Exercise x2: Measuring Execution Times, Primes, and Rational Numbers

Measuring Execution Times

1. *Programs A and B are analyzed and are found to have worst-case running times no greater than 150 N log N and N^2 , respectively. Answer the following questions, if possible:*

   1.1. *Which program has the better guarantee on the running time for large values of N (N > 10 000)?*

   150 N log N is growing slower than N², so A is better.

   1.2. *Which program has the better guarantee on the running time for small values of N (N < 100)?*

   N² would be at least 10.000, for 150 N log N it will be 15.000 log 100. So the program b is better in this case.

   1.3. *Which program will run faster on average for N = 1000?*

   150.000 log 1.000 contra 1.000.000 150.000 * log 1000 = 150.000 * 6,90776 = 1.036160, so program B is still better

   1.4. *Is it possible that program B will run faster than program A on all possible inputs?*

   No, the first program is better for large numbers, because it's growing slower. There should be turning point between 1.000 and 1.500: 150.000 * log 1.500 = 150.000 * 7,31322 = 1.096.980 and for program B it would be 2.250.000.

2. *An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following:*

   2.1. linear = 5 * 0.5 ms = 2.5 ms

   2.2. O (N log N) = (5 log 5) * 0.5 ms = 4,02 ms

   2.3. quadratic = 5 * 5 * 0.5ms = 12,5 ms

   2.4. cubic = 5 * 5 * 5 * 0.5ms = 62,5 ms

3. *An algorithm takes 0.5 ms for input size 100. How large a problem can be solved in 1 min if the running time is the following:*

   3.1. linear = 60 seconds have: 60 * 100 = 6.000ms; 6.000ms * 2 * 100 = 1.200.000

   3.2. O (N log N) = 746.269

   3.3. quadratic = 240.000

   3.4. cubic = 48.000

4. *Order the following functions by growth rate, and indicate which, if any, grow at the same rate.:*
   N, square root of N, $N^{1.5}$, N^2 , N log N, N log log N, N $\log^2$ N, N log (N^2), 2/N, 2N, 2N/2, 37,N^3, N^2 log N.

2/N, 37, square root of N, N = 2N/2, 2N, $N^{1.5,}$ N log N, N^2, N^2 log N, $N^{3,}$ N (log (log N)), N $\log^2$ N, N log (N^2)

Big-Oh analysis

5. *For each of the following six program fragments, do the following:*
   - 5.1. *Give a Big-Oh analysis of the running time (you can do this before you come to lab!)*
   - 5.2. *Implement the code in a simple main class and run it for several interesting values of N.*
   - 5.3. *Compare your analysis with the actual running times for your report.*

```
// Fragment #1
for ( int i = 0; i < n; i ++)
    sum++;
// Fragment #2
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < n; j ++)
        sum++;
// Fragment #3
for ( int i = 0; i < n; i ++)
    sum ++;
    for ( int j = 0; j < n; j ++)
        sum ++;
// Fragment #4
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < n*n; j ++)
    sum++;
// Fragment #5
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < i; j ++)
        sum++;
// Fragment #6
for ( int i = 1; i < n; i ++)
    for ( int j = 0; j < n*n; j ++)
        if (j % i == 0)
           for (int k = 0; k < j; k++)
               sum++;
```

Loop Calculation

23.11.2006 @ 09:45 – 12:00 / On a sheet of paper we try to find out what the given loops will do. The results of the brainstorming:

Fragment 1: O(N)
Fragment 2: O(N²)
Fragment 3: O(Ni) + O(Nj) -> O(N²) + N
Fragment 4: O(Ni) * O(Nj²)
Fragment 5: O(Ni) * O(Nj) -> Sum(N) -> O(N)
Fragment 6: O(Ni) * O(Nj²) * O(Nk) -> O(N^4)

Right after with Eclipse life gets easier because it calculates the algorithms for us. Our suggestions are okay but not correct with every fragment. Put an eye on our class LoopCalculator.

We were very suprised that the fragment 6 needs such long time. But after a short calculation it was clear that there will be some days and year to run it with large numbers.

Prime numbers

6. A prime number has no factors besides 1 and itself. Do the following:

a. *Write a program to determine if a positive integer N is prime.*
   In terms of N, what is the worst-case running time of your program?
b. *Let B equal the number of bits in the binary representation of N. What is the value of B?*
c. *In terms of B, what is the worst-case running time of your program?*
d. *Compare the running times needed to determine if a 20-bit number and a 40-bit number are prime by running 100 examples of each through your program. Report on the results in your lab report.*

Prime checker class

23.11.2006 @ 13:15 – 13:40 / We start thinking of a way to find out whether it is a prime number? First we decide to get rid of number that can be devided by 2 - modulo helps us. So after this we heard about the little trick to check only to the squareroot of the number to search for. We implemented it and leaved the class with a method to check for prime.

Also calculating B – the number if bits in binary representation – was not that difficult, so we switched to c. But we also recognized that the way we did it would maybe to easy. So we tried to implement our own, but we didn't use it in the implementation.

For comparing the running-times we end up extending the whole code for long values. We also found out how to define a long in Java and got it to run. The result was interesting, but in a way we imagined it:

```
Prime check and bit-conversion of 100 20-bit-numbers took us 110 milliseconds.
Prime check and bit-conversion of 100 40-bit-numbers took us 651 milliseconds.
```

The worst case of running the programm would be for the biggest prime-number in the 40bit-representation, because it would need the maximum of divisions for prime-check. And because we would take part in the prime-searching-contest we also defined a method primeIt() which calculates the prime-numbers between 0 and a value free to choose. So: 9592 prime found from 0 to 100000. But we know that we should improve it with more mathematical method.

Rational numbers

7. A rational number can be expressed as a fraction numerator / denominator. A class for modelling rational numbers could look like this:

```
public class Ratio{
// For creating an object that can store a rational number.
// A rational number can be expressed as a fraction
// numerator / denominator
  protected int numerator;
  protected int denominator;
  public Ratio (int top, int bottom)
  // pre:  bottom != 0
  // post: constructs a ratio equivalent to top / bottom
  {
     numerator = top;
     denominator = bottom;
  }
  public int getNumerator()
  // post: returns the numerator of the fraction
  {
     return numerator;
  }
  public int getDenominator()
  // post: returns the denominator of the fraction
  {
     return denominator;
  }
  public double value()
  // post: returns the value of the fraction as a real number
  {
     return (double) numerator / (double) denominator;
  }
```

a. *Implement a class for this data type that supports standard math operations: addition, subtraction, multiplication and division. Offer a `print()` method that returns a nicely formatted `String`. You should also be able to construct `Ratios` from either a numerator-denominator pair, or a single integer, or with no parameter as all (what would a reasonable default value be?). You are going to have to test your `Ratio`, so think about good test cases and implement a test harness so that you can run the tests.*
b. *If you get that done before class is out, add in relational methods for testing equality, inequality, greater than and less than.*

Playing around with the ration-class

23.11.2006 @ 11:10 – 13:15 / We are going to try out the Ratio class. Setting up it as a class leads us to the idea to instantiate it in a second class to test it's behaviour. First problem occurs: when we try to read in two values from the console it terminates the programme after entering the first value. We will find a way to come around this.

Second problem: the System.in.read() produces some strange input. As we type in 6 it stores 54 into our variable. - 20 minutes later! - We cannot find a quick way to solve the problem although it seems to be identified: By pressing enter the console stores the integer value of 'carriage return' which is 13. - (45 minutes break for lunch.)

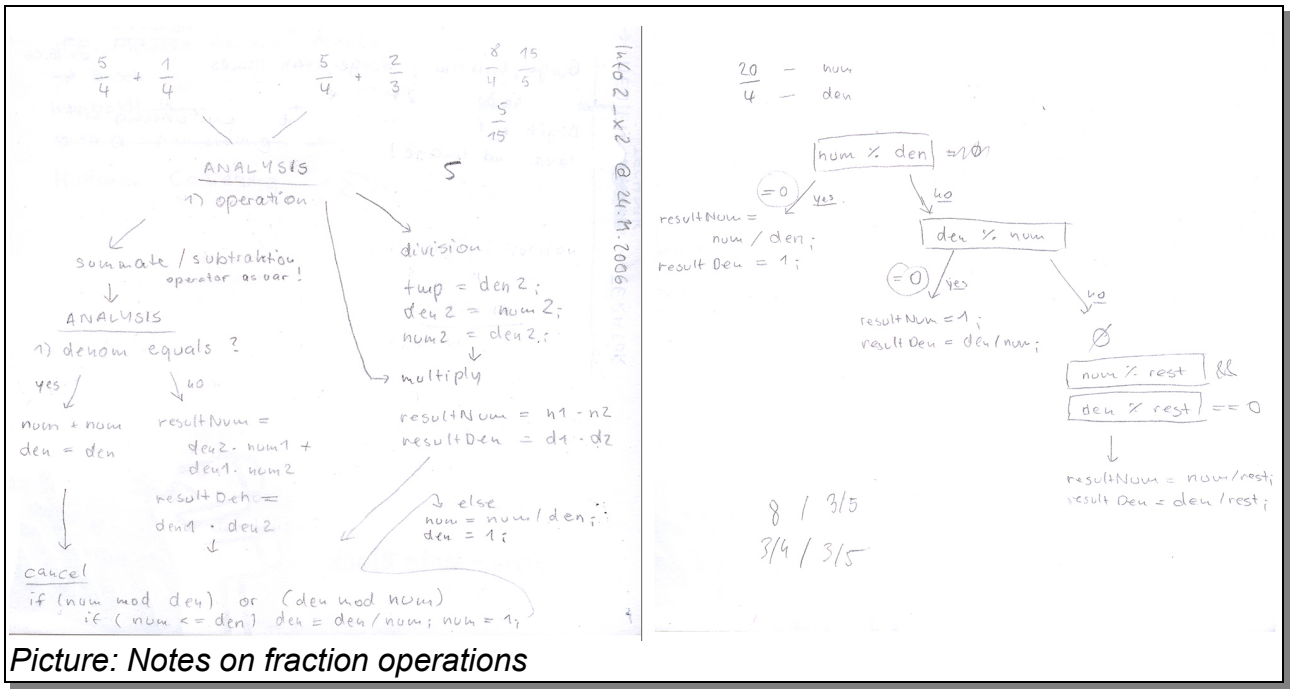
We got it running. Maria helped us by implementing a String to integer algorithm. After we put this in a separate method calling it inputConverter().

Subversion and Trac

24.11.2006 / Today we set up Subversion. It was easy, because we had some experiences. It's nice to have the combination of Subversion and Trac. So now you can find our code online: https://projekte.f4.fhtw-berlin.de/trac/s0517512-tasks/browser. The access to browse the code is granted to anonymous, so you don't have to login. We will include the url in our documents, so it should be easy to browse the code.
It's nice now. We can work independent, but also in a team. It's not that easy to meet during the week. So now it should be better. Thanks to the FHTW offering this possibility.

Implementing mathematical operations

24.11.2006 @ 10:24 – 11:40 / This morning we work on part 7 that deals with mathematical operations on ratios. We already built up a test class to check out the behaviour that the Ratio-class includes.
The first idea is to set up a separate class named RatioOperator that will do all the work: multiply, summate. The decision on which method will be used could be made on the operator supplied by the calling statement. It feels as if we can use an interface to supply the methods - but it's still quite blurry.
As we built up the first method to summate two fractions we notice that we need to think more complex then estimated. It is not just adding the numerators to each other and then - "yes what about the denomerators...!?" We need to implement calculations based on how the fractions relate to each other.
Full stop. We doing some brainwriting now!



Picture: Notes on fraction operations

24.11.2006 @ 14:30 – 18:20 / We meet again to do further work on the project: We try to code what we figured out on a sheet of paper.
15:14 / As we write the different methods like multiply and division one problem occurs. The result of the calculation are two values which need to be returned. The workaround that helps us is to set up fields and to store the return values up there. - To test the implementation we instantiate the RatioOperator as myRO and access the resultNumerator resp. resultDenomerator.

Working on the Calculator

My task is today to write a parse of a String to extract the values of the numerators, denumerators and operators. So I first check if there is a nice possibility to do this with regular expressions.

After checking some website of experts I found that there is a nice String.split()-method which makes the work with the regular expressions not that difficult. So I'll parse the string through some split-method-calls and with the nice Integer.parseInt()-function.

Well after some hours of coding we have a nice class which takes a string in the format (-)XX(/(-)XX) (+|-|*|/|=|>|<) (-)XX(/(-)XX) and extracts all variables we need. Also the operation is extracted and to converted in an integer value. We also handle negative values and a non fractional input like 15 > 16. Just try it out.

Connecting classes

17:18 / We put together both parts that we programmed the last hours. Let's see if it can destroy our computer.
First run works but calculates a wrong result: 15/15 + 15/15 = 1/1. We check the summate-method: it returns 2/1. Further we deactivate the cancel-method that we don't trust in fully - but the same result are calculated.

18:05 / Yeah! "Break It!" After getting around with the debugmode of eclipse we manage to watch our main variables. Everything went fine until we saw that the programme runs into a second calculation because there is no 'break' in the switch-case block! Put it in!

Adding a subtract-method

25.11.2006 @ 20:41 – 21:20 / Let's build up the missing subtract method. The idea is to use the summate-method. We change the numerator of the second fraction into 'minus' by multipling it with (-1).
The first test shows a problem of the cancel method. There is a division by zero happening which leads us to java.lang.ArithmeticException. We need to fix this. - The subtraction itself works fine by now.
While testing out it seems to be a better decision to check if the numerator equals zero before calling the cancel method. We leave the first idea uncommented in the cancel-method anyway.

Download / SourceCode / JavaDoc / Report

You find our code browseable at the trac at:

https://projekte.f4.fhtw-berlin.de/trac/s0517512-tasks/browser

The Java doc you find at:

http://home.fhtw-berlin.de/~s0516424/info2/x2/doc/index.html

The zipped class-files are available at:

http://home.fhtw-berlin.de/~s0516424/info2/x2/Info2_x2.zip

Everything is also available here:

http://home.fhtw-berlin.de/~s0516424/info2/info2.htm