, however, we must first determine what that knowledge is.

For our initial projects, we chose to focus on sorting, because it not only encompasses key computing concepts such as models of storage, iteration, and conditional evaluation, but also is a “real world” task occurring naturally in students’ lives. The first project in this series, described in [18], investigated students’ knowledge of algorithms and sorting by giving them a list of ten positive integers and asking them to write a paragraph describing how they would put those

numbers in ascending order. That data was gathered in the fall and winter of 2005-2006.

The second project, carried out in Spring 2006, used essentially the same question, except that the data to be sorted was a list of ten dates, written in the format month/day/year using numeric values (e.g., 7/31/2006).

The discussion in this paper is on the new date-related data and the new results and implications for teaching that come out of joint analysis across both data sets. The specific questions addressed in this paper are:

- Can students describe algorithms to sort numbers and dates?
- Are there differences in the approaches taken to these two tasks?
- Do student preconceptions of sorting have implications for teaching introductory computer science?

The results suggest that beginners can describe algorithms in both cases, but there are differences observed for the two tasks. Specifically, in the date results we see issues of data representation and difference in student focus than in the numerical sorting. We also confirm student preference for post-test loops. The data further suggest other implications for teaching.

The rest of this paper is organized as follows. In Section 2 we review the related work on preconceptions. In Section 3 we discuss our research methodology. In Section 4 we present the results of the study. In Section 5 we discuss the implications of these results for teaching.

## 2. RELATED WORK

This work is motivated by the constructivist view of learning, which holds that learning takes place by refining and extending what the student already knows [1, 6]. Bransford et al. [5] argue that learning must engage the students' preconceptions to be effective. Schwill [17] applies Bruner's notion of fundamental ideas [6] to computing, and argues that the fundamental computing ideas, which provide a framework for learning constructively, have meaning in everyday life, and can be described in ordinary language.

Several researchers have studied student preconceptions:

- Miller [15] analyzed "natural language" programs by students who had not had a formal programming course, with the purpose of exploring the idea of writing computer programs in natural language. He found that a number of standard programming concepts showed up in these natural language descriptions, but that there were differences between these and programs in computer languages, especially in terms of knowledge implicit in context or general world knowledge.
- Onorato and Schvaneveldt [16] also looked at natural language descriptions of a programming task, comparing subjects drawn from different pools: *naïve*-students with no programming experience, *beginner*-students currently taking their first programming course, and *expert*-students with a good deal of programming experience. The task was to explain how to find a name in a telephone directory either to a person, to a person without any knowledge of telephones (they specified George Washington), or to a computer. Along

with differences between experts and novices, they also found differences between the naïves and beginners though neither had experience programming.

- While studying misconceptions of novice programmers, Bonar and Soloway [4] specifically considered preprogramming knowledge, which they call "step-by-step natural language programming knowledge." They distinguish this preprogramming knowledge from knowledge of the programming language Pascal, which the students were learning in their introductory course. They found that many of the observed bugs could be explained by a mismatch between students' knowledge in these two different domains.
- Ben-David Kolikant [2] looked at student preconceptions about concurrency, asking whether students could solve some simple tasks that require synchronization. She found that students with no experience with synchronization in computing were able to draw upon real-life experience to come up with the necessary mechanisms.
- Gibson and O'Kelly [9] looked at a variety of search problems (with pre-college students) and Towers-of-Hanoi problems (with beginning computing students), and found that both groups showed "algorithmic understanding" of how to solve these problems—they were able to consider and reason about the process used to solve the problem, not just find a solution.

Finally, there is a substantial body of work both in computing and other disciplines on *misconceptions*: incorrect concept understandings that need to be replaced with correct models. Clancy [7] provides a survey of this work in computer science; [8] (Ch. 4) gives an overview and provides references in science education. Smith et al. [19] challenge this view in the context of math and science education, arguing that misconceptions are limited mental models that can be built upon to gain correct understanding.

Like Smith et al. [19], Hammer [11], and Ben-David Kolikant [2], we seek preconceptions that might be built upon to help students learn specific concepts within a particular context. This is in contrast to a number of studies that build models that predict student success based on their background knowledge and experience.

## 3. METHODOLOGY

So far, we have carried out two projects as part of this study. The first dealt mainly with sorting a list of integers and is specifically reported on in [18]. The second dealt with sorting a list of dates. The methodology was similar in both projects.

### 3.1 The tasks

The first project used the following task (the Number Sort):

Write a paragraph in complete English sentences describing how you would arrange a set of 10 numbers in "ascending sorted order" – that is, from smallest to largest. You might consider the following list of numbers, but make sure that your paragraph describes how to do it with any 10 numbers.

We also used a few variants of the list, including negative and real values, without significantly different results.

The second project used the following task (the Date Sort):

Write a paragraph in complete English sentences describing how you would arrange a set of 10 dates in "ascending sorted order," that is, from earliest to latest. You might consider the following list of dates, but make sure that your paragraph describes how to do it with any 10 dates. The dates are in month/day/year format.

12/21/2004 5/1/1988 7/21/1970 8/28/2001 1/31/2002  
6/6/2004 5/20/1988 11/5/1970 4/2/2001 9/9/2002

## 3.2 Subjects

Both tasks were given to beginner students – those taking a first computer science course without prior background in computer science. The Number Sort was given to students at two institutions (N=118). The Date Sort was given to students at three institutions (N=75).

## 3.3 Tagging the data

In each data set, the responses were analyzed for correctness, approach to the problem, the use of control structures. In each of these areas, at least two researchers participated in the coding of the data, with disagreements worked out in discussion with a third researcher. In addition, we computed a few more objective results, such as the number of words in each response, a straightforward process undertaken by one researcher.

**Correctness:** For both tasks, we asked whether the response worked, both for the specific example given and in the general case. To determine whether the response generated an algorithm we could follow, we read the responses with the list of values in front of us and tried to sort the values following the instructions. After evaluating the sort based on the given values, we then considered whether or not that process would have worked on any list.

**Approach:** How do students approach the data as they sort? For the Number Sort, we described each student's approach as String (the values are a string of digits), Numeric (the values are numbers), or Other. For the Date Sort, we distinguished among responses where the student never mentioned any sub-parts of the data at all; broke the data into year, month, day; mentioned only the year; or broke the data down into month, day, year, and broke those values down even further into digits. Finally, there were students who pre-processed the data by concatenating the three sub-parts into one value and then discussing a digit-based sort.

**Focus:** We also examined the approach in terms of the descriptive focus of the subjects, using a grounded theory approach. For the Number Sort, we used a grounded-theory approach. This is a qualitative research method in which theories are derived inductively through close examination of the data [10]. Accordingly, we developed a list of foci through close reading of the student responses.

**Control Structures:** For both tasks, we asked whether students showed a knowledge of control structures, specifically: Did they use iteration (i.e., describe the repetition

of a particular process to achieve the sorted order)? and Did they use conditionals? We tagged each response individually, but also did a word count of terms such as "if," "repeat," "while," "until," and so on.

**Examples:** For both tasks, we categorized responses based on whether or not the student used an example to illustrate their algorithm.

**Content Analysis:** For both tasks, we examined some of the surface characteristics of the responses: How concise is the response? (Measured by counting the number of words used.) How many "computer science" terms are included? (Measured by building a list of computer science terms and counting their occurrence in each response.)

## 4. RESULTS

In [18] we report in detail on the first phase of this study. We include only the results pertinent to our overall discussion in this section, along with the results of the second phase of the study involving dates. When reporting a statistical analysis, we provide the raw value of the test statistic, the number of degrees of freedom and the attained significance ( $p$ ) for each test. For all tests, we set  $\alpha = .05$ , and assume unequal variances for unpaired t-tests.

**Correctness:** In the Number Sort, 69% of the students gave a correct response for the specific set of numbers provided, but only 57% gave a correct response for a general list of 10 numbers. In particular, many students gave responses which only worked for numbers < 1000. (We didn't require them to take into account negative numbers or fractions.) In the Date Sort, 65% of students gave a correct response for the specific set of dates, and 61% gave a response that would be correct for any 10 dates. The difference between student performance on the two tasks was not statistically significant in either the specific or the general case.

**Approach:** In the Number Sort, 63% of students gave a String response, for example:

To arrange a set of 10 numbers in ascending sorted order, you would have to first consider how many digits the number has, and what the numbers of the digits are... [Y F07]

Fewer (35%) treated numbers as primitive types. Among these, selection-sort-style approaches were common. String responses were less likely to be correct (69%) than responses that treated numbers as a primitive type (76%), which was attributed to the greater detail required for a String response.

In the Date Sort, fewer responses treated the data as a single unit (only 13%). 75% of the students explicitly broke the data into three parts and considered the year, month, and day separately, and 3% explicitly broke out the year but treated the remainder of the date as a unit. Surprisingly, having broken the data down into separate numbers for the year, or year/month/day, only 9% went further and broke those numbers down into strings of digits.

**Focus:** In the Number Sort, 50% of beginning students focused on grouping the data by the number of digits and then doing a digit-by-digit comparison of the values in those groups; 19% focused on the process of choosing the smallest value in the list and placing it into the growing sorted list

(i.e. a process related to selection sort); 8% focused in detail on how to find the smallest value in the list; 8% focused on how to place the current value under consideration into a sorted list; 7% focused on giving a thorough definition of how values are known to be larger or smaller than each other.

In the Date Sort, for the large majority of responses the main focus was on explicitly dealing with the date in three parts and using those parts to determine which date was earliest. We further examined the responses to see if this focus naturally led to responses with recognizable pieces of selection sort. We found that 25% of the responses focused on the selection of the smallest date, whereas most of the remaining ones sorted within year groups, then month groups, and then by day. In the latter cases, “sort” was a primitive operation, assumed to work based on the breakdown of the data.

**Control Structures:** In the Number Sort, 65% of the students expressed iteration, and 43% expressed conditionals. Use of conditionals was significantly higher in String-approach responses, due to the need to describe how to order numbers that begin with the same digit. The use of control structures is not correlated with the correctness of the answer.

In the Date Sort, only 27% of the students expressed iteration, whereas 60% expressed conditionals. Students used conditionals in a similar context to that of the Number Sort: when describing how to order dates that have the same year. Of responses breaking down the data into parts, 69% of them explicitly expressed this behavior.

Students differed significantly in their use of conditionals ( $\chi^2 = 4.64$ ,  $df=1$ ,  $p = .031$ ), being more likely to use them with dates and less likely with numbers. Students also differed significantly in their choice of whether to use iteration ( $\chi^2 = 24.9$ ,  $df=1$ ,  $p < 10^{-6}$ ), being less likely to use iteration with dates, and more likely to use iteration with numbers.

**Examples:** Whereas exactly 50% of students gave an example in their responses in the Number Sort, only 34% gave an example in the Date Sort. The difference is statistically significant ( $\chi^2 = 4.25$ ,  $df=1$ ,  $p = .039$ ).

**Content Analysis:** Finally, responses were analyzed for the total number of words in the response and the presence of computing-related terminology. On average, in the Number Sort, students used 169.4 words and 1.8 computer science-related terms. In addition to observing that String responses seemed more complex, we found they were also longer (183.6 words versus 148.4 words for Numeric). However, on average Numeric responses contained more computing terms than String respondents (2.5 versus 1.5).

In the Date Sort, students used an average of 123 words per response. The difference between the length of their responses for sorting numbers and dates is significant using a two-tailed t-test assuming unequal variances ( $t = 1.97$ ,  $df=169$ ,  $p = .000285$ ). Students also used significantly fewer computing terms (1.2) when sorting dates ( $t = 1.98$ ,  $df=154$ ,  $p = .0267$ ).

## 5. IMPLICATIONS FOR TEACHING

### 5.1 Natural Problem Solving Skills

Teachers of introductory programming courses frequently explain that we are not just teaching programming, we are teaching problem solving. This study demonstrates that students bring natural skills with them and can describe a process that is frequently followable as an algorithm. We need to recognize that a large piece of teaching programming and problem solving is facilitating the transfer of natural knowledge into the limited context/framework of the algorithmic language provided. The ability of so many students to provide an algorithm that can be followed suggests leveraging their ability to write natural language solutions as they work on writing programs.

### 5.2 Types Are Not Natural

For most of the subjects in this study, there was no notion of a data type. Most of the students chose to treat numbers as strings in the Number Sort, but as integers in the Date Sort. In the Number Sort, the number was often treated as a string of digits with digits being the only comparable piece of data. In the Date Sort, however, students who are otherwise similar to those working on the Number Sort had no problem treating multi-digit values as comparable integers. This suggests the notion of data type needs explanation as it doesn't appear to be natural or even implicit in the responses; giving a short task like the Number Sort may be useful to bring out this distinction.

### 5.3 Loops

Post-test loops are much more natural to students. They were more frequently used in both tasks. “While” is almost never used (20 times over 185 subjects) and “until” occurs more frequently (70 times over the 185 subjects). In many cases the form of the response describes the basic action to be done, and then refers to repeating the process until completed. This pattern of response suggests students may benefit from starting with the more natural post-test loop, then being presented cases in which the while loop makes sense. Over the long-term, because of its universality, it seems likely that students will come to the same conclusion their instructors have – that “while” is the most useful. Over the short term however, the post-test loop provides students a more intuitive control structure.

### 5.4 Implicit uses of conditionals and iteration

Conditionals and iteration are often implicit, and responses using explicit conditionals and iteration are not correlated to correct responses. In our tasks, subjects tended to use iteration when describing the sorting of numbers, and conditionals when describing sorting dates. Thus in each task, implicit use of iteration and conditionals was frequent. An implication of this result is that we may need to help students explicitly notice where iteration or conditional action is needed, unpacking their natural solution into one translatable to a programming language.

### 5.5 Where is the focus?

Most subjects in the study were able to break the problem down and present what they saw as a key sub-problem to be solved. We note two key points here. First, most students worked on the details of only a single sub-problem, for example how to compare two values; in the case of dates, after describing how one pulls apart the date, students would use “order within a year” as a primitive operation, not bothering

to provide further details.

Second, the students chose a level to work at that seemed most natural to them, though it may not seem natural to an experienced computer scientist. The students broke down numbers the way they were taught to in elementary education – they were taught that digits are the basic parts of a number; similarly, they break down dates the most common way as well, into month/day/year. What they did not do, at this early stage, is describe the solution in terms of a data representation supporting the most efficient algorithm. Similarly, they did not worry about control structures that they find to be obvious, discussing “grouping” of data as a completely primitive operation.

The focus of the responses illustrates the difference in cognitive model that beginning students bring to formal instruction in computer science. The single focus of the response resonates with the *basic level* category discussed in categorization theory. Basic level categories are the most natural level at which one describes something – the distinctions considered most generally useful, e.g. “there’s a dog on the porch” rather than “there’s a mammal on the porch” or “There’s a wire-haired terrier on the porch.” [12]. For our beginning students, distinctions of data type or control are not natural. Most responses in our study suggest the student’s model for instructions are not the same as the model we actually use in programming.

We can exploit the students’ focus on a problem by using it to have a concrete discussion about the model of control and data structures used in programming. Starting with an example like these tasks, students can be brought into a discussion of the actual control and storage structure model used for programming, addressing issues like why grouping data is difficult, why moving data actually involves making copies, and the granularity of a value.

## 6. CONCLUSIONS AND FUTURE WORK

This study of beginning computer science students suggests students bring significant skill in describing algorithms to their first course though their model of computation is not the same as the model they will be using in their computer science career. We believe these natural skills can be leveraged as we teach our students.

While this study has concentrated on a very algorithmic task, we continue to investigate other natural resources students bring to computer science. In particular, we are interested in natural abilities in areas such as debugging, human-computer interfaces, concurrency, data representation, conditional expression, searching, discrete probability, requirements elicitation, modeling, separation of concerns, and abstraction.

## 7. REFERENCES

- [1] M. Ben-Ari. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45–73, 2001.
- [2] Y. Ben-David Kolikant. Gardeners and cinema tickets: High schools’ preconceptions of concurrency. *Computer Science Education*, 11(3):221–245, 2001.
- [3] Y. Ben-David Kolikant. Students’ alternative standards for correctness. In *ICER-05*, pages 37–43, Seattle, WA, October 2005.
- [4] J. Bonar and E. Soloway. Preprogramming knowledge: A major source of misconceptions in novice programmers. In E. Soloway and J. Spohrer, editors, *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.
- [5] J. D. Bransford, A. L. Brown, and R. R. Cocking, editors. *How People Learn: Brain, Mind, Experience, and School*. National Academy Press, Washington, DC, expanded edition, 2000.
- [6] J. Bruner. *The process of education*. Harvard University Press, Cambridge, MA, 1960.
- [7] M. Clancy. Misconceptions and attitudes that interfere with learning to program. In S. Fincher and M. Petre, editors, *Computer Science Education Research*. Taylor and Francis Group, London, 2004.
- [8] Committee on Undergraduate Science Education. *Science Teaching Reconsidered: A Handbook*. National Academy Press, Washington, DC, 1997.
- [9] J. P. Gibson and J. O’Kelly. Software engineering as a model of understanding for learning and problem solving. In *ICER-05*, pages 87–97, 2005.
- [10] B. Glaser and A. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago, 1967.
- [11] D. Hammer. Student resources for learning introductory physics. *Physics Education Research, American J. Physics Supplement*, 68(7):S52–S59, July 2000.
- [12] G. Lakoff. *Women, Fire, and Dangerous Things*. University of Chicago Press, Chicago, IL 60637, 1987.
- [13] R. Lister, E. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. Mostrom, K. Sanders, O. Seppala, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, 36(4):119–150, December 2004.
- [14] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.*, 33(4):125–180, 2001.
- [15] L. Miller. Natural language programming: Styles, strategies, and contrasts. *IBM Systems J.*, 20(2):184–215, 1981.
- [16] L. Onorato and R. Schvaneveldt. Programmer/nonprogrammer differences in specifying procedures to people and computers. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, chapter 9, pages 128–137. 1986.
- [17] A. Schwill. Fundamental ideas of computer science. *Bull. European Association for Theoretical Computer Science*, 53:274–295, 1994.
- [18] B. Simon, T.-Y. Chen, G. Lewandowski, R. McCartney, and K. Sanders. Commonsense computing: What students know before we teach (episode 1: Sorting). In *ICER-06*, Canterbury, UK, September 2006.
- [19] J. Smith, A. diSessa, and J. Roschelle. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *Journal of the Learning Sciences*, 3(2):115–163, 1993.