1 Names and Emails

- Aleksandar Makelov — `amakelov@college.harvard.edu`
- Ben Wetherfield — `bwetherfield@college.harvard.edu`
- Chan Kang — `chankang@college.harvard.edu`
- Michael Fountaine — `mfount@college.harvard.edu`

2 Overview

Problem: Using Coq, verify Timsort, python's preferred sorting algorithm! **[Improve motivation? Possibly include discussion of failures of implementations of Timsort in languages other than Python 3.x.]** Timsort is a hybrid of insertion sort and mergesort, plus some heuristics about memory management and other optimizations. Our simplified version will include a reduced version of these heuristics.

Solution sketch: We will take an incremental approach (and all of our algorithms will be functional, using persistent data structures). We're going to start with a simplified version of Timsort, hybridizing mergesort and insertion sort with a small subset of the heuristics used by Timsort in full. At first, all of these components of this simplified Timsort will be independently verified; the combined algorithm implementation will be thoroughly unit-tested. For short, call this algorithm Simsort. Implementing Simsort will be the conclusion of our core functionality.

Next, we will implement heaps and a verified heapsort; by replacing insertion sort with heapsort in Simsort, we should get a constant-factor time improvement. Then, our primary goal beyond core functionality will be verification of Simsort. From there, we will implement extra extensions, as discussed below, possibly adding more heuristics to Simsort, approaching verification of Timsort in full.

Goals: Primarily, we'd like to verify Timsort (i.e., Simsort) as a way to learn more about Coq and certified programming.

3 Prioritized Feature List

Note: All algorithms and data structures used in this project will be functional; in particular, we'll use persistent data structures.

Core Features

- **Fundamentals.** Booleans, natural numbers (defined inductively), polymorphic lists, stacks (for very basic representations of memory needed within Timsort heuristics).
- **Verified insertion sort.** Verified insertion sort of lists of natural numbers.
- **Verified merge sort.** Verified merge sort of lists of natural numbers.
- **Simsort.** Fully tested implementation of Simsort (our hybridization of verified merge sort, verified insertion sort, and a modified subset of the heuristics used in Timsort). **[talk about what the heuristics might be]**

Cool Extensions

- **Heaps.** Polymorphic priority queues. If it provides an advantage in asymptotics, we will use heaps to re-implement stacks.
- **Heap Sort.** Verified heap sort of heaps of natural numbers. This will operate on lists of natural numbers, represented perhaps as trees or priority queues.
- **Augmenting Simsort with heapsort.** Fully tested implementation of Simsort, with heapsort replacing insertion sort, for a slight improvement in asymptotics.
- **Verified Simsort.** This is our main goal beyond core functionality. We will improve the fully tested Simsort to a rigorously verified Simsort (using Coq).
- **Passing foreign tests.** We have come across a few known to be broken implementations of Timsort in certain languages (e.g., Java's clone of Timsort, early versions of Python 2.x's Timsort). For this cool extension, we would take some of the failing test cases for those other implementations, adapt them to use the same heuristic assumptions that we've used with Simsort, and show that our verified Simsort passes those tests.
- **Adding more heuristics.** If we make it this far, we will add more heuristics to Simsort, showing that each addition passes verification and doesn't break invariants, working our way gradually to a verified, functional Timsort in full.

4 Technical Specification

4.1 Interfaces.

References to code written thus far are included throughout this section. `chicken` is the root directory of our github repository, located at `https://github.com/mfount/chicken`.

Data Structures, related methods, and proofs of them.

- **Booleans.** Our code for bools so far is contained in `chicken/coq/Basics.v`. This includes the proofs of axioms and binary operations
- **Natural numbers.** There is code for nats in `chicken/coq/Basics.v` and `chicken/coq/Induction.v`, which includes `ble_nat` (<= for nats) and its proof (likewise for other comparison functions, which we can pass into sorts). Proofs of axioms and comparison functions are also included.
- **Polymorphic lists.**
- **Stack.** We're going to implement this using polymorphic lists.
- **Heap.**

Algorithms and proofs of them.

- **Mergesort.**
- **Insertion sort.**
- **Heap sort.**
- **Simsort.**

5 Timeline

Remainder of this week.

- foo

Week of April 20.

- foo

[flesh this out]

6 Progress

References

[1] Chlipala, Adam. *Certified Programming with Dependent Types*.

[2] Pierce, Benjamin, et al. *Software Foundations*.