If you have not done so already, download the Erlang VM and libraries and try out the examples we did in class. See:  
http://www.cs.iastate.edu/~smkautz/cs430s15/examples/erlang/  
(download links and references are in erlang_links.txt).

Here are some problems that I have had a lot of fun with and I hope you do too.

1. Write a tail-recursive Erlang function that returns the nth Fibonacci number.

2. Write two functions to create lists of random integers, one that appends new elements at the head, and one that appends new elements at the tail. Which is more efficient? You can time execution using

   \[ \text{Start} = \text{erlang:now}(). \]
   \( \text{(do stuff)} \)
   \[ \text{End} = \text{erlang:now}(). \]
   \[ \text{Elapsed} = \text{timer:now_diff(End, Start)}. \]

   (result is in microseconds). (This explains why the function lists:reverse is so widely used in Erlang.)

   Note you can generate random integers between 1 and N inclusive with random:uniform(N). If you want to replicate the same list you can seed the generator using random:seed(A, B, C) where A, B, and C are integers you choose.

3. Try out the simple form of quicksort in Erlang, e.g.:

   http://www.erlang.org/doc/programming_examples/list_comprehensions.html

   (result is in microseconds). Is the performance similar to that of Java's Arrays.sort(int[])?

4. Write an Erlang implementation of the pi-approximation example
   http://www.cs.iastate.edu/~smkautz/cs430s15/examples/executor_examples/Pi.java
   using Erlang processes.
5. Write a function nodups/2 such that if L is a sorted list, the call

```
foldl(fun nodups/2, [], L)
```

results in a list containing the same elements as L but with duplicates removed. (I.e., the "accumulator" for the fold is also a list.) If you prefer, the result can be reversed.

6. The Eclipse project erlangtest.zip includes a version of the Universe and UniverseUI classes from homework 3. The UI provides the mechanism for manipulating "yahtzee cubes" as before, and the Universe is now an Erlang node (i.e. a distributed Erlang runtime with a known name) implemented using the JInterface library. Your task is to come up with an implementation of the actors for the cubes as Erlang processes.

The JInterface code is in OtpErlang.jar, a copy of which can be found in the project zip or in your Erlang installation. (Note: the version of the jar that was in my distribution requires Java 8. The source code is in the distribution too if you would rather use that.)

As a warm-up, try running EchoServer.java in the default package. It is currently set to produce a lot of verbose output, including output from the "Erlang port mapper daemon" or epmd, which routes messages between different Erlang nodes. Once it is running, start an Erlang REPL using the command

```
erl -sname mynode -setcookie thinmint
```

The "sname" option gives the node a "short name" to use for communication with other nodes, and the "cookie" option is just a string that has to match the cookie for the other nodes we wish to communicate with. Then at the prompt, type

```
>{mymailbox, echoserver@localhost} ! {self(), hello}
```

We don't know the Erlang pid of the echo server, but we can identify it using a tuple containing the mailbox name and host. The server expects a message of the form `{pid, atom}` and echos the atom to the pid as a reply.

By default the Universe is set up as `{mailbox, universe@localhost}` with cookie thinmint. It will handle four message types:

- `{register, {pid}}` - add the pid for a new cube
- `{left, <message>}` - broadcast `<message>` to the left
- `{right, <message>}` - broadcast `<message>` to the right
- `{update, {pid, value}}` - update the UI
You can define the `<message>` to have any form you want, as long as it is a tuple in which the sender's pid is first. The first thing each cube should do is send a register message to the Universe, which arbitrarily associates the cube's pid with one of the physical cubes on the screen.

You can set up a one-shot or periodic timer using the timer module. The code:

```erlang
timer:send_after(millis, pid, message)
```

will evaluate `pid ! message` after the given number of milliseconds.

For example, try this in the shell

```erlang
> timer:send_after(5000, self(), hello).
```

(and check for the result in the shell using `flush()` to see the message.)

Details of JInterface and timer can be found in the man pages,

Also remember that a Java version of a solution was posted earlier,
- [http://www.cs.iastate.edu/~smkautz/cs430s15/examples/hw3_sample_solutions/](http://www.cs.iastate.edu/~smkautz/cs430s15/examples/hw3_sample_solutions/)
if you want to refer to it.

More debugging if needed:

You can check the name of the node using `node()` and you can get the node's cookie using `erlang:getCookie()` to check. The name should not matter since the echoserver will reply using the pid that is sent as part of the message, but the node has to be started with `-sname`, not with `-name` (the "long name" version) or else it can't communicate with the Java based node, which is implicitly started with the `-sname` version.

A common problem is that the cookies don't match. If you see the Java side reporting something like `HANDSHAKE sendChallenge` followed by `CLOSE`, that is likely the problem. Be sure you start the node using `-setcookie thinmint` as an option (an error in an earlier version of this document had this as `-cookie thinmint`.)

The epmd (port mapper daemon) normally starts up when an node is created. You can also start it up in a verbose mode to maybe see more of what is happening. Shut down
the erlang REPL and the Java erlang node. Then at a command prompt, type `epmd -kill` to stop the port mapper daemon. Then restart it using `epmd -d -d -d`.

At a new command shell, you can type `empd -names` to see what names are known to the port mapper. You can get the same information in the REPL with `net_adm:names()`.

You can make a connection to a node without sending a message using `ping`,

```
> net_adm:ping(echoserver@localhost).
```

returns `pong` if successful and `pang` if unsuccessful.