is - e
S2. BBT
I. Specify X, f(X) + Z, xsi(x, z)
II. BBT:
- EC
- BVA
+
TC:
- EC
- BVS

III. Implement TC form II -> JUnit

1. Verify if a natural number is prime
I:
input - x: n
f(x):: n is a natural number
output - z: r
xsi(x, z):: [r is {true, false} ^ [r = true if n is prime v r = false if n != prime]]
v error if n != natural

II. BBT - Equivalence class (EC)
TC - test cases

#EC	condition	Valid EC	Invalid EC
1	n is a natural number n is [0, +infinity] MAXINT	n is a natural number	
2	n is a natural number n is [0, +infinity] MAXINT		n < 0 n is not a natural number
3			n is not a natural number n > MAXINT
4	r is {T, F}	r = T	
5		r = F	
	?		

TC based on EC

#TC	EC	Input	Ouput expected	Output actual
1	2	-3	error	
2 (can't instantiate MAXINT + 1)	3	MAXINT + 1	error	
3	4, 1	7	true	
4	5	4	false	

BVA - boundary value analysis

#BVA	condition	#TC	input n	expected r	actual
1	n is a natural number n is [0, MAXINT]	1	n = 0	F	
		2	n = 1	F	
		3	n = -1	error	
		4	n = MAXINT	T	
		5	n = MAXINT + 1	error	
		6	n = MAXINT - 1	F	
2	r = {T/F}	7	7	r = T	
		8	4	r = F	

look at TC and BVA table

EC	BVA
1	1
3	2
4	3
	4
	6
	7 (same as 4)
	8 (same as 4)

= 10 TC - 2 = 8
-EC4/BVA8
-EC3/BVA7

III. JUnit -> TC__EC__3, TC__BVA__1, TC__EC__1

```
public class VerifyIsPrime
{
	public boolean isPrime(int n) throw ValueException
	{ ... }
}

public class UnitTests
{
	public void CheckNumbers()
	{
		assertEquals(isPrime(7) == true);
	}

	public void CheckNumbers2()
	{
		assertEquals(isPrime(0) == false);
	}

	public void CheckNumbers3()
	{
		assertEquals(isPrime(2) == true);
	}

public void CheckNumbers4()
	{
		try
		{
			assertEquals(isPrime(-2) == true);
		}
		catch (ValueException )
		{ ... }
	}
}
```

2) Max sequence of prime numbers.
NOTE: this covers both cases when arrays start from 0 and from 1, wtf????
input x: l, length
	f(x): length >= 0 ^ is a natural number ^ l(i) is a natural number
output z: startPos, endPos
	xsi(x, z): startPos <= endPos ^ startPos <= length ^ endPos <= length ^ startPos >= 0 ^
		endPos >= 0 V if no prime numbers, startPos = -1 ^ endPos = -1
			V if no elements, startPos = -1 ^ endPos = - 1]
		startPos and endPos are natural numbers

#EC	condition	Valid EC	Invalid EC
1	l(i) is a natural number, i = 1, length in the interval [0, MAXINT]	l(i) is a natural number, i = 1, length	
2		-	l(i) < 0, i = 1, length
3		-	li > MAXINT, i = 1, length
4	length >= 0 in the interval [0, MAXINT]	length >= 0	
5		-	length < 0
6		-	length > MAXINT
7	length si a natural number	length is a natural number	
8		-	length < 0 ^ length is not a a natural number
9		-	length > MAXINT ^ length is not a natural number
10	startPos <= endPos	startPos <= endPos	
11		-	startPos > endPos
12	1<= startPos <= length	1 <= startPos <= length	
13		-	startPos < 1
14		-	startPos > length
15	1 <= endPos <= length	1 <= endPos <= length	
16			
17			
18	startPos = -1 ^ endPos = - 1	startPos = -1 ^ endPos = -1 no prime numbers	
19		length = 0	
20	length of result seq > 0	0 elem in the sequence	
21		1 elem in the sequence	
22		length elem	
23		between 1 and length m result	

24	no. of prime sequences	no sequence	
25		only 1 prime seq	
26		more seq with same length	
27		more seq with different length	
28	position where the output seq is	start from the begin	
29		last position	
30		middle position	
31			l(i) is [0, MAXINT], l(i) is not a natural number

We can't test EC's which have MAXINT

#TC	EC	Input (length, l)	Ouput expected	Output actual
1	2	3, [-3, 0, 7]	error	
2	5	-3, [...]	error	
3	8	-5,5, [...]	error	
4	31	3, [1, 2.7, 8]	error	
5	1, 4, 7, 10, 12, 15, 25, 30	6, [4, 8, 5, 7, 13, 14]		3, 5
6	18, 20, 24	3, [8, 10, 12]	-1, -1	
7	19	0, []	-1, -1	
8	21			