This week (Chapter 2):

- Abstract data types (**Today**)
- Data Structures (Friday and next week Monday)
- Programming practices (next week Monday)

Today:

- Recent exercises
- Analyses of merge sort and quick sort
- Definition *abstract data type*, especially in contrast with *data structure*
- The "canonical" ADTs
- Start data structures (time permitting)

```
public class SimpleLinkedList<E> implements Iterable<E> {
    private class Node {
        E datum;
        Node next;
        Node(E datum, Node next) {
            this.datum = datum;
            this.next = next;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public void set(int index, E element) {
        checkIndex(index);
        Node current = head;
        for (int i = 0; i < index; i++) current = current.next;
        current.datum = element;
    }
    // ...
}
```

Class invariant for `SimpleLinkedList`:

(a) `head` = `null` iff `tail` = `null` iff `size` = 0.

(b) If `tail` ≠ `null` then `tail.next` = `null`.

(c) If `head` ≠ `null` then `tail` is reached by following `size` − 1 `next` links from `head`.

1.17 State and prove a loop invariant for the loop of `set()`. The loop invariant should capture the meaning of the variables `current` and `head`.

1.18 Argue that `set()` preserves the class invariant.

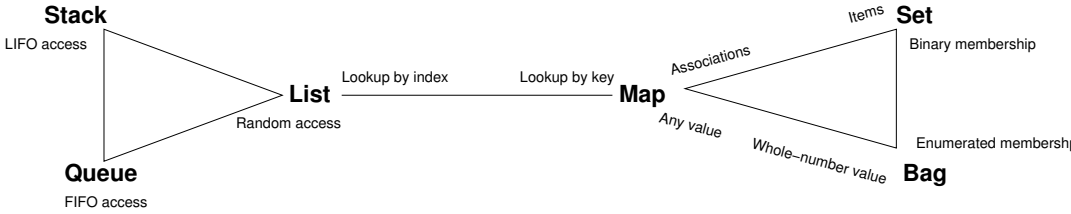
```
public class DataSet {
    private int sum, n, min, max;
    public DataSet(int initialVal) {
        min = max = sum = initialVal;
        n = 1;
    }
    public void add(int x) {
        sum += x;
        n++;
        if (x < min) min = x;
        if (x > max) max = x;
    }
    private double ave() { return ((double) sum) / n;}
    private int range() { return max - min; }
}
```

An abstract data type (ADT) is a data type whose representation is hidden from the client. Implementing an ADT as a Java class is not very different from implementing a function library as a set of static methods. The primary difference is that we associate data with the function implementations and we hide the representation of the data from the client. When using an ADT, we focus on the operations specified in the API and pay no attention to the data representation; when implementing an ADT, we focus on the data, then implement operations on that data.

[Sedgewick and Wayne, *Algorithms*, Pg 64; also cf pg 84]

The “canonical ADTs”:

- **List.** Linear collection with sequential and random access.
- **Stack.** Linear collection with LIFO access.
- **Queue.** Linear collection with FIFO access.
- **Set.** Unordered collection with binary membership.
- **Bag.** Unordered collection with enumerated membership.
- **Map.** Unordered collection of associations between keys and values.


Stack
LIFO access
Queue
FIFO access
List
Random access
Lookup by index
Lookup by key
Map
Items
Set
Binary membership
Associations
Any value
Whole−number value
Enumerated membershp
Bag

The four basic ways to implement an ADT:

- Use an array
- Use a linked structure
- Use an "advanced" data structure, varying and/or hybridizing linked structures and arrays
- Adapt an existing implementation of another ADT.

Coming up:

Due Thurs, Sept 8: (end of day)
Finish reading Section 2.1
Do Ex 1.11
Take ADT quiz

Due Mon, Sept 12:
Read Section 2.(2, 4, & 5)
Take data structures quiz

Also:
Do "basic data structures" practice problems (suggested by Wed, Sept 14)
Do **"Implementing ADTs"** *project (suggested by Fri, Sept 16)*