Misconceptions and Concept Inventory Questions for Binary Search Trees and Hash Tables

Kuba Karpierz
University of British Columbia, CS
Vancouver, BC, Canada
mr.karpierz@gmail.com

Steven A. Wolfman
University of British Columbia CS
Vancouver, BC, Canada
wolf@cs.ubc.ca

ABSTRACT

In this paper, we triangulate evidence for five misconceptions concerning binary search trees and hash tables. In addition, we design and validate multiple-choice concept inventory questions to measure the prevalence of four of these misconceptions. We support our conclusions with quantitative analysis of grade data and closed-ended problems, and qualitative analysis of interview data and open-ended problems. Instructors and researchers can inexpensively measure the impact of pedagogical changes on these misconceptions by using these questions in a larger concept inventory.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer Science Education

General Terms

Human Factors

Keywords

data structures, misconceptions, concept inventory

1. INTRODUCTION

Concept inventories (CIs)—inspired by the Force Concept Inventory (FCI) [7]—have changed pedagogy across many disciplines [6, pp. 20–43]. CIs are "cheap" but powerful tools for probing student misconceptions and the longitudinal impact of pedagogical changes [5].

In this paper, we present several novel data structures CI questions and the research process that led to their design. We focus on Binary Search Trees (BSTs) and Hash Tables—important data structures course topics [11, 12] that were also identified as important during interviews with faculty at our institution. Critically, we *triangulate* evidence for misconceptions and CI question design from a broad data set using quantitative *and* qualitative methods.

SIGCSE '14 Atlanta, Georgia USA

CIs exist for some computing subjects, e.g., digital logic [6] and intro programming [13]. For data structures, researchers have identified misconceptions [4, 9] and developed CI questions targeting misconceptions [3, 10]. Our work contributes both results—new misconceptions and questions—and a thorough account of our approach.

2. METHODS

Our process adapts from Adams and Wieman [1] and Almstrum *et al.* [2]. Focusing on three courses—our data structures course and its pre- and post-requisites—we (1) identified key concepts from instructor interviews and aggregate analysis of course artifacts, (2) identified misconceptions from student interviews and deeper analysis of artifacts, and (3) formulated, piloted, and validated multiple-choice CI questions. Specifically, our data sources include:

Instructor Interviews: interviews with 9 instructors of the three courses, focused on sample exam responses and "when students don't get something important, which an expert *should* get."

Exam Statistics: per-student marks on 200+ exam problems taken by 1000+ students from 15 offerings of the courses and reviewed for common yet difficult problems.

Exam Analysis: open coding—qualitative analysis with iteratively developed codes—of responses to a final exam problem from data structures offerings in Jan–Apr and Jul–Aug 2012 (all 67 exams from one section of the former and all 59 exams from the latter). On the Aug exam, this problem was the third most difficult in our **Exam Statistics**. (The two harder problems' topics are outside this paper's scope.) The problem's format echoed a project given in Jan–Apr but *not* in Jul–Aug [8].

Project Analysis: open coding of reports from the project in **Exam Analysis** (all 68 reports from one section of data structures in Jan–Apr 2011 and all 87 from Jan–Apr 2012).

Think-Aloud Interviews: 25 one-hour student interviews, drawing problems from the data sources above. Students gave informed consent and received $15 for participation. We helped them learn to think aloud with two practice problems but subsequently interjected minimally. We audio-recorded, partially transcribed, and open coded interviews. We ran 4 further interviews using multiple-choice CI questions. These aimed particularly to validate that students select answers for our intended reasons.

We achieved a balanced sample of interview subjects across course level and grades. We classify students by the highest of the three courses in progress or completed as "**CS1xx**" (first-year discrete math), "**CS2xx**" (second-year data structures), and "**CS3xx**" (third-year algorithms). 9 were **CS1xx**, 10 were **CS2xx**, and 10 were **CS3xx**. Subjects covered a wide grade range. The **CS3xx** group earned 4 As, 2 Bs, 2 Cs, and 2 Ds in **CS3xx**. The **CS2xx** and **CS3xx** groups earned 9 As, 4 Bs, and 7 Cs in **CS2xx**. The **CS2xx** and **CS3xx** persistors skew **CS1xx** grades high, but the **CS1xx** group alone earned 3 As, 4 Bs, and 2 Cs. However, 59% of interviewees were women, over-representing their the roughly 24% women in our major.

CI Pilot: an optional quiz of draft multiple-choice CI questions given on the last day of the May–Jun 2013 data structures offering. All attending students chose to participate (77 of 98 enrolled, plus one auditor).

3. MISCONCEPTIONS

In this section, we report on three categories of misconceptions. We begin by identifying the misconceptions, work through their elaboration in the data, and conclude with design and initial validation of targeted CI questions.

3.1 Duplicates in BSTs

We found that students struggled the possibility of duplicates in BSTs—keys already present in the structure—through **Exam Analysis** and explored further with other data sources. Through iterative coding, we identified two major, recurring misconceptions:

- **SeparateOperation:** The misconception that, when inserting a potentially duplicate key, the BST must perform two separate find operations: one to ensure the key isn't already present, and one to find where to insert it.
- **FullSearch:** The misconception that, when inserting a potentially duplicate key, the BST must inspect *every* key in the tree.

Although we document both misconceptions, we focus on the more prevalent **SeparateOperation**.

3.1.1 Identifying the Misconceptions

The problem in **Exam Analysis** was not designed to focus on duplicates. However, of the 126 exams analyzed, 8 exhibited duplicate-related misconceptions:

- Four students erroneously attribute worst-case BST performance to duplicates, but with little explanation. For example, one reasoned "since we are not inserting any duplicates we see normal $\lg n$ behavior."
- Two students hinted at **SeparateOperation**. One wrote that a BST "would run in [worst-case] linear time because it has to check for duplicates on every insert." The other states that "[a BST] should be $O(\lg n)$ time for insertion [as it] must check for duplicates when inserting."
- The final two students clearly articulate **SeparateOperation**. One wrote that "[the BST] search for duplicate first (lgn) and insert if not there (lgn)." The other stated that a "BST shows $O(n)$ behavior in [worst-case insertion] because it has to find a duplicate before inserting."

```
When we insert a key into a binary search tree (BST), the key may be
a duplicate: a key that is already present somewhere in the tree.

Imagine a BST implementation that assumes that no duplicate key is
ever inserted and that is used in a way that guarantees this
assumption is correct.  Which of these best describes this
assumption's effect on insertion of new keys?

a. We no longer need to search for duplicates, and the rest of the
   work is very fast (takes a constant number of steps).

b. We still need search for where to add the key, but we no longer
   need to test as we go whether the key is equal to each other key
   we access during the search.

c. We can now do only one rather than two searches in the tree: we
   still search for where to add the key, but we do not search for
   duplicates.

d. There is not enough information to tell.
```

Figure 1: CI question targeting SeparateOperation. This version is from the first interview. We reworded after each interview and the CI Pilot.

We had yet to identify **FullSearch**, but, in retrospect, this misconception would account for three students' beliefs that duplicates *caused* linear asymptotic bounds.

We next turned to the richer data in the **Project Analysis**. Unfortunately, few reports mention duplicates. However, three did offer insight into duplicate misconceptions.

One report demonstrates **SeparateOperation**, stating for an AVL Tree that "[reinserting previously deleted elements] is the same as inserting the elements for the first time due to the lack of tombstones, and therefore involves searching the tree for duplicates."

One other report carefully detailed **FullSearch** with an example: "when we inserted ... a middle value and alternating smaller and larger values ... 50, 49, 51, 48, 52, ... we were making two singly linked list[s] ... Each insertion would supposedly take half as much time as insertion using an ascending list, but since the ADT would have to check for duplicates in the other branch, it took a longer time." Another showed **FullSearch** in the context of vectors, not BSTs: "This corresponds with a sorted vector: ... random numbers are inserted in linear time due to checking the entire list for many values for duplicates."

Our first 25 **Think-Aloud Interviews** posed no problems aimed at these misconceptions. Yet, two students still exhibited **FullSearch** on this problem about a small BST: "In the worst case, how many nodes would we need to look at to find a key in this tree? Which ones?" One was halfway through data structures, the other finishing its postrequisite. The former said, "[in the worst case, we] need to look for 3 ... That means we need to go through everything else." The latter student said "you would need to look at all of the nodes above the bottom layer to find a specific key", and gave a formula for the search cost: $\approx 1/2 \cdot 2^{\text{depth}}$.

3.1.2 Concept Inventory Question Design

We then designed a multiple-choice CI question targeting **SeparateOperation**, shown in Figure 1.

We put (a)–(c) in order by the number of searches they describe (0, 1, and 2). (a) targets a misconception where students attribute no search cost to insertion operations. Since we found this misconception *only* in **Project Analysis**, we discuss it no further. (b) is the intended correct answer, envisioning removing an "equals" case and the resolution it performs for duplicates. (c) targets **SeparateOperation**.

```
When we insert a key into a binary search tree (BST), the key may be
a duplicate: a key that is already present somewhere in the tree.

Imagine a BST implementation that assumes (as a precondition) that
no duplicate key is ever inserted. Which of these best describes
this assumption's effect on insertion of new keys compared to a
normal BST implementation?

a. We no longer search the tree at all.  We just add the key
   directly where it belongs.

b. We search the tree once either way.  We still search for where to
   add the key, but we no longer check for duplicates as we go.

c. We search the tree once rather than twice.  We still search for
   where to add the key, but we no longer search first for
   duplicates.
```

Figure 2: Final revision of CI question targeted at SeparateOperation.

Answer (d) was only present to camouflage for other questions where it was correct (e.g., see Section 3.2.3). After the first interview, we changed it to "This assumption has no effect." Either way, as indicated below, (d) exposed our early wording of (b) as a weak fit for students with a clear understanding of the problem.

In the **CI Pilot**, 44.9% correctly answered (b), but the bulk of the 24.4% answering (d) may have had a correct model of BST insertion. 24.4% selected (c), suggesting they believe **SeparateOperation**.

We asked this question of three students in **Think-Aloud Interviews**. Two gave correct reasoning. One was concerned about not "search[ing] *through* the tree *at all*" on (a) and that "you wouldn't do two searches, you would just continue on" on (c). The student agreed with (b) and preliminarily circled it, saying "That's true. You can stop when you find it." However, the student switched to (d), concluding "I'm feeling like [the absence of duplicates] really actually doesn't have an effect in a BST. . . " The other student quickly concluded "you wouldn't really need to search for [a duplicate] anyways because it would be where you're inserting [the new key]" and then selected (d).

The third student illustrated **SeparateOperation** and the need for carefully crafted CI questions. While reading (a) but before reading (c), the student said, ". . . if there were duplicates, you're going to need an extra step to basically search through the tree to see if the same key exists somewhere else. So that would pretty much double our work." The student's final selection was based on how "vague" or "explicit" the answers were. (a) "[is] a little ambiguous" and "doesn't mention anything about adding a key to the tree". Compared to answer (c), answer (b) has "a very nice way of saying 'but we do not search for duplicates' . . . [and] a nice way usually involves being more explicit." Thus, the student selected (b) "despite it having the longest answer".

Given (d)'s appeal to students whose reasoning matches (b) and the accidental lack of parallel structure exploited by the last student, we rephrased to eliminate (d) and normalize the other answers, as shown in Figure 2.

Based on our data, we believe this question effectively targets **SeparateOperation**, and that a substantial minority of our students hold this misconception. We believe **FullSearch** is less prevalent but worth exploring further.

3.2 Conflation of Heaps and BSTs

We explored misconceptions related to BSTs and binary heaps because of previous work suggesting some students believe that a heap is also a BST [3, 10]. We attempted to replicate Danielsiek *et al.*'s results on their multiple-choice CI question targeted at heap/BST misconceptions [3]. We later investigated these misconceptions *unintentionally* in **Think-Aloud Interviews**. Our results do suggest students conflate these data structures, but *not* in response to the initial multiple-choice question. Specifically, we found evidence of the following recurring misconception:

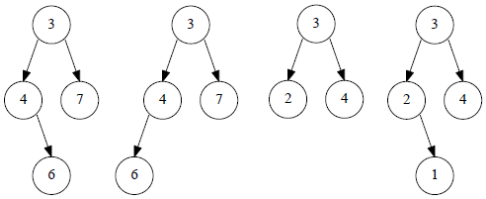

Figure 3: Our rendering of the trees from the heap/BST question [3]. The trees were shown in a column, each prefaced by the options: binary min-heap (only), BST (only), both, or neither.

- **DefaultBalanced:** The misconception that, absent concrete information, a BST tends to have a balanced shape, particularly left-complete (binary heap-shaped).

3.2.1 Initial Multiple-Choice Results

We offered the question in Figure 3 on the Apr 2012 data structures final exam Paul and Vahrenhold highlight two misconceptions the question exposes: (1) overlooking left-completeness, with most respondents indicating that the first tree is a heap, and (2) conflating heaps and BSTs, with 15.6% indicating the third tree is both heap and BST while correctly answering the remaining parts [10].

Consistent with previous work, most students overlooked the left-completeness requirement. Of 67 students, 59.7% indicated that the tree was a binary heap. Only 34.3% correctly answered that the tree was neither heap nor BST. 17.9% believed this tree was a BST, suggesting they checked the order invariant only between nodes and their immediate children, a misconception identified by Fekete [4]. Although the quantitative results support the presence of both misconceptions, we were unable to gather triangulating evidence and so address them no further here.

Surprisingly, we found little evidence that students conflated BSTs and heaps. Of the 20 students who answered all three other parts correctly, only 5% (one) believed the third tree was both BST and heap. The rest correctly identified it as only a BST. Overall, 92.5% of students selected only a BST, while just 6% believed it was *also* a heap and none believed it was *only* a heap. The vast majority also correctly responded to the second and fourth prompts.

We therefore concluded preliminarily that this heap/BST conflation did not occur in our student population.

3.2.2 BST Think-Alouds

We posed five problems about BSTs—none mentioning heaps—in **Think-Aloud Interviews**. 16 subjects solved at least one of these problems; of these, 11 had completed at least two-thirds of the data structures course.[1] Of these 11,

[1]The two other students who had at least started data structures both showed evidence of confusing BSTs and heaps.

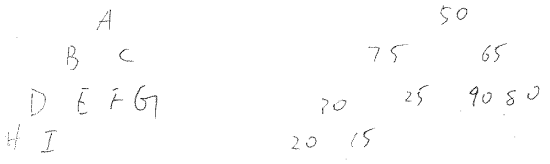

Figure 4: A student's heap-shaped guide and final result on the BST post-order traversal problem.

five showed clear evidence of confusing heaps and BSTs. Often, when uncertain about the shape of a BST, they reached for a balanced, left-complete tree—a binary heap-shaped tree—as a sort of "default" shape for a BST, leading us to postulate **DefaultBalanced** as their misconception.

The two most striking examples responded to the prompt: "Draw a binary search tree whose keys printed in post-order traversal are: 20 15 30 25 75 90 80 65 50". Both sketched empty, heap-shaped trees to get started. Figure 4 shows one student's sketches. This student then performed a post-order traversal on the sketch and lined the result up under the given numbers to solve the problem. The other student used the rule that the root is always last to line up keys with nodes. Both students noticed that what they had created was not a BST. One gave up: "Well, this is not a binary tree ... It's a nothing. I'm done." The other reached a nearly-correct answer by patching a series of problems, such as: "There's four [keys larger than 50], I've only drawn three [nodes right of the root]. So that's a serious problem." One such moment strongly suggests **DefaultBalanced**: "Well, let's just assume we have a properly balanced search tree in search of part marks."

Two of the remaining students facing other BST problems fell back on heap shape properties. Asked why a small binary tree was not a BST, one explained "it's not a binary search tree because it's not a complete and full tree." The other simulated a sequence of insertions in decreasing order into a "standard BST", saying "I think the standard [BST] means ... a complete tree. ... [9999] can stay on the bottom for now, but [9998] is going to have to go and complete the tree. ... When we get to the next full complete bottom, there will be some numbers there that won't work; so, we have to rearrange again."

3.2.3 Concept Inventory Question Design

We then designed a multiple-choice CI question targeting **DefaultBalanced**, as shown in Figure 5.

We ordered answers randomly, but ensured that (b) and (c) appeared together and (e) remained last. (e) is correct. (b) and (c) are targeted at **DefaultBalanced**. (d) is targeted at students who do not read the question carefully. After this answer proved very attractive in our **Pilot CI**, we added the parenthetical "...not necessarily ... in that order" but have not yet piloted or validated this phrase. (a) targets students who do not read carefully and hold a misconception noted occasionally in **Think-Aloud Interviews** where students insert keys at the top of the tree rather than the bottom. Due to a lack of triangulating evidence, we do not discuss this misconception further.

In the **CI Pilot**, 42.3% of students chose the correct response. 20.5% chose (d) and so perhaps did not read carefully. However, 35.9% chose (b) or (c)—most choosing (b)—which suggests they believe **DefaultBalanced**.

```
What shape is a binary search tree that contains the keys 1, 2, 3,
4, 5, 6, and 7?  (Keys were not necessarily inserted in that order.)

a. Exactly this shape:                          7
                                                6
                                               5
                                              4
                                             3
                                            2
                                           1

b. Exactly this shape:                         4
                                             2     6
                                           1 3   5 7

c. This shape with either 1 or 7 at           *
   the root and the other keys              *     *
   arranged appropriately:                 * *   * *


d. Exactly this shape:                     1
                                            2
                                             3
                                              4
                                               5
                                                6
                                                 7

e. There is not enough information to tell.
```

Figure 5: Final version of CI question targeted at DefaultBalanced.

Think-Aloud Interviews with three students added evidence for **DefaultBalanced** and for the question's validity.

One student did not exhibit the misconception and picked the correct answer for the right reasons. This student began by saying, "Which order were [the keys] inserted? I'm not sure." The student then quickly eliminated each distractor with comments like "We don't know that."

Two students exhibited **DefaultBalanced** and picked the balanced BST distractor because of it. One eliminated the non-BST distractor, saying "I just want to pick [the balanced BST] because it looks the nicest." The student discarded the other distractors with a skeptical "If it's not *technically* balanced, then I guess [the two list-shaped trees] would technically uphold the properties." The other student also selected the balanced BST as a matter of taste, e.g., discarding a distractor with: "I don't like [this list-shaped tree] because that seems like how I would insert keys into a BST back when I didn't know how BSTs worked." The deciding factor was that "[the problem] doesn't exactly tell you the order items inserted. So, I'm going to assume they're looking for the perfect BST that contains said keys."

Our research provides strong evidence that our CI question effectively targets **DefaultBalanced** and that the misconception's prevalence merits broader investigation.

3.3 Hash Table Resizing

As with the duplicate issues, we observed hash table resizing misconceptions in **Exam Analysis** and explored further through other data sources. We found substantial evidence for two misconceptions:

- **NoRehash:** The misconception that, when resizing, the hash table leaves keys at their existing indexes.
- **ResizeInPlace:** The misconception that, when resizing, the table is extended *in place* by appending slots.

3.3.1 Identifying the Misconceptions

In the **Exam Analysis** problem, students worked with a hash table using a simple hash function and probing strat-

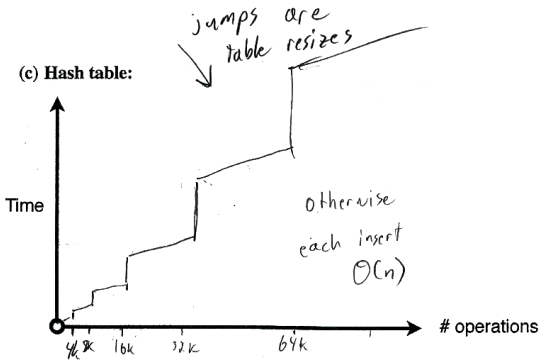

Figure 6: A student's correct response to the hash table insertion problem.

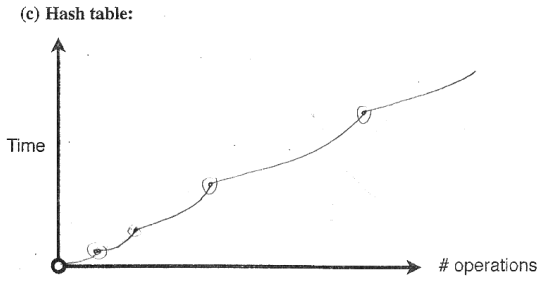

Figure 7: A student's "scallop-shaped" response to the hash table insertion problem.

egy. They graphed its performance on a contiguous, decreasing sequence of key insertions. Figure 6 shows a correct answer with linear regions indicating swift performance without collisions and steep jumps indicating resizes.

Two incorrect answers occurred frequently. Figure 7 shows the combined result of these errors: a "scallop-shaped" graph where each point of the scallop shows a transition from rising per-operation insertion cost back to cheap costs after resizing, but does not show an upward jump in total cost of insertions. 22% of students indicated that collisions slowed down the insertion process as the insertion sequence progressed toward resizes. 16% recognized that resizes occurred yet indicated that they had essentially zero cost.

The prompt did not lead many students to explain these features of their answers, but we found evidence that led us to create the **ResizeInPlace** misconception. These students suggested For example, one student describes the resize as "vector resizing", which does not require individual reinsertion. Another blames increased collisions late in the sequence on "smaller numbers [having] their hash collide with the bigger numbers that was hashed before with a smaller table size."

Isolating the misconception(s) leading to increasingly expensive collision cost proved difficult. Instead, we simply noted common explanations. Two were applications of "book knowledge", statements that are good general guidelines but ignore the details available in the problem: linear probing does a poor job of spreading keys through the hash table, and insertion costs go up with load factor. The last, mentioned in a quote above, led us to the **NoRehash** misconception: the idea that collisions occur between newly inserted keys and keys "... hashed before with a smaller table size."

```
Which of these best describes the process of resizing a hash table to
increase its size?

a. Create a new, separate table larger than the old table, and
   copy the keys from the old table to corresponding slots in the new one.

b. Create a new, separate table larger than the old table, and
   individually re-insert each key from the old table into the new one.

c. Add extra slots at the end of the existing hash table, and
   leave the old keys in place in their existing slots.

d. Add extra slots at the end of the existing hash table, and
   individually re-insert each key from the old table into the new one.
```

Figure 8: Final version of CI question targeted at ResizeInPlace and NoRehash.

3.3.2 *Elaborating the Misconceptions*

To learn more, we added a simplified version of this problem late in our **Think-Aloud Interview** process with four students: one just finishing the data structures course and three in its post-requisite. Three of these gave insight into the misconceptions.

One student exhibited **NoRehash** and **ResizeInPlace** in a connected fashion: "When we say the table ... doubles in size, are we literally just like adding an extra bunch of elements at the end or are we rehashing everything? ... I'm going to assume that we're not rehashing everything." For this student, rehashing is a way to move keys into a new memory block, not a necessary adjustment. The student later describes keys hashing into locations still occupied by "old" keys that should have been rehashed.

The next student shows evidence of **NoRehash**, seeing allocation as the dominant cost of resizing: "The [delays for resizes] get gradually bigger ... because it needs to take more time to make space for the hash table." Later that student shows evidence of **ResizeInPlace**, saying: "with ... linear probing ... you wouldn't have to rehash, ... it resizes at half full, so it would be like the first half, almost half of it is full and then you would have to move down to the n-over-2 spot."

The last student *may* believe **NoRehash**. This student blames slow resizes on copying costs: "When the size of the array reaches a certain number ... the performance becomes really ugly because you have to copy all elements [to the new table]." Unfortunately, the student's analysis then ended with by summarizing that total insertion time would "go up a little, cause assume that the worst case everything hashes to zero when you have to look for the next element."

3.3.3 *Concept Inventory Question Design*

Based on these students' discussion of the problem, we designed a much simplified multiple-choice CI question that asks directly about hash table resizing, as shown in Figure 8.

We chose randomly whether the "create a new" or "add extra slots" answers should go first and, within each group, whether the "copy" or "re-insert" answers should go first. (b) is correct. Together, the answer set targets all possible combinations of **NoRehash**—(a) and (c)—and **ResizeInPlace**—(c) and (d). The answers include phrases from student responses and use line-breaks and consistent wording to draw attention to the differences among answers.

In the **CI Pilot**, 62.8% of students chose the correct response. 33.3% chose (a) or (c), suggesting they believe **NoRehash**. **ResizeInPlace** is rarer in our data, with only 11.5% choosing (c) or (d). The **CI Pilot** version of the

question said "... copy the keys from the old table into the new one" in (a). We altered the wording for the final version shown above based on the **Think-Aloud Interviews** described below.

We conducted **Think-Aloud Interviews** with three students over this question, which support both the presence of both **ResizeInPlace** and **NoRehash**.

One student answered correctly with correct reasoning: "[Creating a new table and individually re-inserting] does seem like a better way, cause that way you would keep constant your hashing rule ... and all your collisions will follow your rules that you set."

However, this student did show some evidence of **ResizeInPlace**, e.g., saying of one "add extra slots" answer "that could work," and "[it] seems like a bad way of increasing its size cause [it would cause unpredictable collisions]". The student saw problems with not re-inserting keys and with re-inserting some in place while others remain in the table, but did *not* recognize that resizing an array in place is, in general, infeasible. This suggests some students who believe **ResizeInPlace** *will* be able to eliminate its distractors, which may explain the lower "yield" of students with this misconception in our **CI Pilot**. We do not yet have a more attractive rewording for these students; so, instructors and researchers interested in array resizing misconceptions may want to use this CI question as a conservative estimate of **ResizeInPlace** and develop more tightly targeted items.

Another student showed clear evidence of **ResizeInPlace** and then selected an "add extra slots" distractor. This student drew a diagram of a hash table resizing by adding slots to the end *after* reading the prompt but *before* reading the answers. The student then worked through the answers, eliminating the "re-inserting" ones because "it just seems really cost-heavy to individually re-insert anything," which is consistent with **NoRehash** and selecting the distractor targeted at both misconceptions.

However, the problem version given to this student included a **NoRehash** distractor that said to "copy the contents of the old table into the new one." This wording led the student to worry about copying pointers versus copying the referenced data. We therefore rephrased: first replacing "contents of" with "keys from" and, after the **CI Pilot** and our final interview, to the final version shown above.

That final student clearly displayed **ResizeInPlace** and **NoRehash** and selected the answer targeted at both. Consistent with **NoRehash**, the student eliminated the "re-insert" answers because "[re-inserting] would take forever" and "... if you're just adding extra slots ... [you wouldn't need to] re-insert things." The student had trouble selecting between the "copy the keys" and "leave the old keys in place" distractors but eventually chose the latter. The interviewer made a rare interjection to ask why, and the student gave reasoning consistent with **ResizeInPlace**, saying: "It just didn't really make sense to me to create a new table instead of just increasing the size of the one you already have."

Our quantitative and qualitative data together suggest this CI question is effective in probing **NoRehash** and, perhaps conservatively, probing **ResizeInPlace**. Each is present in a substantial minority of our students.

4. CONCLUSIONS AND FUTURE WORK

We presented important new misconceptions about hash tables and binary search trees. We also created three easily reusable concept inventory questions, and ensured that they assess the intended misconceptions. We demonstrated that students with data structures background harbor these misconceptions, in some cases in much greater numbers than indicated on the exam questions we analyzed.

In future work, we plan to verify the CI questions' applicability at other institutions. Other misconceptions that arose during our study also merit further research, particularly **FullSearch**, for which lack a satisfactory CI question.

In the long run, with our CI questions and others', we hope to produce a modest-sized inventory that may not cover the breadth of data structures, but—as with the tightly-focused Force Concept Inventory [7]—still provides a valuable bomb calorimeter[2] to measure student understanding.

5. ACKNOWLEDGMENTS

Thanks to many students and faculty who consented to participate in these studies and to colleagues who advised us on our work. Special thanks to Allison Elliott Tew, Kimberly Voll, and Elizabeth Patitsas for feedback and guidance. This work was funded by the Carl Wieman Science Education Initiative at the University of British Columbia.

6. REFERENCES

[1] W. K. Adams and C. E. Wieman. Development and validation of instruments to measure learning of expert-like thinking. *Int'l. J. of Science Ed.*, 33(9):1289–1312, 2011.

[2] V. L. Almstrum, P. B. Henderson, V. Harvey, C. Heeren, W. Marion, C. Riedesel, L.-K. Soh, and A. E. Tew. Concept inventories in computer science for the topic discrete mathematics. In *Working Group Reports on Innov. and Tech. in CS Ed.*, pages 132–145, 2006.

[3] H. Danielsiek, W. Paul, and J. Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proc. of the Tech. Symp. on CS Ed.*, pages 21–26, 2012.

[4] A. Fekete. Using counter-examples in the data structures course. In *Proc. of the Australasian Conf. on Comp. Ed.*, pages 179–186, 2003.

[5] R. R. Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1):64–74, 1998.

[6] G. L. Herman. *The development of a digital logic concept inventory.* PhD thesis, UIUC, 2011.

[7] D. Hestenes, M. Wells, and G. Swackhamer. Force concept inventory. *The Physics Teacher*, 30(3):141–158, March 1992.

[8] K. Karpierz, J. Kitching, B. Shillingford, E. Patitsas, and S. A. Wolfman. "Dictionary Wars": an inverted, leaderboard-driven project for learning dictionary data structures. In *Proc. of the Tech. Symp. on CS Ed.*, pages 740–740, 2013.

[9] E. Patitsas, M. Craig, and S. Easterbrook. On the countably many misconceptions about #hashtables. In *Proc. of the Tech. Symp. on CS Ed.*, page 739, 2013.

[10] W. Paul and J. Vahrenhold. Hunting high and low: instruments to detect misconceptions related to

[2]Or maybe thermometer.

algorithms and data structures. In *Proc. of the Tech. Symp. on CS Ed.*, pages 29–34, 2013.

[11] B. Simon, M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders. Making sense of data structures exams. In *Proc. of the Int'l. Wkshp. on Comp. Ed. Research*, pages 97–106, 2010.

[12] J. Tenenberg and L. Murphy. Knowing what I know: An investigation of undergraduate knowledge and self-knowledge of data structures. *CS Ed.*, 15(4):297–315, 2005.

[13] A. E. Tew and M. Guzdial. Developing a validated assessment of fundamental CS1 concepts. In *Proc. of the Tech. Symp. on CS Ed.*, pages 97–101, 2010.