Instructions

Please edit the pages for the three questions in-place.

Provide the answer to each question, plus the code used to solve it.

Any programming language may be used, any libraries, etc.

If using an external resource please credit it.

Large summation

The multiples of 5 are: 5, 10, 15, 20, 25, 30, 35...

The multiples of 7 are: 7, 14, 21, 28, 35...

If we add the multiples of 5 and 7 that are less than 40, we get:

5 + 7 + 10 + 14 + 15 + 20 + 21 + 25 + 28 + 30 + 35 = 210

What is the sum of the multiples of 5 and 7 that are less than 1 million?

157143042856.800

BigDecimal N = new BigDecimal(1000000);  
BigDecimal k1 = new BigDecimal(5);  
BigDecimal k2 = new BigDecimal(7);  
BigDecimal k3 = k1.multiply(k2);  
  
BigDecimal sum1 = N.subtract(BigDecimal.*ONE*)  
 .divide(k1)  
 .multiply(N.subtract(BigDecimal.*ONE*)  
 .divide(k1)  
 .add(BigDecimal.*ONE*))  
 .multiply(k1.divide(new BigDecimal(2)));  
  
BigDecimal sum2 = N.subtract(BigDecimal.*ONE*)  
 .divide(k2)  
 .multiply(N.subtract(BigDecimal.*ONE*)  
 .divide(k2)  
 .add(BigDecimal.*ONE*))  
 .multiply(k2.divide(new BigDecimal(2)));  
  
BigDecimal sum3 = N.subtract(BigDecimal.*ONE*)  
 .divide(k3)  
 .multiply(N.subtract(BigDecimal.*ONE*)  
 .divide(k3)  
 .add(BigDecimal.*ONE*))  
 .multiply(k3.divide(new BigDecimal(2)));  
BigDecimal result = sum1.add(sum2).subtract(sum3);  
System.*out*.println(result);

Recursively-defined sequence

Let the function f(n) return (n / 2) if n is even, and (1 + 3 \* n) if n is odd.

Examples:

• f(1) = 4

• f(2) = 1

• f(3) = 10

• f(4) = 2

The function a(n) returns a sequence of numbers. That sequence is determined as follows:

• The initial element is n.

• Each subsequent element is the function f applied to the previous element.

• The sequence terminates after it evaluates to 1.

Example: a(3) returns[3, 10, 5, 16, 8, 4, 2, 1]

• n=3, so that is the first element.

• We apply f to the previous element. f(3) is 10, so that is the second element.

• We continue: f(10) = 5, f(5) = 16, f(16) = 8, f(8) = 4, f(2) = 1.

• We terminate after the sequence reaches 1.

The sum of the elements returned by a(3) is 3 + 10 + 5 + 16 + 8 + 4 + 2 + 1 = 49.

What is the sum of the elements returned by a(6171)?

12152173

public static void main(String[] args) {  
 System.*out*.println(*sum*(*a*(6171)));  
}  
  
public static Integer f(Integer n) {  
 return n % 2 == 0 ? n / 2 : 1 + 3 \* n;  
}  
  
public static List<Integer> a(Integer n) {  
 List<Integer> result = new ArrayList<>();  
 result.add(n);  
 Integer temp = n;  
 do {  
 temp = *f*(temp);  
 result.add(temp);  
 } while (temp != 1);  
 return result;  
}  
  
public static BigDecimal sum(List<Integer> sequence) {  
 return sequence.stream().map(BigDecimal::new).reduce(BigDecimal::add).orElse(BigDecimal.*ZERO*);  
}

Leap day

February 29th is called the "Leap Day."

For our purposes, a year has a Leap Day if:

• The year number is divisible by 400, OR

• The year number is divisible by 4, but not by 100

Examples:

• 2000 is divisible by 400, so it has a leap day.

• 1940 is divisible by 4 and not by 100, so it has a leap day.

• 1900 is divisible by 4, but it is divisible by 100, so it does not have a leap day.

A calendar shows what day of the week a leap day falls on.

• The leap day in 1940 fell on a Thursday.

• The leap day in 2000 fell on a Tuesday.

Between 1900 and 2000 (inclusive), how many leap days fell on each weekday?

FRIDAY 3

TUESDAY 3

SUNDAY 3

WEDNESDAY 3

THURSDAY 4

SATURDAY 4

MONDAY 4

public static void main(String[] args) {  
 System.*out*.println(*getLeapDays*(1900, 2000));  
}  
  
public static String getLeapDays(Integer start, Integer end) {  
 List<Integer> years = *getYearsWithLeapDay*(start, end);  
 Map<DayOfWeek, Long> table = *getDaysOfWeek*(years);  
 return table.keySet().stream().map(key -> key + " " + table.get(key)).collect(Collectors.*joining*("\n"));  
}  
  
public static List<Integer> getYearsWithLeapDay(Integer start, Integer end){  
 return IntStream.*range*(start, end)  
 .filter(n -> n % 400 == 0 || (n % 4 == 0 && n % 100 != 0))  
 .boxed()  
 .collect(Collectors.*toList*());  
}  
  
public static Map<DayOfWeek, Long> getDaysOfWeek(List<Integer> years) {  
 return years.stream()  
 .map(year -> LocalDate.*parse*(year + "-02-29"))  
 .map(LocalDate::getDayOfWeek)  
 .collect(Collectors.*groupingBy*(Function.*identity*(), Collectors.*counting*()));  
}