Markus Arendt (538148)
Joshua Widmann (537813)

INFO 2 Laboratory

30th October 12

Laboratory 3: Execution Times

(http://people.f4.htw-berlin.de/~weberwu/info2/labs/Exer3.shtml)

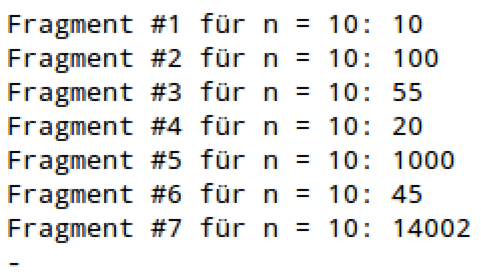
1. For each of the following seven program fragments, do the following:

1. Give a Big-Oh analysis of the running time (you can even do this before you come to lab!)

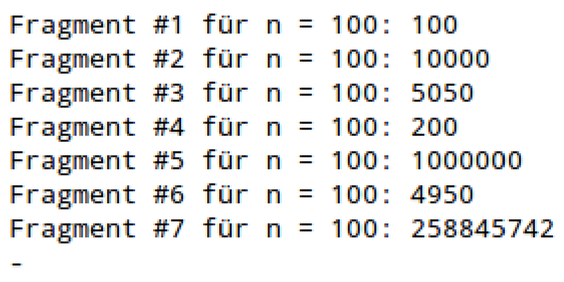
Fragment #1: O(N)
Fragment #2: O(N^2)
Fragment #3: log(N)
Fragment #4: O(N^2) + N
Fragment #5: O(N) * O(N^2) -> O(N^3)
Fragment #6: ?
Fragment #7: O(N^4)

2. Implement the code in a simple main class and run it for several interesting values of N. What are interesting values?

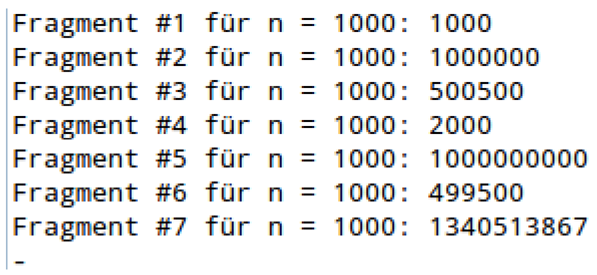
We chose 10, 100, 1000 and 10000 for N. At N=10000 Fragment #5 got overflowed and Fragment #7 never finished calculating.
Here are all results:



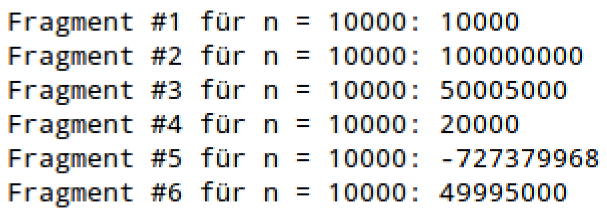
```
Fragment #1 für n = 10: 10
Fragment #2 für n = 10: 100
Fragment #3 für n = 10: 55
Fragment #4 für n = 10: 20
Fragment #5 für n = 10: 1000
Fragment #6 für n = 10: 45
Fragment #7 für n = 10: 14002
-
```



```
Fragment #1 für n = 100: 100
Fragment #2 für n = 100: 10000
Fragment #3 für n = 100: 5050
Fragment #4 für n = 100: 200
Fragment #5 für n = 100: 1000000
Fragment #6 für n = 100: 4950
Fragment #7 für n = 100: 258845742
-
```



```
Fragment #1 für n = 1000: 1000
Fragment #2 für n = 1000: 1000000
Fragment #3 für n = 1000: 500500
Fragment #4 für n = 1000: 2000
Fragment #5 für n = 1000: 1000000000
Fragment #6 für n = 1000: 499500
Fragment #7 für n = 1000: 1340513867
-
```



```
Fragment #1 für n = 10000: 10000
Fragment #2 für n = 10000: 100000000
Fragment #3 für n = 10000: 50005000
Fragment #4 für n = 10000: 20000
Fragment #5 für n = 10000: -727379968
Fragment #6 für n = 10000: 49995000
```

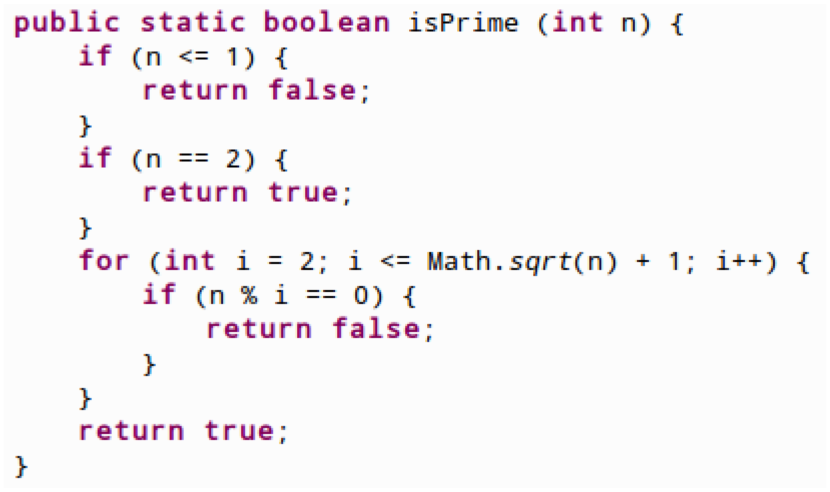
3. Compare your analysis with the actual number of steps (i.e. the value of sum after the loop) for your report.

Fragment #1 seems to be linear as guessed.
Fragment #2 seems to be quadratic as guessed.
Fragment #3 seems to be quadratic in a lower increasing way.
Fragment #4 seems to be also linear but its increasing is doubled: O(2N).
Fragment #5 seems to be cubic as guessed.
Fragment #6 seems also quadratic.
Fragment #7 is a bit tricky and is not exactly N^4 but seems to be very similar

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

a. Write a simple method `public static bool isPrime (int n) {...}` to determine if a positive integer N is prime.

We create a new method:



```java
public static boolean isPrime (int n) {
    if (n <= 1) {
        return false;
    }
    if (n == 2) {
        return true;
    }
    for (int i = 2; i <= Math.sqrt(n) + 1; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

Then we test the method with different numbers. It works fine!

b. In terms of N, what is the worst-case running time of your program?

The worst-case running time is if the number is nor smaller, equal than 1 or equal than 2 and if it is a very very big prime number.

c. Let B equal the number of bits in the binary representation of N. What is relationship between B and N?

If we pick for example our 10000 we need 14 bits to represent N. Because 13 bits can represent a number up to 8192 and 14 bits up to 16384. These numbers are calculated with the power of 2 (2^x; where x stands for the number of bits). To do this calculation backwards, we need to use the logarithm with the base of 2. Since this calculates floating-point numbers we always need to add +1 to N to get the right amount of bits needed. The final formula is:

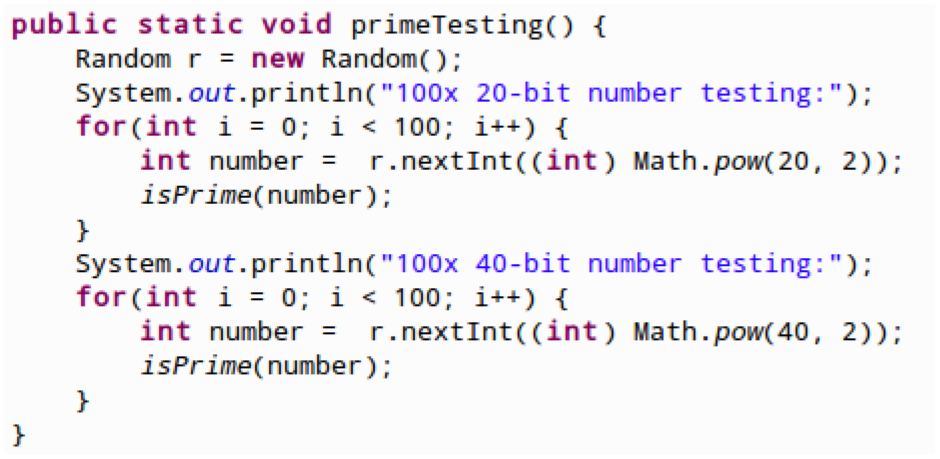
B = log2(N) + 1

d. In terms of B, what is the worst-case running time of your program?

log_2(N)

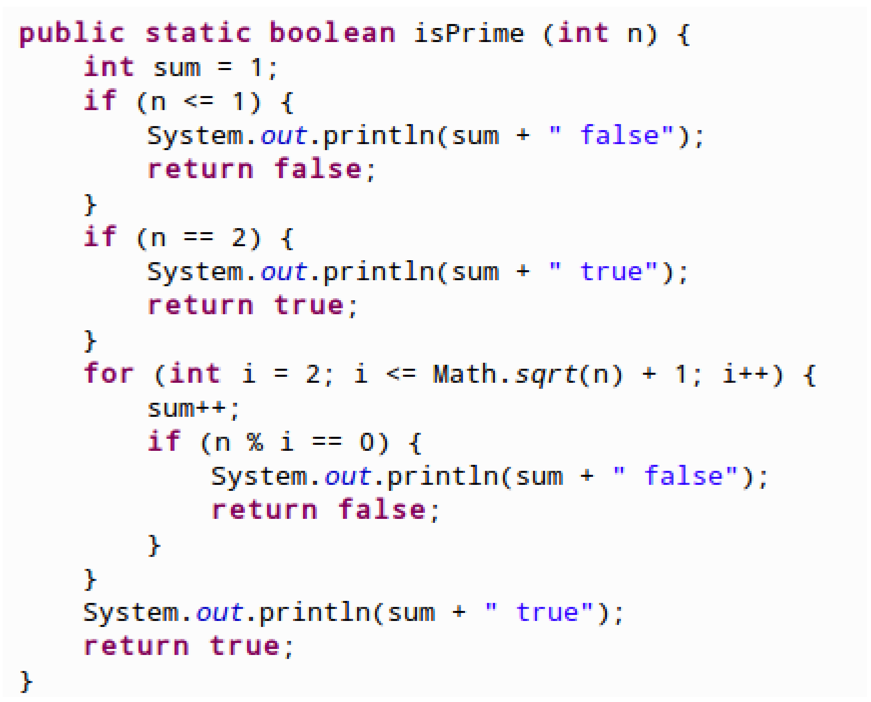
e. Compare the running times needed to determine if a 20-bit number and a 40-bit number are prime by running 100 examples of each through your program. Report on the results in your lab report. You can use Excel to make some diagrams if you wish.

For this task we created a testing method, which looked like thhis:



```java
public static void primeTesting() {
    Random r = new Random();
    System.out.println("100x 20-bit number testing:");
    for(int i = 0; i < 100; i++) {
        int number =  r.nextInt((int) Math.pow(20, 2));
        isPrime(number);
    }
    System.out.println("100x 40-bit number testing:");
    for(int i = 0; i < 100; i++) {
        int number =  r.nextInt((int) Math.pow(40, 2));
        isPrime(number);
    }
}
```

We create 100 random 20-bit numbers and pass them to the isPrime() method, than we do the same with 40-bit numbers. So that we can see wether it was a prime number or not and to see how long it took the method to get a solution, we modified our isPrime() method with print outs like this:



```java
public static boolean isPrime (int n) {
    int sum = 1;
    if (n <= 1) {
        System.out.println(sum + " false");
        return false;
    }
    if (n == 2) {
        System.out.println(sum + " true");
        return true;
    }
    for (int i = 2; i <= Math.sqrt(n) + 1; i++) {
        sum++;
        if (n % i == 0) {
            System.out.println(sum + " false");
            return false;
        }
    }
    System.out.println(sum + " true");
    return true;
}
```

100x 20-bit number testing:

```
2 false   3
          false
5 true    2
          false
2 false   2
          false
2 false   2
          false
2 false   2
          false
20 true   2
          false
2 false   18
          true
2 false   17
          true
11        2
false     false
7 false   2
          false
2 false   9 true
2 false   2
          false
2 false   2
          false
2 false   2
          false
2 false   2
          false
2 false   2
          false
20 true   2
          false
2 false   11
          true
2 false   14
          true
5 false   2
          false
6 true    3
          false
2 false   2
          false
2 false   10
          true
7 true    3
          false
2 false   2
          false
3 false   5 true
2 false   2
          false
5 false   2
          false
7 false   2
          false
16 true   9 true
6 true    3
          false
9 true    2
          false
2 false   2
          false
5 false   2
          false
2 false   3
          false
14 true   2
          false
```

```
2 false   2
          false
2 false   2
          false
2 false   12
          true
2 false   2
          false
2 false   2
          false
15 true   3
          false
5 false   3
          false
3 false   6 true
16 true   2
          false
2 false   2
          false
2 false   13
          false
3 false   2
          false
2 false   17
          true
2 false   5
          false
```

100x 40-bit number testing:

```
5 false   19
          false
2 false   2 false
19        2 false
false
2 false   6 true
2 false   2 false
3 false   2 false
20 true   2 false
3 false   3 false
5 false   5 false
2 false   19
          false
5 false   11 true
17        2 false
false
2 false   3 false
32 true   2 false
3 false   5 false
2 false   3 false
2 false   2 false
2 false   17
          false
2 false   2 false
33 true   35 true
21 true   17
          false
17        29 true
false
3 false   3 false
2 false   2 false
2 false   2 false
2 false   2 false
5 false   2 false
1 true    2 false
7 false   2 false
27 true   2 false
26 true   5 false
5 false   3 false
5 false   7 true
2 false   23
          false
```

```
3 false   3 false
3 false   5 false
2 false   2 false
2 false   2 false
2 false   2 false
2 false   3 false
2 false   3 false
2 false   2 false
27 true   2 false
3 false   3 false
2 false   10 true
2 false   2 false
7 false   2 false
2 false   2 false
11        2 false
false
7 false   2 false
```

The lab taught us to analyze algorithms and to deal with the big-oh notation.
All in all it took us 3 hours of work.

<u>Source code:</u>

Class ExTimes:

```
import java.util.Random;

public class ExTimes {

	public ExTimes() {

	}

	public static void main(String[] args) {
	primeTesting();
```

```
/*
int n = 100;
int sum = 0;

// Fragment #1
for ( int i = 0; i < n; i ++)
    sum++;

System.out.println("Fragment #1 für n = " + n + ": " + sum);
sum = 0;

// Fragment #2
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < n; j ++)
        sum++;

System.out.println("Fragment #2 für n = " + n + ": " + sum);
sum = 0;

// Fragment #3
for ( int i = 0; i < n; i ++)
    for ( int j = i; j < n; j ++)
        sum++;

System.out.println("Fragment #3 für n = " + n + ": " + sum);
sum = 0;


// Fragment #4
for ( int i = 0; i < n; i ++)
    sum ++;
    for ( int j = 0; j < n; j ++)
        sum ++;

System.out.println("Fragment #4 für n = " + n + ": " + sum);
sum = 0;


// Fragment #5
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < n*n; j ++)
    sum++;

System.out.println("Fragment #5 für n = " + n + ": " + sum);
sum = 0;

// Fragment #6
for ( int i = 0; i < n; i ++)
    for ( int j = 0; j < i; j ++)
        sum++;

System.out.println("Fragment #6 für n = " + n + ": " + sum);
sum = 0;

// Fragment #7
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++;

      System.out.println("Fragment #7 für n = " + n + ": " + sum +
"\n-\n");
      sum = 0;
      */
      }

      public static void primeTesting() {
      Random r = new Random();
      System.out.println("100x 20-bit number testing:");
      for(int i = 0; i < 100; i++) {
            int number =  r.nextInt((int) Math.pow(20, 2));
            isPrime(number);
      }
      System.out.println("100x 40-bit number testing:");
      for(int i = 0; i < 100; i++) {
            int number =  r.nextInt((int) Math.pow(40, 2));
            isPrime(number);
      }
      }

      public static boolean isPrime (int n) {
      int sum = 1;
      if (n <= 1) {
            System.out.println(sum + " false");
            return false;
        }
        if (n == 2) {
            System.out.println(sum + " true");
            return true;
        }
        for (int i = 2; i <= Math.sqrt(n) + 1; i++) {
            sum++;
            if (n % i == 0) {
                System.out.println(sum + " false");
                return false;
            }
        }
        System.out.println(sum + " true");
        return true;
      }
}
```