Developing a Formative Assessment of Instruction for the Foundations of Computing Stream

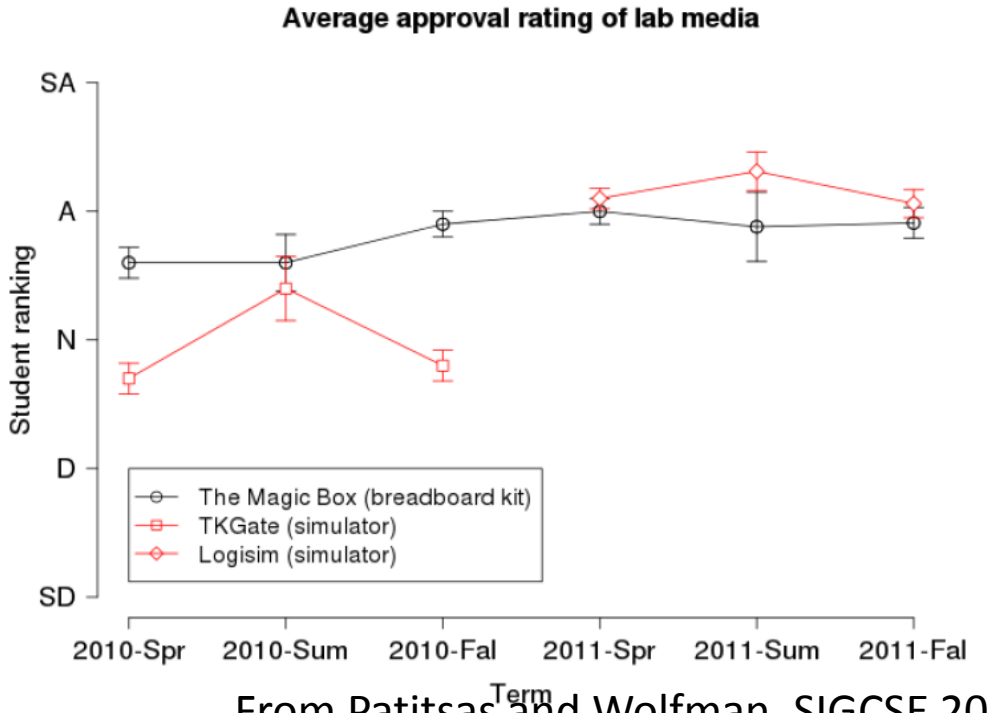
2013/04/08

Steve Wolfman

UBC CS

Acknowledgments: Help, data, and input from *many* faculty and students at UBC.

Inspiration



From Patitsas and Wolfman, SIGCSE 2012

From Mazur, Int'l Newsletter on Physics Ed, Apr 1996

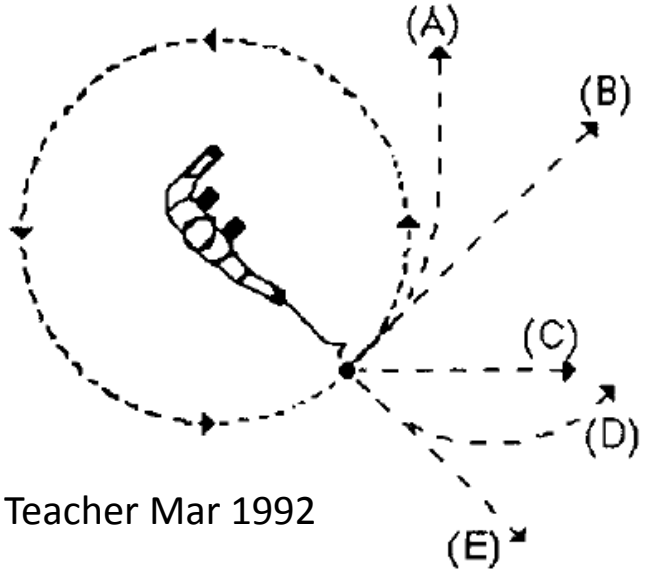
But about a year ago, I came across a series of articles by David Hestenes of Arizona State University(1) that completely and permanently changed my views on teaching. In these articles, Hestenes shows that students enter their first physics course possessing strong beliefs and intuitions about common physical phenomena. These notions are derived from personal experiences and color students' interpretations of material presented in the introductory course. Instruction does very little to change these "common-sense" beliefs.

Ideal Goal

Sustainable assessment as a "thermometer" for health of the courses.
What makes it sustainable? Let's look at the FCI:

- 29 multiple choice Qs and takes 23.3 minutes
- Founded on previous/ongoing physics ed. research
- Focuses on a few key topics
- Deliverable on paper (if needed!)

A heavy ball is attached to a string and swung in a circular path in a horizontal plane as illustrated in the diagram to the right. At the point indicated in the diagram, the string suddenly breaks at the ball. If these events were observed from directly above, indicate the path of the ball after the string breaks.



From Hestenes, Wells, and Swackhamer, Physics Teacher Mar 1992

Methodology

- Gather goals by interviewing faculty involved in the stream
- Augment & winnow goals by analysis of exam results
- Draft questions to assess key goals
- Validate expert responses to questions
- Collect student misconceptions in think-aloud interviews
- Formulate forced-answer versions of Qs based on student responses
- Validate forced-answer Qs in think-aloud interviews
- Pilot assessment, confirming reliability and validity
- General use for assessment of courses, longitudinal analysis, etc.

How do we assess these??

Grand Goals…

What are the key learning goals for the Foundations of Computing stream?

- Recursive/inductive thinking
- Analysis of resource (time, space, energy, …) costs of solutions
- Formalization/specification of ill-specified problems
- Comfort with “dense” formal descriptions
- Proposal and explanation of multiple solution approaches to a problem
- Meta-cognitive management of the solution process
- Generalizing/abstracting problems/sol’n properties

…to Assessable Goals (?)

"Think about times when you cringe inside because your students just don't get something that seems very important to you, and which you expect any expert to get."

+

Quick review of low-/mid-/high-scoring exams.

Induction (6/2/-1): "should be able to do .. themselves from scratch without requiring additional input"

Divide & Conquer / Recurrences / Dynamic Programming (4/0/-3): "express the solution to a problem in terms of subproblems"

Logarithmic Tree Height (3/0/-0): "How many times can I give away half my apples before being left with just one?"

…

Exam Analysis: Low Mean

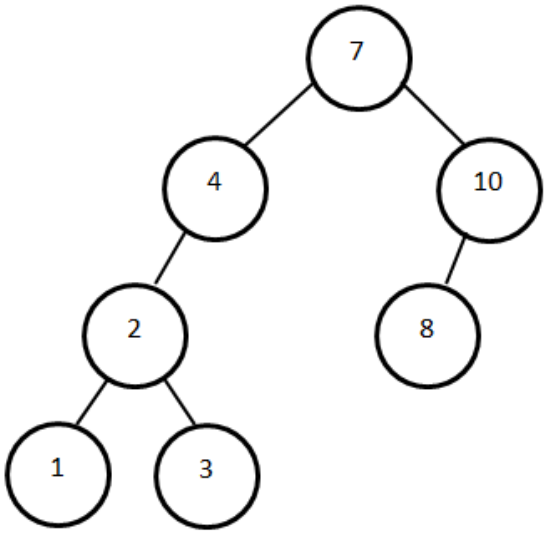
Unweighted Mean	Problem Mean	Problem Correlation	Adjusted Mean	Topic
0.732222744	0.199122807	0.391418802	-4.205337015	Heap/Asymptotic Analysis (2 variable analysis, find k-th smallest in n nu
0.705661765	0.32254902	0.374684015	-2.668872943	Asymptotic Analysis/ADTs ("dictionary dora" question, sketch behaviour
0.688011564	0.41087963	0.543997801	-2.203896611	Induction Proof (open-ended, strong, code-based (Racket-ish recursive p
0.617412418	0.303191489	0.572218428	-2.02942582	Heaps (complex problem troubleshooting)
0.705661765	0.464705882	0.50537874	-1.678567584	Combinatorics (order doesn't matter, replacement, "at least 10" limited t
0.732222744	0.546052632	0.520809062	-1.468595307	Functions (MC, cardinality, injection/bijection/surjection, pigeonhole)
0.6836933	0.476911977	0.79708724	-1.439631886	P vs. NP (show prob in P, show prob in NP, reduction from IS)
0.683545973	0.466569767	0.624281229	-1.383294823	Working Computer (newer, open-ended, somewhat similar to 2010W2.9
0.705661765	0.508235294	0.353694098	-1.375329253	Sorts/BSTs/Heaps (open-ended asymptotic analysis in interesting situati
0.700276696	0.494318182	0.646238294	-1.374351207	Memory management/Sorting algorithm (C++, space complexity)
0.653105503	0.458815029	0.56689642	-1.273761376	Predicate Logic Proof (open-ended, direct, EEA, big-O like)
0.659589036	0.476293103	0.450958138	-1.232959284	Sorting Algorithms (selection and quick comparison)

High Correlation

Unweighted Mean	Problem Mean	Problem Correlation	Adjusted Mean	Topic
0.595117845	0.433333333	0.811163781	-0.785871568	Induction Proof (strong, open-ended, dividing stack of a+b scores a*b bu
0.6836933	0.476911977	0.79708724	-1.439631886	P vs. NP (show prob in P, show prob in NP, reduction from IS)
0.595117845	0.569444444	0.796360036	-0.124709068	Functions (closed-ended MC-ish, pre-image/image/injective/surjective,
0.661328656	0.618253968	0.796029275	-0.234615294	Miscellaneous (MC)
0.688011564	0.548821549	0.791351467	-1.106911058	Working Computer/Seq'l Circuit (open-ended + MC, similar to parts of se
0.661328656	0.708333333	0.781922156	0.256020799	Functions (MC + brief justification, injective/bijective/surjective)
0.653105503	0.502023121	0.779417012	-0.990490672	Induction Proof (open-ended, strong, base case given, triangulation, ver
0.617412418	0.541843972	0.770617916	-0.488066015	Parallel Algorithms/ADTs (really about linked lists (no random access) an
0.628239728	0.439393939	0.7652708	-1.04917734	Induction Proof (strong, open-ended, dividing stack of a+b scores a*b bu
0.628239728	0.686363636	0.747386088	0.322921087	Functions Proof (open-ended, prove or disprove, somewhat similar to 20

Interview Analysis Examples

We often draw diagrams of binary search trees like this one:



We have shown the keys but not the values in this tree. Where are the values?

"the values are pointed to or stored in the same node object"

"The value [are] stored in these bubbles"

"the values would reside in memory or on disk"

"So the values are.. in the nodes"

"the values should be 1, 2, 3, 4, 5, so they're index values ... 6, 7, 8"

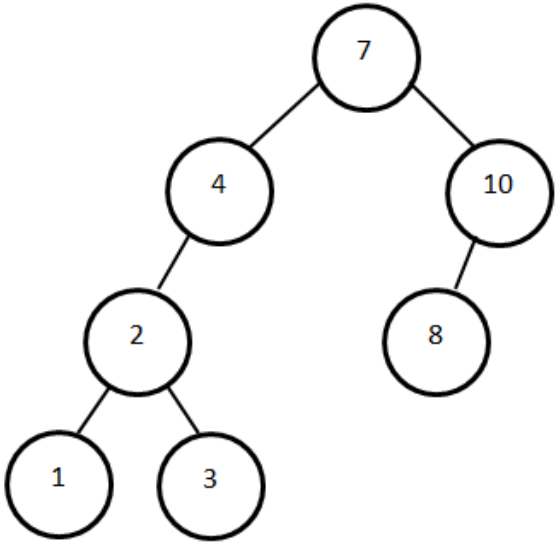
"... an array ... the keys could just be the indices of a giant array"

"[the key is] a lookup for the value"

"the values are being represented in the tree by the keys. So knowing ... the key like unlocks ... what the value is"

"the values would actually be in the leafs"

We often draw Binary Search Trees (BSTs) like this, showing the keys but not the values:



The keys in this BST are numbers;
[assume that the values are as well **OR**
assume that the values are *images***]**.

Where are the values in such a BST?
Choose the **best** answer.

(a) The values are stored in the same node as the keys.
(b) The values are at the leaves.
(c) The values are pointed to from the same node as the keys.
(d) The keys are indices into an array that stores the values.
(e) The keys point to the values.
(f) The values are represented in the tree by the keys.
(g) The values are 1 (for the node labeled 7), 2 (for the node labeled 4), 3 (for the node labeled 10), 4 (for the node labeled 2), and so on.
(h) Not enough information to tell.
(i) I don't know.

Imagine you were creating a dance. Here's a procedure you could use to describe the dance:

```
Dance(n):
  if n = 1:
    walk forward 1 meter,
    turn left (by 90 degrees)
  else:
    do the steps in Dance(n - 1),
    turn right (by 90 degrees),
    do the steps in Dance(n - 1).
```

Let $M(n)$ be the number of meters of walking you have to do in the dance. So, $M(1) = 1$.

Give a formula for $M(n)$ that is correct for all $n \geq 2$. (That is, for $n = 2, 3, 4$, and all larger values.) Your formula **can and should be in terms of $M(\,)$.**

Imagine you were creating a dance. Here's a procedure you could use to describe the dance:

```
Dance(n):
  if n = 1:
    walk forward 1 meter,
    turn left (by 90 degrees)
  else:
    do the steps in Dance(n - 1),
    turn right (by 90 degrees),
    do the steps in Dance(n - 1).
```

Let *M(n)* be the number of meters of walking you have to do in the dance. So, *M(1)* = 1.

Give a formula for *M(n)* that is correct for all $n \geq 2$. (That is, for $n = 2, 3, 4$, and all larger values.) Your formula **can and should be in terms of *M( )*.**

$$M(n) = 2M(n-1) + 1$$

"We first do one step, then M(n-1) steps..."

"...it would always be reduced to 1 ultimately. Since it's just Dance(n-1) and then it's just turning not really moving..."

$M(n) = (n(0)+1 \quad . \quad)$ for $n \geq 2$

Kahney, CHI 1983

```
SOLUTION-1:

TO INFECT /X/
1 NOTE /X/ HAS FLU
2 CHECK /X/ KISSES ?
2A If Present: INFECT * ; EXIT
2B If Absent:  EXIT
DONE
```

```
SOLUTION-2:

TO INFECT /X/
1  CHECK /X/ KISSES ?
1A If Present: INFECT * ; CONTINUE
1B If Absent:  CONTINUE
2  NOTE /X/ HAS FLU
DONE
```

paragraph. Under the Loop model, however, a programmer would argue that the first Solution would be okay, but not the second. In

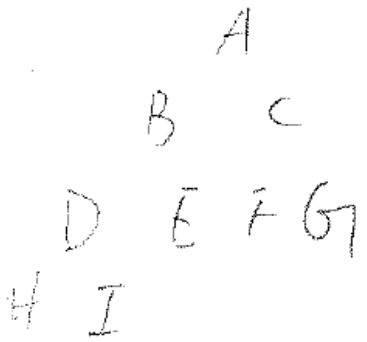
"Starting at 4, you.. Do 4, turn right, do 3, turn right, do 2, turn right, walk forward 1 m, turn left. … So only at one point will I do 1, that's when you walk forward. So M(n) = 1."

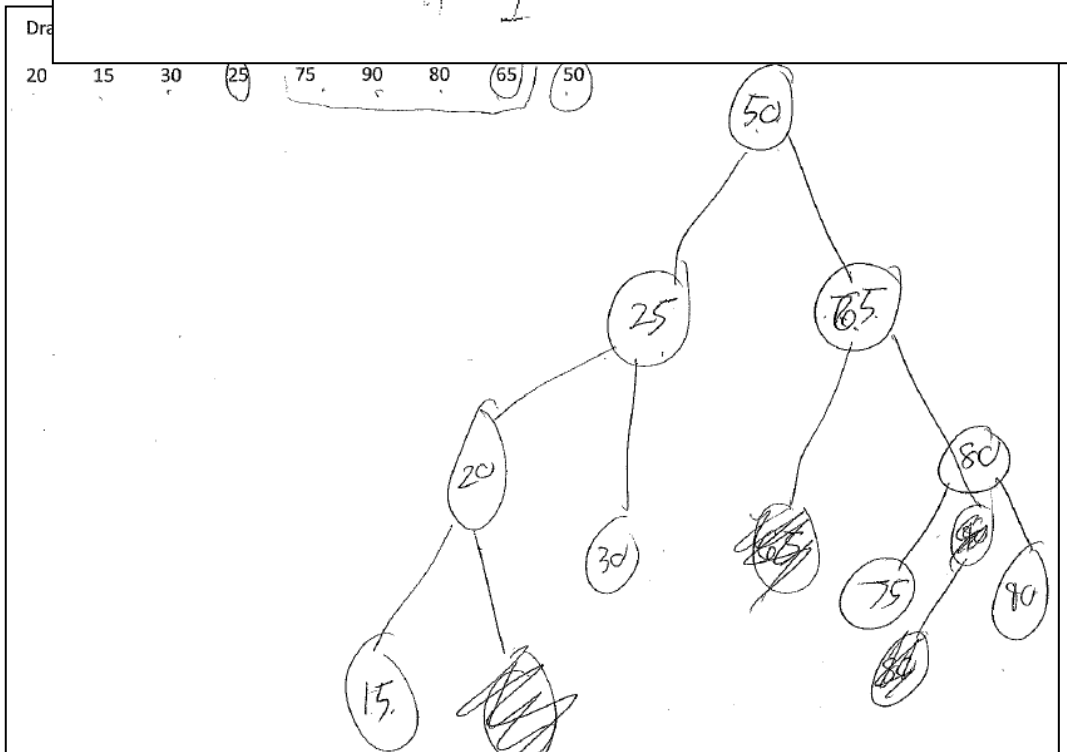
"If I'm correct, it would always be reduced to 1 ultimately. Since it's just Dance(n-1) and then it's just turning not really moving… and that wouldn't affect …. the number of metres that I've walked."

Heap/BST Confusion? Never hinted at by my faculty i'views…

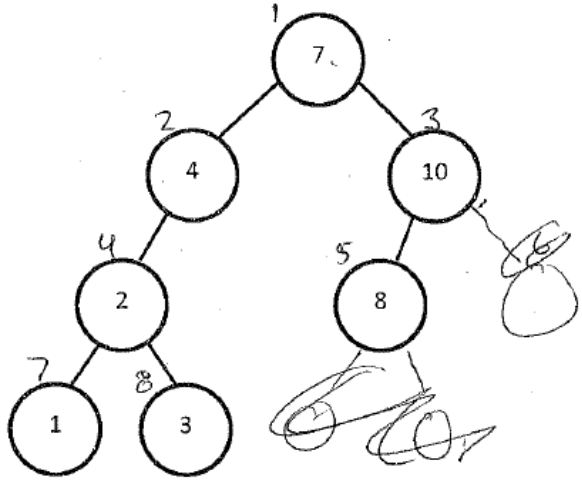
Draw a binary search tree whose keys printed in post-order traversal are:

20 15 30 25 75 90 80 65 50





We often draw diagrams of binary search trees like this one:



We have shown the keys but not the values in this tree. Where are the values?

In answer to "Why isn't this a BST?"

"it's because … the right only has depth 1, while the left has depth 3. … BSTs should have both sides equal depth. Is that a heap? It doesn't matter."

Danielsiek et al. SIGCSE 2012

The central new insight with respect to heaps is that students – even though they seem to have a rather good passive knowledge of the formal definition of a heap – tend to conflate heaps with binary search trees. Since the vast

A: *A heap is, er* [laughs], *a heap is a tree with an ordering and, er, that's hard to explain. Er, a binary tree.*

$\mathcal{I}$: *Let's talk about binary trees, shall we?*

A: *Binary trees are trees, all of them, which have, have two children, that's indicated by the word binary already, two, er, children, or two child nodes, at most that is, we could also have null. Er, and the child node, the one that's stuck to the left, is smaller than the the node itself and the one, the one stuck on the right has a bigger value than the node itself and that's than kept up recursively and then we get, if we keep hanging on to that thought, far, far, far down on the left the smallest one, the smallest element; far, far, far on the bottom on the right is the biggest element.*

B: *Yes, the heap was like this, er. We put the first number into the root and then we take the second number and we check whether it's bigger, er, than the root. If it's bigger, we write the number down on the right side, and if its smaller, then we write it down on the left side. And then we take the next number and do the same again starting with the root, we check whether it's bigger. If it's smaller, we go to the left again. And then we take a look at the next. If this one is smaller again then we will go to the left again of this one and to the right if its bigger.*