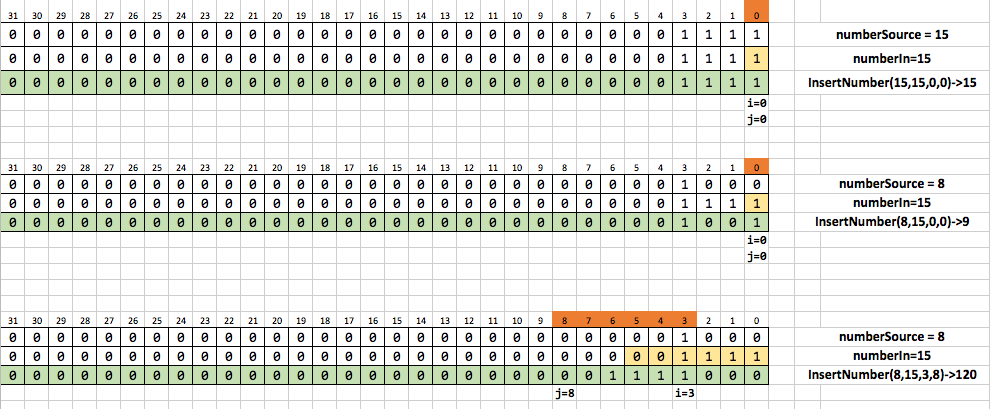
1. Даны два целых знаковых четырех байтовых числа и две позиции битов i и j (i<j). Реализовать алгоритм **InsertNumber** вставки битов с j-ого по i-ый бит одного числа в другое так, чтобы биты второго числа занимали позиции с бита j по бит i (биты нумеруются справа налево). Разработать модульные тесты (**NUnit и(!) MS Unit Test**) для тестирования метода. (Ниже пояснение к алгоритму)



1. Реализовать метод **FindNextBiggerNumber**, который принимает положительное целое число и возвращает ближайше наибольшее целое, состоящее из цифр исходного числа, и - 1 (null), если такого числа не существует. Разработать модульные тесты (NUnit **или** MS Unit Test) для тестирования метода.

Примерные тест-кейсы

[TestCase(12, ExpectedResult = 21)]

[TestCase(513, ExpectedResult = 531)]

[TestCase(2017, ExpectedResult = 2071)]

[TestCase(414, ExpectedResult = 441)]

[TestCase(144, ExpectedResult = 414)]

[TestCase(1234321, ExpectedResult = 1241233)]

[TestCase(1234126, ExpectedResult = 1234162)]

[TestCase(3456432, ExpectedResult = 3462345)]

[TestCase(10, ExpectedResult = -1)]

[TestCase(20, ExpectedResult = -1)]

1. Добавить к методу **FindNextBiggerNumber** возможность вернуть время нахождения заданного числа, рассмотрев ***различные языковые возможности****.* Разработать модульные тесты (NUnit **или** MS Unit Test) для тестирования метода.
2. Реализовать метод **FilterDigit** который принимает список целых чисел и фильтрует список, таким образом, чтобы на выходе остались только числа, содержащие заданную цифру. ***LINQ не использовать!*** Например, для цифры 7, FilterDigit (**7**,1,2,3,4,5,6,7,68,69,70, 15,17) -> {7, 70, 17}. Разработать модульные тесты (NUnit **или** MS Unit Test) для тестирования метода.
3. Реализовать алгоритм **FindNthRoot**, позволяющий вычислять корень n-ой степени ( n∈N ) из числа ( a∈R ) методом Ньютона с заданной точностью. Разработать модульные тесты (NUnit и (или) MS Unit Test) для тестирования метода.

[TestCase(1, 5, 0.0001,ExpectedResult =1)]  
 [TestCase(8, 3, 0.0001,ExpectedResult = 2)]  
 [TestCase(0.001, 3, 0.0001,ExpectedResult = 0.1)]  
 [TestCase(0.04100625,4 , 0.0001, ExpectedResult =0.45)]  
 [TestCase(8, 3, 0.0001, ExpectedResult =2)]  
 [TestCase(0.0279936, 7, 0.0001, ExpectedResult =0.6)]

[TestCase(0.0081, 4, 0.1, ExpectedResult =0.3)]  
 [TestCase(-0.008, 3, 0.1, ExpectedResult =-0.2)]  
 [TestCase(0.004241979, 9, 0.00000001, ExpectedResult =0.545)]

-----------------------------

[TestCase(8, 15, -7, -5)]// <-ArgumentOutOfRangeException  
 [TestCase(8, 15, -0.6, -0.1)]// <-ArgumentOutOfRangeException

public void *MethodName*\_Number\_Degree\_Precision\_ArgumentOutOfRangeException(double number, int degree,  
 double precision, double expected)  
 {  
 Assert.Throws<ArgumentOutOfRangeException>(() => *ClassName*.*MethodName*(number, degree, precision));  
 }