

1. MQP: Experimental investigation of heap-sort.

The method of sorting numbers in main storage known as heap-sort has been in a constant state of reinvention, with the objective of surpassing quicksort, which performs marvelously on the average, but has unfortunate worst case behavior. Several variants have been proposed that seem promising but are hard to analyze conclusively, and there is the need for a considerable experimental investigation, with the ultimate goal of tying the observations to suggestions that partial analysis has provided.

Desirable number of students: two, though it is possible there is room here for three.

Prerequisites: for CS seniors, probably nothing special, beyond a sufficient interest in the theory of algorithmics and some statistical insight (that might be acquired “on the job.”)

2. MQP: asymptotic behavior of solutions of divide-and-conquer equations.

A divide-and-conquer equation arises when we analyze a thus-called algorithm. A simple equation is as follows:

$$A(n) = 2A(\lfloor n/2 \rfloor) + n + 2, \quad n \geq 2, \quad A(1) = 1.$$

Over the decade several treatments of such equations have appeared in the literature. The reported results appear at first sight inconsistent.

This is probably not the case, but a clarification of the subject is desirable, and there is probably some room for improvement as well. A successful project is likely to lead to a journal or conference publication.

Number of students: one or, preferably, two.

Prerequisites: This is basically a mathematical work, but it requires little mathematics more advanced than the discrete math you have taken. It does require a mathematical turn of mind.

A possible plus is getting to learn to use symbolic calculations, using Maple or Mathematica, since I expect this approach to be suitable for the kind of calculations this project needs.

3. MQP: Finding average-optimal algorithms to locate the median of a small set of distinct numbers

Several algorithms need to find the median of a small sets of numbers (5, 7, 9) many times. Hence the interest in doing the selection optimally. Knuth gives for 5 and 7 numbers, claiming they are the best possible, but no algorithm or reference (he says they were found by one Mr. Howey who conducted an exhaustive search).

But the algorithms are of interest, and finding them, even for this few elements is not trivial.

Number of students: two.

Prerequisites: Nothing beyond software skills all seniors are expected to have. The project will require careful design, and possibly complicated data structures.

This problem could very likely be extended to a Master-level project.

4. IQP/MQP: Free, open-source software and its impact on software quality

The claim that open-source software can leverage the ability of a large number of knowledgeable users who can—and tend to—report on problems and propose fixes (which are in turn incorporated in later releases), has led to the claim that “*the best software is free software.*” The purpose of this project is to examine this claim, and the processes that contribute to its being right or wrong.

This is very much a sociology/professionalism studies topic. But will also involve a careful analysis of the concept “software quality” – which may well be the hardest, and most interesting part of the activity! (I am aware the concept has been bruited much about, but I am not familiar with the state of the argument).

Other aspects of the topic merit investigation. I describe here one, students with specific interests may consider others:

Economics: Who pays for this free software. Clearly a user who downloads an ftp version of Linux has not paid for the resources and efforts that were involved in creating it. This is also true for a company that obtained Apache in a similar way, to drive its web servers. What happened? Something out of nothing? Clearly not. A brief examination suggests there are two sources, for both kinds of resources – time of the developers and the machines used for development and delivery.

One source is volunteered free time of people interested in promoting the Open software movement, and their home computing/communications equipment. The other one is the “deep pockets” phenomenon, where employees of various companies use some of their company-paid time, and company equipment to develop this software. The significance of this source is not known, and unless this activity is done with the explicit permission of management, it may be hard to estimate...

5. MQP with MS potential: Maple packages for Analysis of Algorithms

Maple is known to you from the calculus courses. It is a marvelous tool in analysis of algorithms where often even modest analyses require significant calculations. The various projects I have here all call for *programming* in maple, not just using its functions.

There are several specific projects I would like to get done, and I expect new ones will surface continually. The descriptions below are only given to suggest the kind of projects possible.

1. Given a specification of a deterministic finite automaton (DFA), obtain the regular expression for the language it accepts, and create the corresponding univariate generating function. Maple has now tools to produce from such a function the number of words in the language of a given length, and to some extent an asymptotic series as well – but these functions would need to be interfaced to the DFA package.

2. Extending the above by a few types of preprocessing. Examples:

- Converting NFA to DFA.
- Minimizing a DFA.
- Creating a DFA for a language specified by its grammar; also for languages specified by requiring each word to include (or avoid) a given string.

3. Search trees. Procedures to manipulate binary search trees, compute characteristics such as width, height, external path length, produce graphic representations etc. . .

The manipulations of interest are creating a tree from an insertion sequence, adding/deleting a node, performing a “rotation,” finding a minimal-access cost shape for the tree, and several related heuristics.

4. Maple gives up on relatively simple calculations because it does not have a rich enough collection of techniques, including some that are very simple for humans to use, but not easy to automate. Allowing for interaction of the user with such procedures may be the best way to improve them. Example: a double summation which can be done by changing the order, something Maple never does on its own.

6. PhD suggestion: Implementing in the Maple language the Karr summation algorithms

Summation is harder than integration. It is a fact that seems surprising until you get into the heart of the matter. Now there is enough understanding to make it possible, for a very large class of summands, to be able to decide whether a closed-form results exists or not, and for most instances of the first case, derive it.

In spite of the title, this is not simply an implementation of a completely specified algorithm, since the papers of Michael Karr and several others, while algorithmic in nature, concentrate on the mathematical underpinning. Some interesting research would be required. A similar effort has been carried out in the context of Mathematica, and reading the relevant materials reveals that there is much scope for invention yet, probably of fundamental nature as well.

The completed work, if successful, stands to be popular tool among CS people, combinatorialists and numerous other groups.