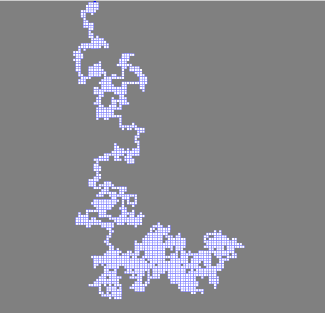
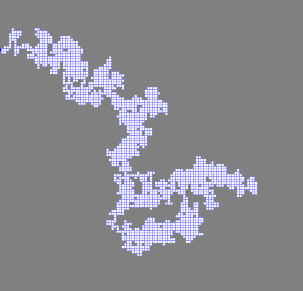
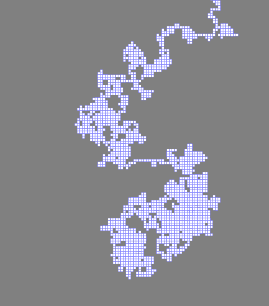
1. Use and adapt the code [PowersOfTwo.java](http://introcs.cs.princeton.edu/java/13flow/PowersOfTwo.java.html), to print the first 50 powers of 2^N. Include your code as well as the output result.
2. **public** **class** PowersOfTwo {
3. **public** **static** **void** main(String[] args) {
4. Scanner in = **new** Scanner(System.***in***);
5. System.***out***.print("Number of powers of 2^N : ");
6. String powers = in.next();
7. in.close();
9. // read in one command-line argument
10. **int** N = Integer.*parseInt*(powers);
11. **int** i = 0; // count from 0 to N
12. **int** powerOfTwo = 1; // the ith power of two
13. // repeat until i equals N
14. **while** (i <= N) {
15. System.***out***.println(i + " " + powerOfTwo); // print out the power of two
16. powerOfTwo = 2 \* powerOfTwo; // double to get the next one
17. i = i + 1;
18. }
19. }
20. }
21. Use the code for [RandomWalk.java](http://introcs.cs.princeton.edu/java/15inout/RandomWalk.java.html) to create 3 pictures that you like, using the number 100 as argument. To compile, you are required to previously compile [StdDraw.java](http://introcs.cs.princeton.edu/java/stdlib/StdDraw.java.html). You will produce 3 plots to be copied into your Activity log document.



1. Use the code [Factors.java](http://introcs.cs.princeton.edu/java/13flow/Factors.java.html) 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](http://stackoverflow.com/questions/3382954/measure-execution-time-for-a-java-method). Include source code and output in your working document.

**public** **class** Factors {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

System.***out***.print("Number of Factors : ");

String factors = in.next();

in.close();

**long** n = Long.*parseLong*(factors);

System.***out***.print("The prime factorization of " + n + " is: ");

**long** startTime = System.*currentTimeMillis*();

// for each potential factor i

**for** (**long** i = 2; i\*i <= n; i++) {

// if i is a factor of N, repeatedly divide it out

**while** (n % i == 0) {

System.***out***.print(i + " ");

n = n / i;

}

}

// if biggest factor occurs only once, n > 1

**if** (n > 1) System.***out***.println(n);

**else** System.***out***.println();

**long** stopTime = System.*currentTimeMillis*();

System.***out***.println("Total Execution Time: "+(stopTime - startTime) +" miliseconds");

}

}

1. Use the program [FunctionGrowth.java](http://introcs.cs.princeton.edu/java/13flow/FunctionGrowth.java.html) that prints a table of the values of *log N*, *N*, *N log N*, *N2*, *N3*, and *2N* 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.

**public** **class** FunctionGrowth {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

System.***out***.print("log N limit number : ");

String logN = in.next();

in.close();

**long** n = Long.*parseLong*(logN);

System.***out***.println("log N \tN \tN log N\tN^2 \tN^3");

**for** (**long** i = 2; i <= n; i \*= 2) {

System.***out***.print((**int**) Math.*log*(i));

System.***out***.print('\t'); // tab character

System.***out***.print(i);

System.***out***.print('\t');

System.***out***.print((**int**) (i \* Math.*log*(i)));

System.***out***.print('\t');

System.***out***.print(i \* i);

System.***out***.print('\t');

System.***out***.print(i \* i \* i);

System.***out***.println();

}

}

}

log N limit number : 1073741824

log N N N log N N^2 N^3

0 2 1 4 8

1 4 5 16 64

2 8 16 64 512

2 16 44 256 4096

3 32 110 1024 32768

4 64 266 4096 262144

4 128 621 16384 2097152

5 256 1419 65536 16777216

6 512 3194 262144 134217728

6 1024 7097 1048576 1073741824

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 0

15 8388608 133734419 70368744177664 0

16 16777216 279097919 281474976710656 0

17 33554432 581453998 1125899906842624 0

18 67108864 1209424316 4503599627370496 0

18 134217728 2147483647 18014398509481984 0

19 268435456 2147483647 72057594037927936 0

20 536870912 2147483647 288230376151711744 0

20 1073741824 2147483647 1152921504606846976 0

1. Modify the code [Binary.java](http://introcs.cs.princeton.edu/java/13flow/Binary.java.html) 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.

**public** **class** Binary {

**public** **static** **void** main(String[] args) {

Scanner in = **new** Scanner(System.***in***);

System.***out***.print("Number to Binary : ");

String number = in.next();

in.close();

**long** n = Long.*parseLong*(number);

System.***out***.println("HEX VALUE: "+Long.*toHexString*(n));

// set v to the largest power of two that is <= n

**int** v = 1;

**while** (v <= n/2) {

v = v \* 2;

}

System.***out***.print("BIN VALUE: ");

// check for presence of powers of 2 in n, from largest to smallest

**while** (v > 0) {

// v is not present in n

**if** (n < v) {

System.***out***.print(0);

}

// v is present in n, so remove v from n

**else** {

System.***out***.print(1);

n = n - v;

}

// next smallest power of 2

v = v / 2;

}

System.***out***.println();

System.***out***.println("HEXADECIMAL FIRST 256 NUMBERS: ");

**for**(**int** i=0; i<256; i++){

System.***out***.println("INT: "+i+" - HEX: "+Integer.*toHexString*(i));

}

}

}

**OUTPUT:**

Number to Binary : 10

HEX VALUE: a

BIN VALUE: 1010

HEXADECIMAL FIRST 256 NUMBERS:

INT: 0 - HEX: 0

INT: 1 - HEX: 1

INT: 2 - HEX: 2

INT: 3 - HEX: 3

INT: 4 - HEX: 4

INT: 5 - HEX: 5

INT: 6 - HEX: 6

INT: 7 - HEX: 7

INT: 8 - HEX: 8

INT: 9 - HEX: 9

INT: 10 - HEX: a

INT: 11 - HEX: b

INT: 12 - HEX: c

INT: 13 - HEX: d

INT: 14 - HEX: e

INT: 15 - HEX: f

INT: 16 - HEX: 10

INT: 17 - HEX: 11

INT: 18 - HEX: 12

INT: 19 - HEX: 13

INT: 20 - HEX: 14

INT: 21 - HEX: 15

INT: 22 - HEX: 16

INT: 23 - HEX: 17

INT: 24 - HEX: 18

INT: 25 - HEX: 19

INT: 26 - HEX: 1a

INT: 27 - HEX: 1b

INT: 28 - HEX: 1c

INT: 29 - HEX: 1d

INT: 30 - HEX: 1e

INT: 31 - HEX: 1f

INT: 32 - HEX: 20

INT: 33 - HEX: 21

INT: 34 - HEX: 22

INT: 35 - HEX: 23

INT: 36 - HEX: 24

INT: 37 - HEX: 25

INT: 38 - HEX: 26

INT: 39 - HEX: 27

INT: 40 - HEX: 28

INT: 41 - HEX: 29

INT: 42 - HEX: 2a

INT: 43 - HEX: 2b

INT: 44 - HEX: 2c

INT: 45 - HEX: 2d

INT: 46 - HEX: 2e

INT: 47 - HEX: 2f

INT: 48 - HEX: 30

INT: 49 - HEX: 31

INT: 50 - HEX: 32

INT: 51 - HEX: 33

INT: 52 - HEX: 34

INT: 53 - HEX: 35

INT: 54 - HEX: 36

INT: 55 - HEX: 37

INT: 56 - HEX: 38

INT: 57 - HEX: 39

INT: 58 - HEX: 3a

INT: 59 - HEX: 3b

INT: 60 - HEX: 3c

INT: 61 - HEX: 3d

INT: 62 - HEX: 3e

INT: 63 - HEX: 3f

INT: 64 - HEX: 40

INT: 65 - HEX: 41

INT: 66 - HEX: 42

INT: 67 - HEX: 43

INT: 68 - HEX: 44

INT: 69 - HEX: 45

INT: 70 - HEX: 46

INT: 71 - HEX: 47

INT: 72 - HEX: 48

INT: 73 - HEX: 49

INT: 74 - HEX: 4a

INT: 75 - HEX: 4b

INT: 76 - HEX: 4c

INT: 77 - HEX: 4d

INT: 78 - HEX: 4e

INT: 79 - HEX: 4f

INT: 80 - HEX: 50

INT: 81 - HEX: 51

INT: 82 - HEX: 52

INT: 83 - HEX: 53

INT: 84 - HEX: 54

INT: 85 - HEX: 55

INT: 86 - HEX: 56

INT: 87 - HEX: 57

INT: 88 - HEX: 58

INT: 89 - HEX: 59

INT: 90 - HEX: 5a

INT: 91 - HEX: 5b

INT: 92 - HEX: 5c

INT: 93 - HEX: 5d

INT: 94 - HEX: 5e

INT: 95 - HEX: 5f

INT: 96 - HEX: 60

INT: 97 - HEX: 61

INT: 98 - HEX: 62

INT: 99 - HEX: 63

INT: 100 - HEX: 64

INT: 101 - HEX: 65

INT: 102 - HEX: 66

INT: 103 - HEX: 67

INT: 104 - HEX: 68

INT: 105 - HEX: 69

INT: 106 - HEX: 6a

INT: 107 - HEX: 6b

INT: 108 - HEX: 6c

INT: 109 - HEX: 6d

INT: 110 - HEX: 6e

INT: 111 - HEX: 6f

INT: 112 - HEX: 70

INT: 113 - HEX: 71

INT: 114 - HEX: 72

INT: 115 - HEX: 73

INT: 116 - HEX: 74

INT: 117 - HEX: 75

INT: 118 - HEX: 76

INT: 119 - HEX: 77

INT: 120 - HEX: 78

INT: 121 - HEX: 79

INT: 122 - HEX: 7a

INT: 123 - HEX: 7b

INT: 124 - HEX: 7c

INT: 125 - HEX: 7d

INT: 126 - HEX: 7e

INT: 127 - HEX: 7f

INT: 128 - HEX: 80

INT: 129 - HEX: 81

INT: 130 - HEX: 82

INT: 131 - HEX: 83

INT: 132 - HEX: 84

INT: 133 - HEX: 85

INT: 134 - HEX: 86

INT: 135 - HEX: 87

INT: 136 - HEX: 88

INT: 137 - HEX: 89

INT: 138 - HEX: 8a

INT: 139 - HEX: 8b

INT: 140 - HEX: 8c

INT: 141 - HEX: 8d

INT: 142 - HEX: 8e

INT: 143 - HEX: 8f

INT: 144 - HEX: 90

INT: 145 - HEX: 91

INT: 146 - HEX: 92

INT: 147 - HEX: 93

INT: 148 - HEX: 94

INT: 149 - HEX: 95

INT: 150 - HEX: 96

INT: 151 - HEX: 97

INT: 152 - HEX: 98

INT: 153 - HEX: 99

INT: 154 - HEX: 9a

INT: 155 - HEX: 9b

INT: 156 - HEX: 9c

INT: 157 - HEX: 9d

INT: 158 - HEX: 9e

INT: 159 - HEX: 9f

INT: 160 - HEX: a0

INT: 161 - HEX: a1

INT: 162 - HEX: a2

INT: 163 - HEX: a3

INT: 164 - HEX: a4

INT: 165 - HEX: a5

INT: 166 - HEX: a6

INT: 167 - HEX: a7

INT: 168 - HEX: a8

INT: 169 - HEX: a9

INT: 170 - HEX: aa

INT: 171 - HEX: ab

INT: 172 - HEX: ac

INT: 173 - HEX: ad

INT: 174 - HEX: ae

INT: 175 - HEX: af

INT: 176 - HEX: b0

INT: 177 - HEX: b1

INT: 178 - HEX: b2

INT: 179 - HEX: b3

INT: 180 - HEX: b4

INT: 181 - HEX: b5

INT: 182 - HEX: b6

INT: 183 - HEX: b7

INT: 184 - HEX: b8

INT: 185 - HEX: b9

INT: 186 - HEX: ba

INT: 187 - HEX: bb

INT: 188 - HEX: bc

INT: 189 - HEX: bd

INT: 190 - HEX: be

INT: 191 - HEX: bf

INT: 192 - HEX: c0

INT: 193 - HEX: c1

INT: 194 - HEX: c2

INT: 195 - HEX: c3

INT: 196 - HEX: c4

INT: 197 - HEX: c5

INT: 198 - HEX: c6

INT: 199 - HEX: c7

INT: 200 - HEX: c8

INT: 201 - HEX: c9

INT: 202 - HEX: ca

INT: 203 - HEX: cb

INT: 204 - HEX: cc

INT: 205 - HEX: cd

INT: 206 - HEX: ce

INT: 207 - HEX: cf

INT: 208 - HEX: d0

INT: 209 - HEX: d1

INT: 210 - HEX: d2

INT: 211 - HEX: d3

INT: 212 - HEX: d4

INT: 213 - HEX: d5

INT: 214 - HEX: d6

INT: 215 - HEX: d7

INT: 216 - HEX: d8

INT: 217 - HEX: d9

INT: 218 - HEX: da

INT: 219 - HEX: db

INT: 220 - HEX: dc

INT: 221 - HEX: dd

INT: 222 - HEX: de

INT: 223 - HEX: df

INT: 224 - HEX: e0

INT: 225 - HEX: e1

INT: 226 - HEX: e2

INT: 227 - HEX: e3

INT: 228 - HEX: e4

INT: 229 - HEX: e5

INT: 230 - HEX: e6

INT: 231 - HEX: e7

INT: 232 - HEX: e8

INT: 233 - HEX: e9

INT: 234 - HEX: ea

INT: 235 - HEX: eb

INT: 236 - HEX: ec

INT: 237 - HEX: ed

INT: 238 - HEX: ee

INT: 239 - HEX: ef

INT: 240 - HEX: f0

INT: 241 - HEX: f1

INT: 242 - HEX: f2

INT: 243 - HEX: f3

INT: 244 - HEX: f4

INT: 245 - HEX: f5

INT: 246 - HEX: f6

INT: 247 - HEX: f7

INT: 248 - HEX: f8

INT: 249 - HEX: f9

INT: 250 - HEX: fa

INT: 251 - HEX: fb

INT: 252 - HEX: fc

INT: 253 - HEX: fd

INT: 254 - HEX: fe

INT: 255 - HEX: ff

1. Modify the code [DayOfWeek.java](http://introcs.cs.princeton.edu/java/12types/DayOfWeek.java.html) to print the Day of the Week (Sunday, Monday, ...).
2. Calendar c = Calendar.*getInstance*();
3. c.set(Calendar.***MONTH***, m-1);
4. c.set(Calendar.***DAY\_OF\_MONTH***, d);
5. c.set(Calendar.***YEAR***, y);
6. System.***out***.println(c.getDisplayName(Calendar.***MONTH***, Calendar.***LONG***, Locale.***US***));
7. Let's play cards. Use the code [Deal.java](http://introcs.cs.princeton.edu/java/14array/Deal.java.html) 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. **public** **class** Deal {
9. **public** **static** **void** main(String[] args) {
10. Scanner in = **new** Scanner(System.***in***);
11. System.***out***.print("Number of Players : ");
12. String players = in.next();
13. **int** PLAYERS = Integer.*parseInt*(players);
15. **int** CARDS\_PER\_PLAYER = 5;
16. String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
17. String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace" };
18. // avoid hardwired constants
19. **int** SUITS = suit.length;
20. **int** RANKS = rank.length;
21. **int** CARDS = SUITS \* RANKS;
22. **if** (CARDS\_PER\_PLAYER \* PLAYERS > CARDS) **throw** **new** RuntimeException("Too many players");
23. // initialize deck
24. String[] deck = **new** String[CARDS];
25. **for** (**int** i = 0; i < RANKS; i++) {
26. **for** (**int** j = 0; j < SUITS; j++) {
27. deck[SUITS\*i + j] = rank[i] + " of " + suit[j];
28. }
29. }
30. // shuffle
31. **for** (**int** i = 0; i < CARDS; i++) {
32. **int** r = i + (**int**) (Math.*random*() \* (CARDS-i));
33. String t = deck[r];
34. deck[r] = deck[i];
35. deck[i] = t;
36. }
37. String [] player = **new** String[5];
38. **int** index = 0;
39. // print shuffled deck
40. **for** (**int** i = 0; i < PLAYERS \* CARDS\_PER\_PLAYER; i++) {
41. System.***out***.println(deck[i]);
42. player[index++] = deck[i];
43. **if** (i % CARDS\_PER\_PLAYER == CARDS\_PER\_PLAYER - 1){
44. System.***out***.println("\nTHIS IS YOUR PREVIOUS DECK, Do you want to change a card (1=YES 0=NO) : ");
45. String extra = in.next();
46. **int** EXTRACARD = Integer.*parseInt*(extra);
48. **if**(EXTRACARD == 1){
49. **int** randomCard = (**int**)(Math.*random*()\*5 + 1);
50. player[randomCard] = deck[deck.length - i];
51. System.***out***.println("YOUR NEW DECK: ");
52. } **else** **if** (EXTRACARD == 0){
53. System.***out***.print("ACCEPT YOUR ORIGINAL DECK: ");
54. } **else** {
55. System.***out***.print("NOT VALID OPTION; ACCEPT YOUR ORIGINAL DECK: ");
56. }
58. **for**(**int** l=0; l<player.length; l++){
59. System.***out***.println(player[l]);
60. }
61. System.***out***.println();
62. System.***out***.println("---------------------------");
63. player = **new** String[5];
64. index = 0;
66. }
67. }
69. in.close();
70. }

**I won several times (12/20) and I just need to identify in where part of the program needs to do the program modification (we can do it in different places.)**

1. Use the code [Birthday.java](http://introcs.cs.princeton.edu/java/14array/Birthday.java.html), 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.

**11, 29, 19, 23, 55, 2, 25, 58, 17, 15, 14, 48, 11, 16, 22, 36, 25, 27, 35, 14 = 25.1**

1. Use the code to build the [Pascal triangle](http://en.wikipedia.org/wiki/Pascal's_triangle), [Pascal.java](http://introcs.cs.princeton.edu/java/14array/Pascal.java.html). Produce a Pascal Triangle to level 10

**1**

**1 1**

**1 2 1**

**1 3 3 1**

**1 4 6 4 1**

**1 5 10 10 5 1**

**1 6 15 20 15 6 1**

**1 7 21 35 35 21 7 1**

**1 8 28 56 70 56 28 8 1**

**1 9 36 84 126 126 84 36 9 1**

1. You are required to run the code that generates a [Sierpinski triangle](http://en.wikipedia.org/wiki/Sierpinski_triangle): [Sierpinski.java](http://courses.cs.washington.edu/courses/cse143/11sp/lectures/Sierpinski.java). This code requires compiling beforehand [DrawingPanel.java](https://www.cs.utexas.edu/~scottm/cs312/javaCode/Assignment1/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.

**Level 0 1 2 3 4**

**Inverted triangles: 0 1 4 (8/2) 13 (26/2) 40 (80/2)**

**Upright triangles: 1 3 9 27 81**

**With all this data test we can define (where level = n):**

**Inverted triangles: 3ⁿ- 1 / 2**

**Upright triangles: 3ⁿ**