In this problem you will implement four static methods that manipulate ints.

The first static method is partialSumDivisor(int num), which returns true if the parameter num has the partial sum property. A number is said to have the partial sum property if it is divisible by the sum exactly two of its digits

The following code shows the results of the partialSumDivisor method.

|  |  |
| --- | --- |
| The following code | Returns |
| NumberFun.partialSumDivisor(63) // 63 % (6+3) == 0 | true |
| NumberFun.partialSumDivisor(14989) // 14989 % (4+9) == 0 | true |
| NumberFun.partialSumDivisor(1423) | false |
| NumberFun.partialSumDivisor(90473) | false |

The second static method is sumProd(int sum, int prod, int x) which finds two int values m and n such that sum = m + n and prod = mn, and returns mx + nx. If no two values are found, return -1.

You may assume prod != 0 (yes, sum may equal 0) and x > 0.

The following code shows the results of the sumProd method.

|  |  |
| --- | --- |
| The following code | Returns |
| NumberFun.sumProd(1, -12, 3) | -27+64 |
| NumberFun.sumProd(4, -5, 2) | 1+25 |
| NumberFun.sumProd(-2, -15, 3) | -98 |
| NumberFun.sumProd(-4, -32, 2) | 80 |
| NumberFun.sumProd(2, 1, 1) | 2 |
| NumberFun.sumProd(10, 16, 2) | 68 |
| NumberFun.sumProd(-5, 6, 3) | -35 |
| NumberFun.sumProd(-8, 15, 5) | -3368 |

The third static method is getNumDigitSums(int low, int high, int target), which returns the number of ints greater than or equal to low and less than or equal to high that the sum of all digits equal the parameter target.

The following code shows the results of the getNumDigitSums method.

|  |  |
| --- | --- |
| The following code | Returns |
| NumberFun.getNumDigitSums (1339, 1505, 9) | 8 |
| NumberFun. getNumDigitSums(112, 1905, 5) | 29 |
| NumberFun. getNumDigitSums(1075, 1301, 11) | 21 |

The fourth static method is factorialFun(int n, int k), which returns the number of times k is a factor of n! (n factorial).

Recall: n! = n \* (n-1) \* (n-2) \* … \* 3 \* 2 \* 1

For example:

* given 6! and k = 3: 6! = 6 \* 5 \* 4 \* 3 \* 2 \* 1 = 2^4 \* 3^2 \* 5, return 2
* given 16! and k = 15: 16! = 16 \* 15 \* 14 \* ... \* 3 \* 2 \* 1 = 2^15 \* 3^6 \* 5^3 \* 7^2 \* 11 \* 13, return 3

you may assume 0 <= n < 100 and k > 1.

The following code shows the results of the factorialFun method.

|  |  |
| --- | --- |
| The following code | Returns |
| NumberFun.factorialFun(6, 3) | 2 |
| NumberFun.factorialFun(16, 15) | 3 |
| NumberFun.factorialFun(29, 5) | 6 |
| NumberFun.factorialFun(10, 2) | 8 |