System Test Plan
Graph-Based Coverage

By: Karl Dylan Baes

Table of Contents

1. System Under Test

The Apache Commons Numbers "provides implementations of number types and utilities" for those numbers. Some features include and are not limited to prime numbers, fractions, complex numbers, and gamma functions.

1.1. Details

Homepage: https://commons.apache.org/proper/commons-numbers/developers.html
GitHub: https://github.com/apache/commons-numbers

The project source code includes 13 packages and their corresponding class files:

- Commons Numbers Angles
  - Angle; LOC 311
  - CosAngle; LOC 44
  - Reduce; LOC 49
- Commons Numbers Arrays
  - MultidimensionalCounter; LOC 214
  - Enum File SortInPlace; LOC 138
- Commons Numbers Combinatorics
  - BinomialCoefficient; LOC 147
  - BinomialCoefficientDouble; LOC 125
  - Combinations; LOC 376
  - CombinatoricsException; LOC 46
  - Factorial; LOC 279
  - FactorialDouble; LOC 71
  - LogBinomialCoefficient; LOC 85
  - LogFactorial; LOC 119
- Commons Numbers Complex Streams
  - ComplexUtils; LOC 1624
- Commons Numbers Complex
  - Complex; LOC 3697
- Commons Numbers Core
  - Interface Addition; LOC 47
  - ArithmeticUtils; LOC 606
  - ExtendedPrecision; LOC 307
  - Interface Multiplication; LOC 47
  - Interface NativeOperators; LOC 59
  - Enum Norm; LOC 571
  - Sum; LOC 255
- Commons Numbers Examples

    - ComplexPerformance; LOC 655
    - SinCosPerformance; LOC 331
    - DoublePrecision; LOC 705
    - DoubleSplitPerformance; LOC 615
    - DoubleUtils; LOC 72
    - EuclideanNormAlgorithmPerformance; LOC 224
    - EuclideanNormAlgorithms; LOC 751
- Commons Numbers Field
    - AbstractField; LOC 69
    - BigFractionField; LOC 47
    - FP64; LOC 172
    - FP64Field; LOC 49
    - Interface Field; LOC 83
    - FractionField; LOC 47
- Commons Numbers Fraction
    - BigFraction; LOC 1292
    - ContinuedFraction; LOC 158
    - Fraction; LOC 910
    - FractionExceotion; LOC 55
    - GeneralizedContinuedFraction; LOC 460
- Commons Numbers Primes
    - Primes; LOC 115
    - SmallPrimes; LOC 289

1.2. Software Architecture

The source code is structured generically in a way that each library is nested in its own folder. Fortunately, this software has exceptional documentation in which most of the functionalities can easily be accessed and read via the Javadoc that is provided in each library. When looking at each file and their documentation, the methods are arranged by their categories: static, instance, or concrete methods. There are also not much global variables used in these methods. Most of the variables are local to their class and are mostly input variables, if not, variables extracted from Java libraries which is then used to develop their methods.

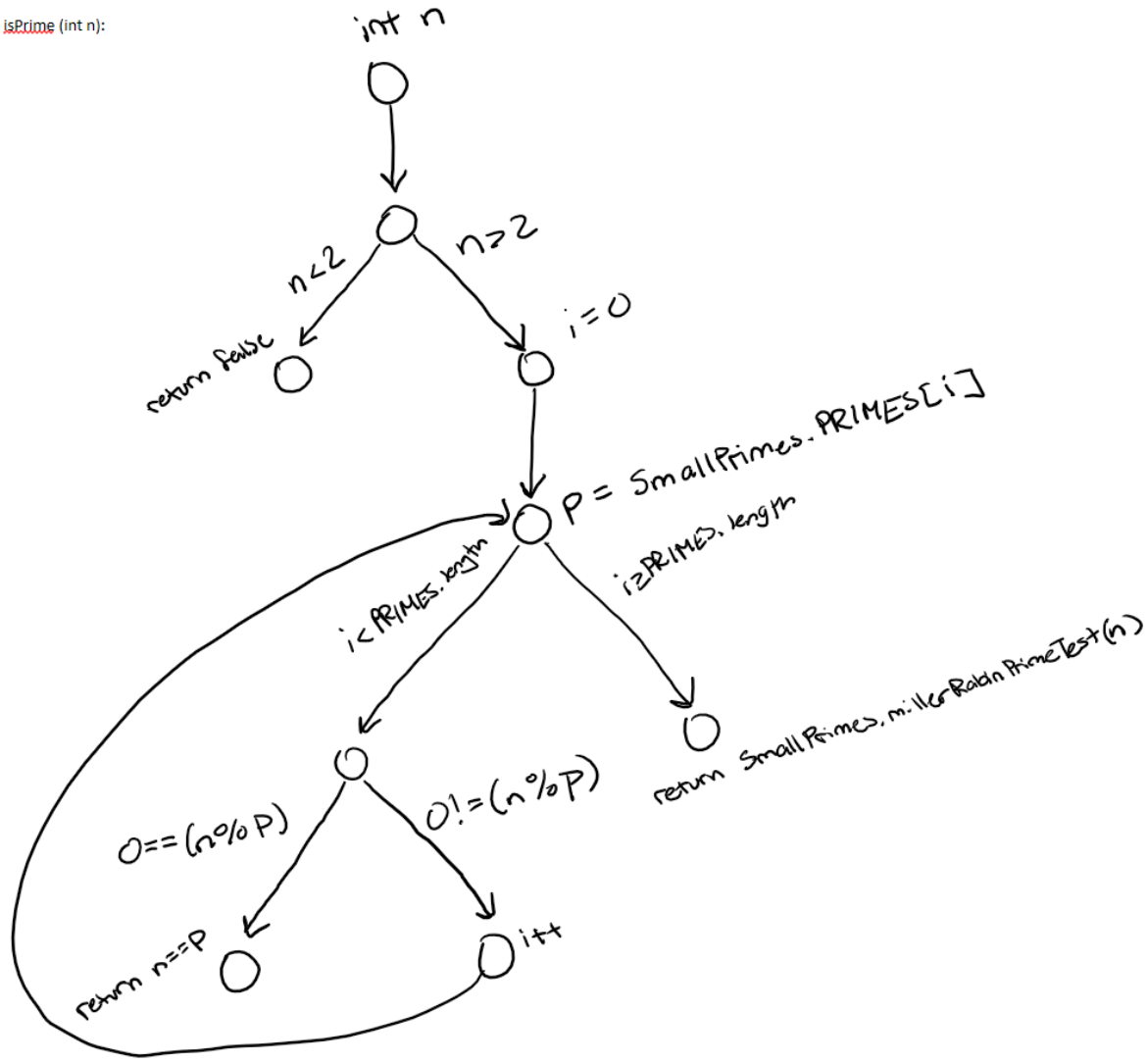
Test Environment

Project under test is deployed and hosted on GitHub as linked ***Details*** section. The operating system used to run tests is Windows 10, on IntelliJ IDE 2022.2.2, Maven, OpenJDK version 11, JUnit 5.8.1.

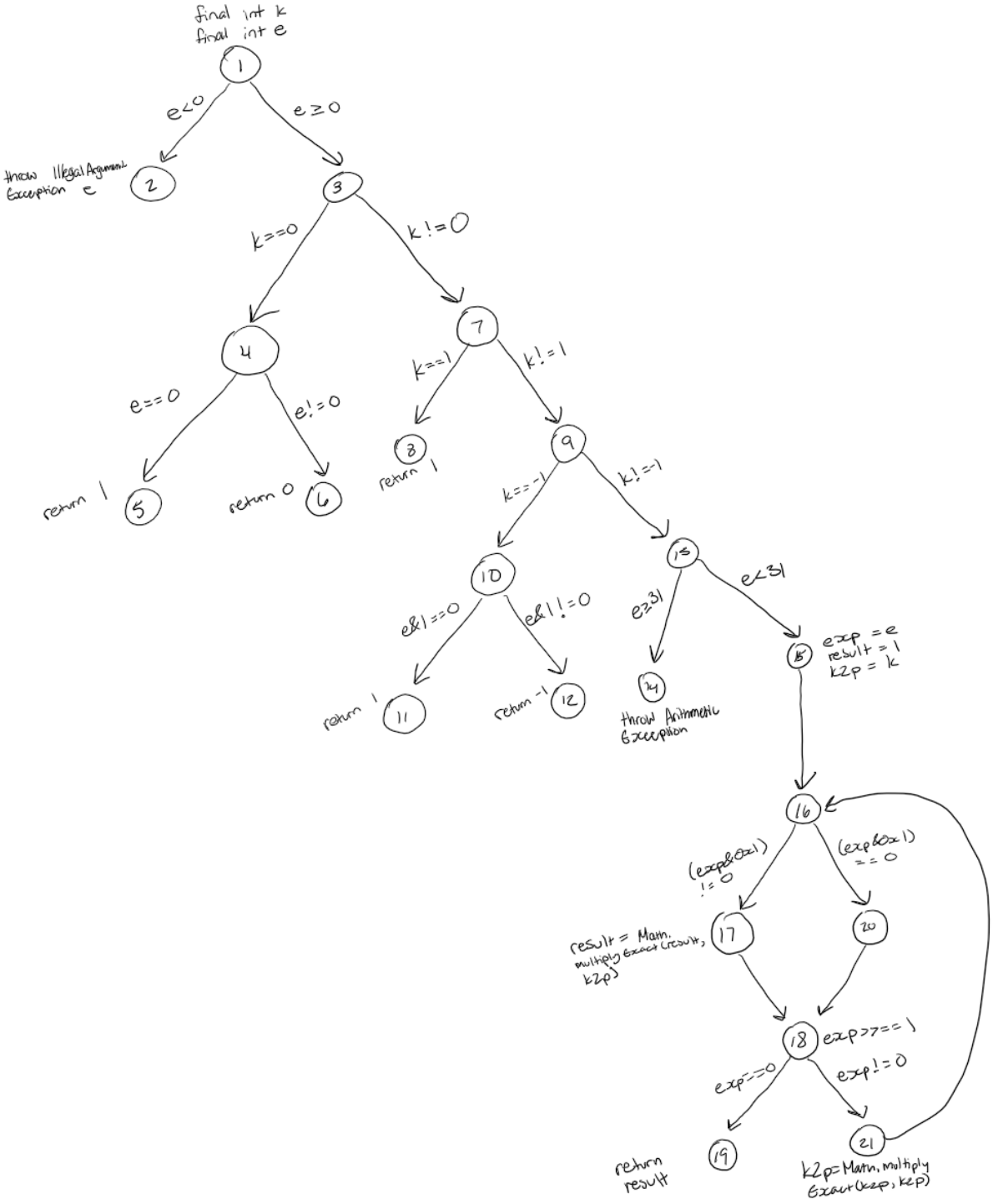
2. Control Flow

2.1. isPrime Control Flow Graph

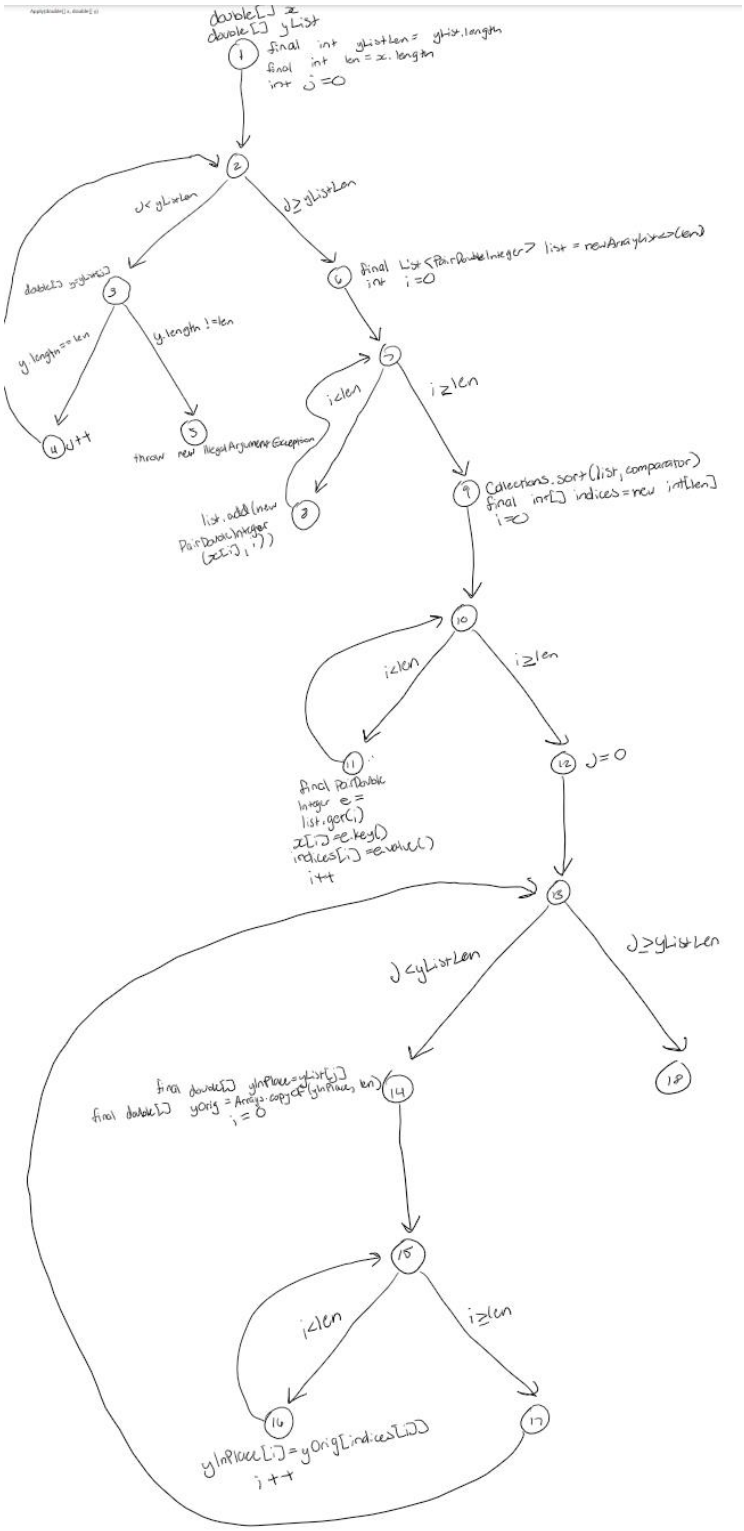
isPrime (int n):



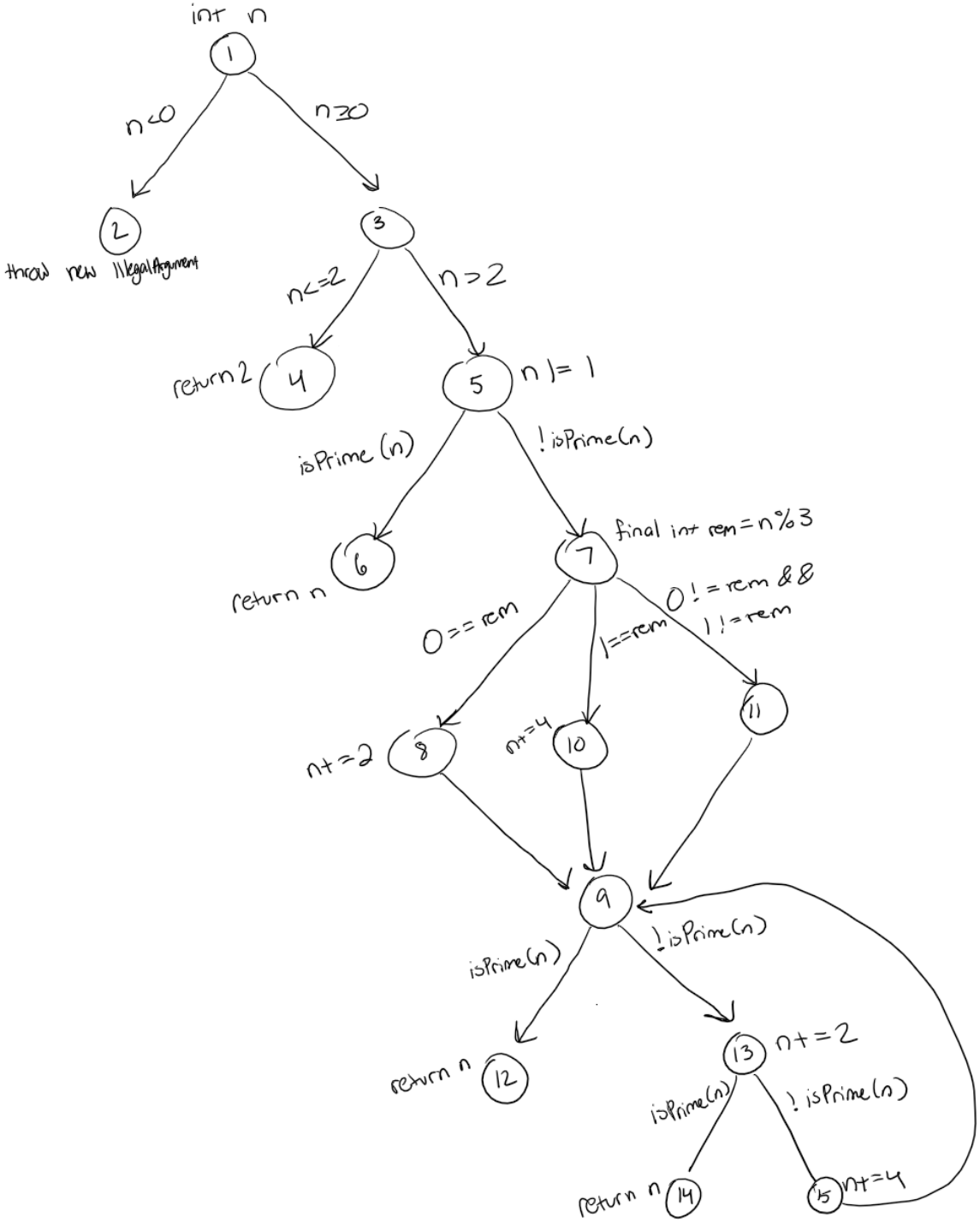
2.2. pow Control Flow Graph



2.3. apply Control Flow Graph



2.4. nextPrime Control Flow Graph



Test Requirements

2.5. isPrime Coverage Criteria

For this method, prime path coverage will be used.

TR ID	Test Requirement
1	[1,2,3]
2	[1,2,4,5,6,7]
3	[1,2,3,4,5,8]
4	[1,2,4,5,9]
5	[5,6,8,5]

Test Path ID	Test Path	Covered TR	Feasible?
1	[1,2,3]	1	Y, n = 1
2	[1,2,4,5,6,7]	2	Y, n = 3
3	[1,2,4,5,6,8,5,6,7]	3,5	Y, n = 7
4	[1,2,4,5,9]	-1	N, the PRIMES[] array is static and therefore I can never start off being greater than PRIMES.length
5	[1,2,4,5,6,8,5,9]	4	Y, n = 3673

2.6. apply Coverage Criteria

For this method, du-path coverage will be used.

TR ID	Test Requirement
6	(-1, 1, yList)
7	(-1, 3, yList)
8	(-1, 14, yList)

Test Path ID	Test Path	Covered TR	Feasible?
6	[1,2,3,5]	1,2	Y, x[] array size != yList[] array size
7	[1,2,3,4,2,6,7,9,10,12,13,14,15,17,13,18]	3	N, infeasible as the size of the array would have to be < 0 which is impossible
8	[1,2,3,4,2,6,7,8,7,9,10,11,10,12,13,14,15,16,17,13,18]	3	Y, x = [1, 8, 2] yList = [5, 12, 33]

2.7. nextPrime Coverage Criteria

For this method, du-path coverage will be used.

TR ID	Test Requirement
9	(-1, 1, n)
10	(-1, 2, n)
11	(-1, 3, n)
12	(5, 6, n)
13	(5, 7, n)
14	(5, 9, n)
15	(5, 12, n)
16	(8, 9, n)
17	(8, 12, n)
18	(10, 9, n)
19	(10, 12, n)
20	(13, 14, n)
21	(15, 9, n)
22	(15, 12, n)
23	(1, 5, n)
24	(5, 8, n)
25	(5, 10, n)
26	(8, 13, n)
27	(10, 13, n)
28	(13, 15, n)

Test Path ID	Test Path	Covered TR	Feasible?
9	[1,2]	9, 10	Y, n = -1
10	[1,3,4]	11	Y, n = 2
11	[1,3,5,6]	23,12	Y, n = 3
12	[1,3,5,7,11,9,12]	13,14, 15	N, n is not prime and will not be able to satisfy isPrime(), therefore (5,12) is infeasible
13	[1,3,5,7,8,9,12]	13, 14, 24, 17, 16	Y, n = 8
14	[1,3,5,7,10,9,12]	18, 19,25	Y, n = 48

15	[1,3,5,7,11,9,13,14]	20	Y, n = 208
16	[1,3,5,7,8,9,13,14]	26	Y, n = 206
17	[1,3,5,7,10,9,13,14]	27	Y, n = 205
18	[1,3,5,7,8,9,13,15,9,12]	21,22,28	Y, n = 549

2.8. pow Coverage Criteria

For this method, prime path coverage will be used.

TR ID	Test Requirement
29	[1,2]
30	[1,3,4,5]
31	[1,3,4,6]
32	[1,3,7,8]
33	[1,3,7,9,10,11]
34	[1,3,7,9,10,12]
35	[1,3,7,9,13,14]
36	[1,3,7,9,13,15,16,17,18,19]
37	[1,3,7,9,13,15,16,20,18,19]
38	[16,17,18,21,16]
39	[16,20,18,21,16]
40	[21,16,20,18,21]
41	[21,16,17,18,21]

Test Path ID	Test Path	Covered TR	Feasible?
19	[1,2]	29	Y, e = -1
20	[1,3,4,5]	30	Y, e = 0, k = 0
21	[1,3,4,6]	31	Y, e=3, k=0
22	[1,3,7,8]	32	Y, e=3, k=1
23	[1,3,7,9,10,11]	33	Y, e=4, k = -1
24	[1,3,7,9,10,12]	34	Y, e=3, k= -1
25	[1,3,7,9,13,14]	35	Y, e=42, k=4
26	[1,3,7,9,13,15,16,17,18,19]	36	Y, e=6, k=4
27	[1,3,7,9,13,15,16,20,18,19]	37	Y, e=7, k=4
28	[1,3,7,9,13,15,16,17,18,21,16,17,18,19]	38	Y, e=11, k=4
29	[1,3,7,9,13,15,16,20,19,21,16,20,18,19]	39	Y, e=12, k=4
30	[1,3,7,9,13,15,16,20,19,21,16,20,18,21,16,17,18,21,16,17,18,19]	40,41	N, both TR 40,

			41 would be infeasible because e can never be more than 2 digits

3. Test Results with Traceability

Test ID	Targeted TR	MUT	Observed Output	Result
1	1	isPrime	False	Pass
2	2	isPrime	True	Pass
3	3	isPrime	True	Pass
4	5	isPrime	True	Pass
5	6	apply	IllegalArgument	Fail
6	8	apply	Arrays are arranged based off array x	Pass
7	9	nextPrime	IllegalArgument	Fail
8	10	nextPrime	2	Pass
9	11	nextPrime	3	Pass
10	13	nextPrime	11	Pass
11	14	nextPrime	53	Pass
12	15	nextPrime	211	Pass
13	16	nextPrime	211	Pass
14	17	nextPrime	211	Pass
15	18	nextPrime	557	Pass
16	19	pow	IllegalArgument	Fail
17	20	pow	1	Pass
18	21	pow	0	Pass
19	22	pow	1	Pass
20	23	pow	1	Pass
21	24	pow	-1	Pass
22	25	pow	ArithmeticException	Fail
23	26	pow	4096	Pass
24	27	pow	16384	Pass
25	28	pow	4194304	Pass
26	29	pow	16777216	Pass