1 Departmental learning goals

The Department’s primary goal is to increase students’ proficiency in computational thinking and practice in a liberal arts context. We’re focusing on our introductory courses (21 and 21B), with the aim of teaching both non-majors and majors algorithmic problem solving, abstraction, design, and analysis by drawing on concepts fundamental to computer science. An additional goal is to expose students to how computation can be used to solve problems from many different disciplines. These broad goals are enumerated in more detail in the next section.

2 Departmental learning objectives

A student completing our introductory courses should be able to:

1. Use top-down design to sub-divide a large problem into reasonably sized sub-problems.

2. Given a problem described in English, design a clear, concise, and correct pseudocode algorithm to solve it.

3. Given a pseudocode algorithm, successfully implement it in a high-level programming language.

4. Given several algorithms for solving the same problem, analyze which would be more efficient.
5. Given a program, simulate on paper how a computer would execute the program. Show the internal state of the program during execution and show the program’s output.

6. Apply algorithmic problem solving skills to a variety of real-world applications.

Most of these goals (2-6) are easily assessed through traditional methods such as quizzes and lab assignments. However, the goals most closely linked to our overall aim of teaching computational thinking—using top-down design and developing algorithms—are the hardest to assess through standard methods. Our assessment activities are designed to measure how well these particular computational thinking skills are affected by taking our introductory courses.

3 Assessment activities

We propose a new assessment activity to get a better quantitative view of whether students’ computational thinking has improved. We will create a pre-test and post-test of algorithmic problem solving ability. The pre-test would be given in the first or second week of the semester and the post-test would be given in the twelfth or thirteenth week of the semester. We plan to give each student a unique id to allow us to easily compare their pre and post test scores while maintaining anonymity of responses. Each solution would be coded by a grader who will not know whether a response was from the pre or post test.

These problems should be designed so that they are general enough that students with no computing experience, but with already good problem solving skills, could solve them. We are currently exploring using problems available on the Math Forum, which has an archive of Problems of the week that may provide a good starting point.

We also currently do an end of the semester course evaluation as well as requesting a short bio from the students early on in the semester. We propose to tie these two activities together more directly. On the bio we ask students to say why they are taking the introductory course and to describe any prior computing experience. We will add an additional question asking them to assess their current problem solving ability. In addition, we will add a follow up question to the end of semester course evaluation to ask them to assess their problem solving abilities before and after taking our course. This will provide some useful qualitative data.
4 Using the results

Having both a quantitative and qualitative measure of how much students’ algorithmic problem solving abilities improve in our introductory courses will help us determine whether the current format of these courses is working well or needs to be modified.

Currently the course uses a mix of lectures and in-class exercises. The in-class, hands-on work is designed to give students frequent practice at taking a description of a problem and figuring out how to break it up into reasonable sub-problems, and how to implement the sub-problems computationally. We would like to have a better sense of whether these hands-on exercises are having the desired effect.