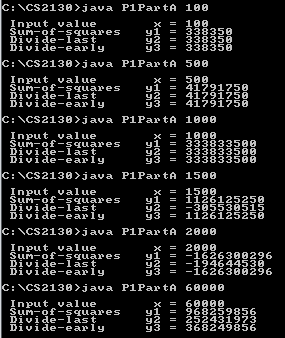
Nathan Brooks CS2130

Part A:

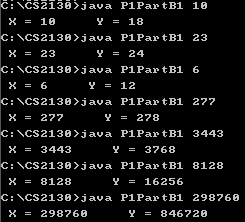
1 // Program to calculate the sum of squared integers  
 2 // using 3 algorithms.  
 3 public class P1PartA  
 4 {  
 5 public static void main(String args[])  
 6 {  
 7 int x, y1, y2, y3;  
 8 if(args.length >= 1){  
 9 x = Integer.parseInt(args[0]);  
10 System.out.println("\n Input value x = " + x);  
11 if (x >= 1){  
12 // Sum-of squares algorithm  
13 y1 = f1(x);  
14 System.out.println(" Sum-of-squares y1 = " + y1);  
15 // Divide-last algorithm  
16 y2 = f2(x);  
17 System.out.println(" Divide-last y2 = " + y2);  
18 // Divide-early algorithm  
19 y3 = f3(x);  
20 System.out.println(" Divide-early y3 = " + y3);  
21 }  
22 }  
23 return;  
24 } // end main  
25   
26 private static int f1(int x)  
27 {  
28 // Sum-of-squares algorithm  
29 int y = 0;  
30 for (int k = 1; k <= x; k++){  
31 y = y + k\*k;  
32 }  
33 return y;  
34 }  
35   
36 private static int f2(int x)  
37 {  
38 // Divide-last algorithm  
39 int y = 0;  
40 y = x\*(x + 1)\*(2\*x + 1)/6;  
41 return y;  
42 }  
43   
44 private static int f3(int x)  
45 {  
46 // Divide-early algorithm  
47 int y = 0;  
48 int A = x\*(x+1)/2;  
49 int B = 2\*x + 1;  
50 if(A % 3 == 0)  
51 y = (A/3)\*B;  
52 else  
53 y = A\*(B/3);  
54 return y;  
55 }  
56   
57 } // end class



1. For each of the three functions, what is the largest input that gives the correct output?
   1. Sum-of-Squares: 1500
   2. Divide-Last: 1000
   3. Divide-early: 1500
2. Which algorithm is "best"? Why?
   1. The Sum-Of-Squares algorithm and he Divide-Early algorithm seem to have the same limit, but the Sum-Of-Squares requires a for loop. It does a lot more work to achieve the same number. When we passed in 60,000 it had to loop through that many times, it sounds really intense. The Divide-Early algorithm is far less processor intensive.

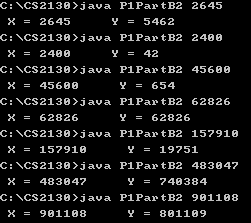
PartB1: Sum of Divisors

1 public class P1PartB1  
 2 {  
 3 public static void main(String args[])  
 4 {  
 5 long X, Y;  
 6 if(args.length >= 1){  
 7 X = Long.parseLong(args[0]);  
 8 if(X >= 1){  
 9 Y = SumOfDivisors(X);  
10 System.out.println("\n X = " + X + " Y = " + Y);  
11 }  
12 }  
13 return;  
14 }  
15   
16 public static long SumOfDivisors(long x)  
17 {  
18 long y = 0;  
19 for(long i=1; i<=x; i++)  
20 {  
21 if(x % i == 0)  
22 {  
23 y += i;  
24 }  
25 }  
26   
27 return y;  
28 }  
29   
30 } // end class



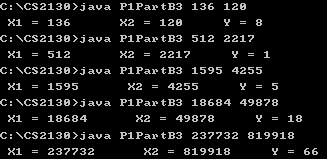
PartB2: Reverse Digits

1 public class P1PartB2  
 2 {  
 3 public static void main(String args[])  
 4 {  
 5 long X, Y;  
 6 if(args.length >= 1){  
 7 X = Long.parseLong(args[0]);  
 8 if(X >= 1){  
 9 Y = reverseDigits(X);  
10 System.out.println("\n X = " + X + " Y = " + Y);  
11 }  
12 }  
13 return;  
14 }  
15   
16 public static long reverseDigits(long x)  
17 {  
18 long y = 0;  
19 int intLength = 0; // to test how many digits x has  
20   
21 for(int i=1; i<=x; i\*=10)  
22 {  
23 if(x >= i)  
24 {  
25 intLength++; // count how many digits x has  
26 }  
27 }  
28   
29 for(int i=0; i<intLength; i++)  
30 {  
31 // shift result to the left one  
32 y \*= 10;  
33   
34 // add the remainder from dividing by 10  
35 y += x % 10;  
36   
37 // shift x right one and truncates off the decimal because it is a long  
38 x /= 10;  
39 }  
40   
41 return y;  
42 }  
43 } // end class



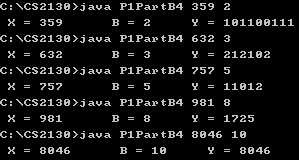
PartB3: GCD

1 public class P1PartB3  
 2 {  
 3 public static void main(String args[])  
 4 {  
 5 long X1, X2, Y;  
 6 if(args.length >= 1){  
 7 X1 = Long.parseLong(args[0]);  
 8 X2 = Long.parseLong(args[1]);  
 9   
10 if(X1 >= 1 && X2 >= 1){  
11 Y = GCD(X1, X2);  
12 System.out.println("\n X1 = " + X1 + " X2 = " + X2 + " Y = " + Y);  
13 }  
14 else{  
15 System.out.println("Invalid numbers");  
16 }  
17   
18 }  
19 return;  
20 }  
21   
22 public static long GCD(long x1, long x2)  
23 {  
24 long y = 0;  
25 long swapper = 0;  
26 long larger = 0;  
27   
28 // find out which of the values is smaller  
29 if(x1 - x2 < 0)  
30 {  
31 y = x1;  
32 larger = x2;  
33 }  
34 else  
35 {  
36 y = x2;  
37 larger = x1;  
38 }  
39   
40 // while remainder != 0  
41 while(larger % y != 0)  
42 {  
43 swapper = y;  
44 y = larger % y;  
45 larger = swapper;  
46 }   
47 return y;  
48 }  
49 } // end class



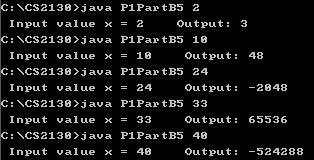
PartB4: BaseB

1 public class P1PartB4  
 2 {  
 3 public static void main(String args[])  
 4 {  
 5 long X, B, Y;  
 6 if(args.length >= 1){  
 7 X = Long.parseLong(args[0]);  
 8 B = Long.parseLong(args[1]);  
 9   
10 Y = BaseB(X, B);  
11   
12 System.out.println("\n X = " + X + " B = " + B + " Y = " + Y);   
13 }  
14 return;  
15 }  
16   
17 //--------------------------------------------  
18 // I wrote my own exponent function right here  
19 //--------------------------------------------  
20   
21 public static long power(long base, long exponent)  
22 {  
23 long product = base;  
24 for(int i=0; i<exponent; i++)  
25 {  
26 product \*= base;  
27 }  
28 return product;  
29 }  
30   
31 //------------------------------------------  
32   
33 public static long BaseB(long x, long b)  
34 {  
35 long exponentCounter = b;  
36 long exponentCount = 0;  
37 long remainder = 0;  
38 long numToReturn = 0;  
39 long timesMultiplied;  
40   
41   
42 // finding the largest exponent possible for our base  
43 while(exponentCounter < x)  
44 {  
45 exponentCounter \*= b;  
46 exponentCount++;  
47 }  
48   
49 for(long i=exponentCount; i>=-1; i--)  
50 {   
51 // shift the number to the left one space. First time does nothing because it is 0  
52 numToReturn \*= 10;  
53   
54 // while the base raised to the current power is less than the x being manipulated  
55 timesMultiplied = 0;  
56 while(power(b, i) < x)  
57 {  
58 timesMultiplied++;  
59 x -= power(b, i);  
60 }  
61 numToReturn += timesMultiplied;  
62 }  
63   
64 // for the final addition so I don't end up with a 0 as my final digit  
65 // the power function I wrote does not handle a 0 exponent, it just returns 0  
66 // this line fixes my lack-luster power function I wrote  
67 numToReturn += x;  
68   
69 return numToReturn;  
70 }  
71 } // end class



PartB5: RecursiveSeq

1 public class P1PartB5  
 2 {  
 3 public static void main(String args[])  
 4 {  
 5 long[] arr = new long[51];  
 6 int x;  
 7   
 8 if(args.length >= 1){  
 9 x = Integer.parseInt(args[0]);  
10   
11 arr[0] = 0;  
12 arr[1] = 1;  
13 arr[2] = 3;  
14   
15 for(int i=3; i<=50; i++)  
16 {  
17 arr[i] = 2 \* arr[i-1] - 2 \* arr[i-2];  
18 }  
19   
20 System.out.println("\n Input value x = " + x + " Output: " + arr[x]);  
21 }  
22 return;  
23 } // end main  
24 } // end class



PartB2: Reverse Digits is not one to one because:

X = 40, Y = 4,

X = 400, Y = 4

PartB3: GCD is not one-to-one because it is a 3 dimensional algorithm, but still:

X1 = 3 X2 = 9 Y = 12

X1 = 3 X2 = 12 Y = 12

PartB4: BaseB is not one-to-one because this is a 3 dimensional algorithm. It takes 2 variable inputs and gives you an output so it is not one-to-one, it is more like a two-to-one, except that it is possible to get the same output from varying the two inputs. For example, maybe 12345 in base 3 is the same as 56654 in base 6, this type of output is possible.

PartB5: RecursiveSeq could possibly not be one-to-one because the outputs bounce all over the place. I did not find any area that it is not one-to-one, but it still seems possible.