Question 1:

It's not advisable to combine multiple test requirements to generate a single test case when considering error conditions. It's possible that multiple inputs to a function may qualify for errors but once one error is raised, other errors that could result from the same inputs are not thrown.

Assume:
$10 < x < 20$,
X >= 20 should throw an exception.
$5 < y < 15$,
Y >= 15 should throw an exception.

Also assume that a tester selects X = 20 and Y = 20 and the code is structured as:

```
if X >= 20 then
  throw ExceptionA()

if Y >= 15 then
   throw ExceptionB()
```

In this case the module would throw ExceptionA() and then terminate. ExceptionB() will not be triggered.

The order of the code would also affect testing in this situation because if the statements were reordered, below, then ExceptionB() would be thrown without being able to generate ExceptionA(). In a black-box scenario, it would be difficult to understand how to structure tests to get the expected results.

```
if Y >= 15 then
   throw ExceptionB()

if X >= 20 then
  throw ExceptionA()
```

Instead of trying to combine two tests to cover the X >= 20 and Y >= 15 partition, two explicit tests where X >= 20 and Y < 15 (ExceptionA) and Y >= 15 and X < 20 (ExceptionB) should be executed.

Question 2:
a.
First we need to identify the boundaries, then we need to identify the nearest left of boundary, exactly at boundary and nearest right of boundary.

So, for 'x', the boundaries are 3 and 7. Similarly for 'y' the boundaries are 5 and 9.

Since the boundary is an integer variable for both 'x' and 'y',
For $3 \leq x$, nearest left would be 2, exactly at 3, slightly right is 4
For $x \leq 7$, nearest left would be 6, exactly at 7 and slightly right is 8
For $5 \leq y$, nearest left would be 4, exactly at 5, slightly right is 6

For $y \leq 9$, nearest left would be 8, exactly at 9 and slightly right 10

Thus, a set of test cases using "boundary value analysis" are:

T1 X=2, Y=4

T2 X=3, Y=5

T3 X=4, Y=6

T4 X=6, Y=8

T5 X=7, Y=9

T6 X=8, Y=10

b.

Unidimensional partitioning is an approach to divide the input domain.

First, we use BVA to create partitions.

We include multiple input values into one partition.

3 partitions for variable X:

X= $(-\infty, 2]$ - This partition includes all the values from -infinite to 2 (outside the range to the left)

X = [3,7] - This partition includes all the values from 3 and 7 (inside the range)

X= $[8, +\infty)$ - This partition includes all the values from 8 to +infinite (outside the range to the right)

Similarly, we have 3 partitions for variable Y:

Y= $(-\infty, 4]$ - This partition includes all the values from -infinite to 4 (outside the range to the left)

Y = [5,9] - This partition includes all the values from 5 and 9 (inside the range)

Y= $[10, +\infty)$ - This partition includes all the values from 10 to +infinite (outside the range to the right)

In unidimensional partitioning, we only care one partition in a test case and do not care about rest of the value.

Therefore, we would need 6 test cases. For example,

T1 x=$(-\infty, 2]$, y = don't care; (x=-100,y=123)

T2 x=[3,7], y = don't care; (x=7,y=123)

T3 x=$[8, +\infty)$, y = don't care; (x=251,y=123)

T4 x=don't care, y = $(-\infty, 4]$; (x=123,y=3)

T5 x=don't care, y = [5,9]; (x=123,y=8)

T6 x=don't care, y = $[10, +\infty)$; (x=123,y=100)

c.

Multi-dimensional partitioning is another approach to divide the input domain.

In this case, we do care of all combinations between all the partitions we made.

The partitions we have are:

X= $(-\infty, 2]$

X = [3,7]

X= [8, +∞)

Y= (-∞, 4]

Y = [5,9]

Y= [10, +∞)

So total test cases we would need is 3 * 3 = 9 test cases since we have 3 partitions for each of the variables. For example,

T1	x= (-∞, 2], y = (-∞, 4]; (x=2,y=3)
T2	x= (-∞, 2], y = [5,9]; (x=-1,y=6)
T3	x= (-∞, 2], y = [10, +∞); (x=0,y=100)
T4	x= [3,7], y = (-∞, 4]; (x=3,y=-2)
T5	x= [3,7], y = [5,9]; (x=4,y=9)
T6	x= [3,7], y = [10, +∞); (x=7,y=10)
T7	x= [8, +∞), y = (-∞, 4]; (x=8,y=4)
T8	x= [8, +∞), y = [5,9]; (x=288,y=5)
T9	x= [8, +∞), y = [10, +∞); (x=10,y=10)