Homework 6 Due: Midnight Wednesday, December 10

In class, we focused our attention on decision problems. These problems always have a "yes" or "no" answer. Thus, we define machines with "accepting" and "rejecting" states. If we encounter the end of the tape and we are in an "accepting state", we "accept", otherwise, we "reject".

In this homework, we are not interested in solving a decision problem. Thus, we will have no special "accepting" or "rejecting" states. Instead, we are interested in enumerating a list of values. In this homework, we will have an extra state called the "halt state". When the "halt state" is entered, the machine comes to a stop.

Write a Java program that simulates a Turing machine working on a tape filled with blanks and counting in binary until the size of the desired output is reached. The size of the output is specified in bits by the user at run time. In this homework, you may use any data structures that you like – built in data structures are fine. You may also use data structures that you have built in a previous homework.

For example, here are two example runs of my solution:

Java Turing

How many bit positions?
5
Running the machine on an initially empty tape.
0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001

```
11010
11011
11100
11101
11110
11111
Program complete in 2503000 nano seconds
Program complete in roughly 0  seconds
```

```
java Turing

How many bit positions?
2
Running the machine on an initially empty tape.
0
1
10
11
Program complete in 225000 nanoseconds
Program complete in roughly 0  seconds
```

Here is a near copy of my main routine which you should feel free to use or modify. Note that it does not contain the code to handle the timing of the program (in nanoseconds and seconds). Your solution will contain this timing related code.

```
public static void main( String args[]) throws IOException {


        System.out.println("How many bit positions?");
        BufferedReader is = new BufferedReader(new
                                    InputStreamReader(System.in));
        String line = is.readLine();
        n = Integer.parseInt(line);

        tape = new char[n];

        // This is a three state machine. It has states 0, and 1, and a halt state of 2.
        Turing machine = new Turing(3);

        //System.out.println("Adding transitions to state 0");

        machine.addTransition(0,'B', new Transition('0',Transition.LEFT, 2));
        machine.addTransition(0,'0', new Transition('1',Transition.LEFT, 2));
        machine.addTransition(0,'1', new Transition('0',Transition.LEFT, 1));

        //System.out.println("Adding transitions to state 1");

        machine.addTransition(1,'1', new Transition('0',Transition.LEFT, 1));
        machine.addTransition(1,'B', new Transition('1',Transition.LEFT, 2));
        machine.addTransition(1,'0', new Transition('1',Transition.LEFT, 2));
```

```
        // place n B's on tape
        initializeTapeWithBlanks();

        System.out.println("Running machine on an initially empty tape");


        while(stillRoomOnTape()) {
           // machine is in state 0
           //  tape head points to least significant bit
           machine.execute();   // execute until halt state is entered
           displayTape();
        }

    }
```

For another example of a program that simulates an automata, see the example program that simulates a DFSA from our first lecture in the slides entitled "Finite State Machines I". Note that there was a fundamental problem with the design of that program but it is certainly is in the spirit of what we are doing here.

When the grader grades your work, he will be looking at the code to see if it has stayed well within the spirit of a Turing machine simulation. A program that deviates significantly from that style will not earn many points. However, if you see ways to improve on the main routine above (again, staying within the spirit of a Turing machine simpulation) then, by all means, make improvements.

After you have completed your programming and have a solid working solution, run it several times and experiment with different values of n (the number of bit positions used). Within your documentation, you must provide a table showing the time it takes for your solution to run with the following values of n. Use the timing code that wrote into the main routine above. If you are unable to wait for a solution to run to completion, provide a reasoned estimate of the time it would take to complete. Complete the table below.

In addition, also within the comments of your code, provide a big theta estimate of how many operations are required with an input of size n.

In the table below, there are three columns. The first describes the size of the input tape used (number of bit positions). The second describes the amount of time (you choose appropriate units) that the program takes to run when the counting is displayed (with a println statement) and the third describes the amount of time (you choose appropriate units) that the program takes to run when the counting is not displayed (with the println statement commented out).

n	time to complete (with output)	time to complete (with no output)
10		
15		
20		
22		
24		
26		
28		
30		
35		
40		
45		
50		
55		
60		
65		

Submit to Blackboard your Netbeans project containing Turing.java and any other classes it may use. Your main routine may look nearly identical to the main routine above (but include the code to handle the timing of the program).

The grader will be looking for a clear discusssion of the run time performance described above, solid documentation (including pre- and post-conditions where appropriate), separation of concerns, and your ability to stay within the spirit of a Turning Machine simulation.