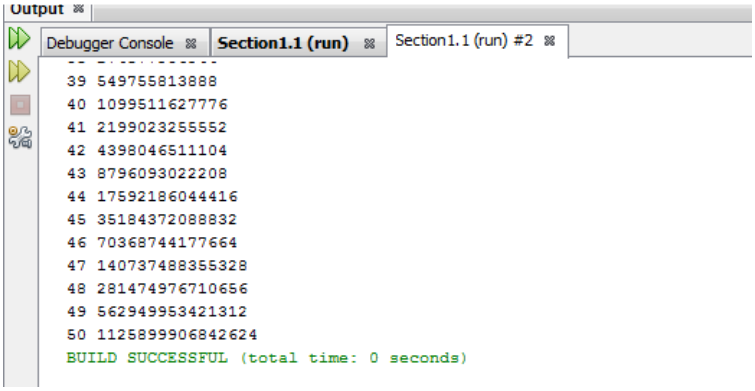
Section 2

Exercises

1. Use and adapt the code PowersOfTwo.java, to print the first 50 powers of 2^N. Include your code as well as the output result.



2. Use the code for RandomWalk.java to create 3 pictures that you like, using the number 100 as argument. To compile, you are required to previously compile StdDraw.java. You will produce 3 plots to be copied into your Activity log document.

3. Use the code Factors.java that prints the prime factors of a number. Follow the examples in the code headings comments and you are required to measure the computation time for the next 6 cases: 3, 6, 9, 12, 15, and 18 digit primes

- java Factors 997
- java Factors 999983
- java Factors 999999937
- java Factors 999999999989
- java Factors 999999999999989
- java Factors 999999999999999989

You are free to modify the source code to include a timing function. Here is an example you can review at stackoverflow.com. Include source code and output in your working document.

```
compile:
run:
The prime factorization of 997 is: 997
time: 958633 ns
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
compile:
run:
The prime factorization of 999983 is: 999983
time: 1166312 ns
BUILD SUCCESSFUL (total time: 0 seconds)
```

Section 2

```
compile:
run:
The prime factorization of 999999937 is: 999999937
time: 3825848 ns
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
compile:
run:
The prime factorization of 999999999989 is: 999999999989
time: 46361852 ns
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
compile:
run:
The prime factorization of 999999999999989 is: 999999999999989
time: 1220031649 ns
BUILD SUCCESSFUL (total time: 1 second)
```

```
compile:
run:
The prime factorization of 9999999999999989 is: 9999999999999989
time: 41435311193 ns
BUILD SUCCESSFUL (total time: 41 seconds)
```

4. Use the program FunctionGrowth.java that prints a table of the values of *log N*, *N*, *N log N*, N^2, N^3, and 2^N for *N* = 16, 32, 64, ..., 2048. What are the limits of this code? Suppose we want to stop not at N=2048. but at N=1073741824. Modify your code to do this. Add the modified code to your document and include generated output.

```
Section1.1 (run) %  Debugger Console %
7       2048    15615   4194304 8589934592
8       4096    34069   16777216        68719476736
9       8192    73817   67108864        549755813888
9       16384   158991  268435456       4398046511104
10      32768   340695  1073741824      35184372088832
11      65536   726817  4294967296      281474976710656
11      131072  1544487 17179869184     2251799813685248
12      262144  3270678 68719476736     18014398509481984
13      524288  6904766 274877906944    144115188075855872
13      1048576 14536349        1099511627776   1152921504606846976
14      2097152 30526334        4398046511104   9223372036854775808
15      4194304 63959939        17592186044416  73786976294838206464
15      8388608 133734419       70368744177664  590295810358705651712
16      16777216        279097919       281474976710656 4722366482869645213696
17      33554432        581453998       1125899906842624        37778931862957161709568
18      67108864        1209424316      4503599627370496        30223145490365729367654 4
18      134217728       2147483647      18014398509481984       24178516392292583494123 52
19      268435456       2147483647      72057594037927936       19342813113834066795298816
20      536870912       2147483647      288230376151711744      15474250491067253436239 0528
20      1073741824      2147483647      1152921504606846976     12379400392853802748991 24224
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Modify the code Binary.java that converts any number to binary form, to convert any number to its hexadecimal form. Print the first 256 numbers in hex. Include code and output in your working document.

Section 2

```
242 BINARY: 010 HEXA: F2
243 BINARY: 011 HEXA: F3
244 BINARY: 100 HEXA: F4
245 BINARY: 101 HEXA: F5
246 BINARY: 110 HEXA: F6
247 BINARY: 111 HEXA: F7
248 BINARY: 000 HEXA: F8
249 BINARY: 001 HEXA: F9
250 BINARY: 010 HEXA: FA
251 BINARY: 011 HEXA: FB
252 BINARY: 100 HEXA: FC
253 BINARY: 101 HEXA: FD
254 BINARY: 110 HEXA: FE
255 BINARY: 111 HEXA: FF
256 BINARY: 000 HEXA: 100
BUILD SUCCESSFUL (total time: 1 second)
```

6. Modify the code DayOfWeek.java to print the Day of the Week (Sunday, Monday, ...).
7. Let's play cards. Use the code Deal.java to play 21 or BlackJack for 2 users. You are always the first deal of cards, the house the second. Modify the code to ask for an additional card (Hit=1) or none (Stay=0) for the user. In 20 trials, how many times did you beat the house?. Add the modified code to your working document and describe your experience.
8. Use the code Birthday.java, to run at least 20 experiments and compute the average number of people needed to show up in a room in order that 2 people share the same birthday.
9. Use the code to build the Pascal triangle, Pascal.java. Produce a Pascal Triangle to level 10
10. You are required to run the code that generates a Sierpinski triangle: Sierpinski.java. This code requires compiling beforehand DrawingPanel.java. Can you guess an algorithm that counts how many solid black inverted triangles and how many upright white triangles per level N. Justify your answer.