REPORT

Q.1)

a) Test Cases:

- Test for value beyond the lower boundary, i.e -10 : X=-11
- Test for value beyond the upper boundary, i.e 10: X=11
- Test for value at the lower boundary, i.e -10 which is valid: x=-10
- Test for value at the upper boundary, i.e -10 which is valid: x=10
- Test for value within the range -10 to 10 which is valid: x=0

Unit Name	Range function for integer	
Summary	Given an integer as input determine if it is in the range -10<=x<=10	
Input Type(s)	Input Value(s)	Expected Result
int	-11	false
int	11	false
Int	-10	true
Int	10	true
Int	0	true

b) Test Cases:s

- Test for value beyond the lower boundary: X=-10f
- Test for value beyond the upper boundary : X=10f
- Test for value at the lower boundary, i.e -10 which is valid: x=-10.01f
- Test for value at the upper boundary, i.e -10 which is valid: x=10.01f
- Test for value beyond the ranges: x=0.0f
- Test for value within the range from 10 which is valid: x = Float.MAX_VALUE

Unit Name	Range function for float	
Summary	Given an integer as input determine if it is in the range x<-10 or x>10	
Input Type(s)	Input Value(s)	Expected Result
float	-10f	false

float	10f	false
float	-10.01f	true
float	10.01f	true
float	0.0f	false
float	Float.MAX_VALUE	true

c) Test Cases:

- Test for input string length>10 : "61776767511"
- Test for input string length<10 : "617767675"
- Test for input string length=10 and valid: "5567676750"
- Test for input string length = 10 and starts with 0: "0617767675"
- Test for input string length = 10 and starts with 1: "1617767675"
- Test for string with length 10 but starts with 555: "5556767675"
- Test for string with length 10 but with -, valid : "(555)676-7675"
- Test for invalid string length 10 : "TestTest"
- Test for empty string or input left blank: “”
- Test for null string: null
- Test for string with length 10 but filled with spaces: “ “
- Test for string with special characters: "6177!@67#0"

Unit Name	Check phone number	
Summary	Given an string as input determine if it is a valid phone number	
Input Type(s)	Input Value(s)	Expected Result
String	"61776767511"	false
String	"617767675"	false
String	"5567676750"	true
String	"0617767675"	false
String	"1617767675"	false
String	"5556767675"	false
String	"(555)676-7675"	true
String	"TestTest"	false
String	null	false
String	"6177!@67#0"	false

Q.2)

Boundary conditions are highlighted:

Unit Name	Sqrt function	
Summary	Given a non-negative number as input, should return its square root. If input is invalid integer, throw a invalid input exception	
Input Type(s)	Input Value(s)	Expected Result
int	**0**	**0**
int	1	1
int	+1	1
int	004	2
int	**-1**	**Invalid argument exception**
int	**MAX_INT**	**46340.950001051984**

Unit Name	Square function	
Summary	Given an integer as input, should return its square as an integer. If input is not an integer throw an invalid input exception. If result of input is beyond infinity throw overflow exception	
Input Type(s)	Input Value(s)	Expected Result
int	**0**	**0**
int	1	1
int	-1	1
int	002	4
int	MAX VAL	Overflow exception
int	MIN VAL	Overflow exception
int	**46341**	**Overflow exception**
int	**-46341**	**Overflow exception**
int	**46340**	**2147395600**
int	**-46340**	**2147395600**

Unit Name	Factorial	
Summary	Given an integer as input, should return its factorial. If the input is not a valid positive integer, raise an invalid input exception	
Input Type(s)	Input Value(s)	Expected Result
int	**0**	**1**
int	1	1
int	**MAX_INT**	**Overflow exception**
int	**-1**	**Invalid argument exception**
int	**MIN_INT**	**Invalid argument exception**
Int	002	2
Int	+1	1
Int(-ve)	-1	Invalid argument exception

Q.3)

The given test specification seems wrong to me because it does not cover all the boundary conditions.

For eg. The boundary for MAX_INT is not checked. This can be done by adding the test case for input

(0,MAX_INT) which should give MAX_INT as the result.

Also the lower boundary can be checked with input (0, MIN_INT) which should give MIN_INT as the result.

The specifications also do not test if the one of the input is MIN_INT and the other is <0, hence the result would be <MIN_INT and should give an exception.

(-1,MIN_INT) should give an underflow exception.

The test case for (1, -MAX_INT) is wrong, since the expected output is misleading. It should

Also, the summary does not specify clearly what will happen if a condition fails, for eg. It should give an exception if the input is not an integer.

Unit Name	Adder	
Summary	Takes two integers as input and returns their sum (as an integer). If the result is not in the range, it should throw an exception	
Input Type(s)	Input Value(s)	Expected Result
int	1,1	2
int	5, 35	40
int	100,936, 1,327,927	1,428,863
int	-3, -5	-8
int	-100,936, -1,327,927	-1,428,863
int	1, MAX_ INT	Overflow Exception
int	1, -MAX_INT	-2147483646
int	0,MAX_INT	MAX_INT
int	0,MIN_INT	MIN_INT
int	-1,MIN_INT	Underflow exception

Also, the test case for (1,MAX_INT) seems to be correct if we do not consider java as the sole language for the add function. In case of java, the addition of 1 to MAX-INT gives MIN_INT and not an overflow exception. Here, I am considering that the program is not platform specific and should throw an overflow exception in such a case.

Q.4

a)

- The test coverage including the statement and branch coverage for Sorter1 is in the csv file Sorter1TestCoverage.csv under HW4 folder.

  The Class QuickSorter, Person and Sorter1Tests are the classes pertinent to this coverage run and shows 100% branch and statement coverage for those classes.

  The number of branches covered is 16 and branches missed is 0. The number of lines covered in 38 and lines missed is 0 for QuickSorter.

  The number of lines covered is 5 and lines missed is 0 for Person class.

- The test coverage including the statement and branch coverage for Sorter2 is in the csv file Sorter2TestCoverage.csv under HW4 folder.

The Class HeapSorter, Person and Sorter2Tests are the classes pertinent to this coverage run and shows 100% branch and statement coverage for those classes.

The number of branches covered is 18 and branches missed is 0. The number of lines covered in 32 and lines missed is 0 for QuickSorter.

The number of lines covered is 5 and lines missed is 0 for Person class.

- The test coverage including the statement and branch coverage for SampleTest for QuickSort is in the csv file SampleTests1Coverage.csv under HW4 folder.

  The Class QuickSorter, Person and SampleTests1 are the classes pertinent to this coverage run and shows 100% branch coverage but <100% statement coverage for the Person class due to the compareTo method of Person not being called. This has been covered by adding more tests in the next section by modifying the Sample Tests and including it in package test3d.

  The number of lines covered is 4 and lines missed is 1 for Person class.

- The test coverage including the statement and branch coverage for SampleTest for HeapSort is in the csv file SampleTests2Coverage.csv under HW4 folder.

  The Class HeapSorter, Person and SampleTests1 are the classes pertinent to this coverage run and shows 100% branch coverage and 100% statement coverage.

Since the test cases in Q.4 had 100% coverage for branch and statement, I have not added new tests cases for the tests3c package.

While running the Sample Tests, sorter was taking a lot of time more than an hour and half for test with Large Array due to memory limitations in my system. Hence, that test has been modified for the purpose of running.