Thesis Timeline Proposal

Goal: focus on immutable lists for the purposes of my thesis
Out of scope: mutable lists, non-list collections
Acknowledgment: this may be an aggressive plan, what we actually complete may be a subset of the following goals.
General design plan for immutable lists: a generic interface, backed by a fancy persistent implementation using finger-trees. see jack's discussion on Github for more info

1. Finger-tree implementation (this is most of the work; 8-10 weeks)
    1. read about finger-tree / RRB trees and take some notes: one week
       2) implement the following functionality:
    - `List.add(element)` - adds an element to the end of the list *core*
    - `List.removeAt(index)` - remove the element at `index` *core*
    - `List.insert(index, element)` - push a new element into the list at `index` *core*
    - `List.set(index, element)` - change the element at `index` to `element` *core*
    - `List.get(index)` - returns the element at a position
    - `List.remove(element)` - remove the first occurrence of `element` from the list, if present
    - `List.sublist(start, end)` - return a view of this list's contents between `start` and `end`
    - `List.size` - get the size of the list
      plan to spend *4 - 7* weeks on the *core* methods (with tests!), rest of the time on the remaining methods
2. Benchmarking: 2 - 4 weeks (possibly also interleave benchmarking with implementation of specific generic methods?). We plan to benchmark the following situations
    1. Finger-tree list vs ConsList when functionally updating a list
    2. Case of for loop building up a list `builder` vs `conslist` vs `finger-tree list`
    3. How/where do we want to benchmark within the compiler to address Matthew's concerns?
    4. Generic method dispatch/dynamic dispatch overhead being a potential issue

In parallel jack will be working on the following (as well as assisting me with implementing the finger-tree stuff)

1. `List` interface
2. Porting existing List to ConsList which implements `List`
3. similar porting work for Array type which implements `List`

4. `List.builder()` implementation
5. Specialization for empty and singleton lists
6. `Indexable` stuff and `my_list[x]` notation stuff
7. Docs

Work schedule plan: Jack and I will meet on Mondays 6-8pm to work on this. I will work individually as well.