B1 Informatics 2

Lab-Report

Exercise 02:

Leon Romanenko (s0592041),
Maksym Pidluzhnyi (s592916)

Assignments

0. (not graded, but you should still do that)

0.1 Move your implementation of rnd from Lab 1's SortRandom class to Randomness and make the SortTest class use Randomness rather than SortRandom. You also may want to move SortTest.randomInts to Randomness (it's a generally useful method).

```
public interface Randomness
{
    public static final Random random = new Random();

    public default int rnd(int min, int range)
    {
        return min + random.nextInt(range);
    }

    private int[] randomInts(int size, int min, int range){
        int[] a = new int[size];
        for (int i = 0; i < size; i++)
            a[i] = rnd(min, range);
        return a;
    }
    ...
}
```

and **SortTest** implements now **Randomness**

```
public class SortTest implements Randomness
{
    private ArrayList<SortAlg> algs;
    ...
}
```

0.2 Similarly, move your implementation of SortTest.log to the Arithmetic class and make SortTest use that method instead.

```
public interface Arithmetic
{
   private static int log(int n)
   {
       return (int) Math.ceil((Math.log(n) / Math.log(2)));
   }
}
```

1.

1.1 Write a method boolean isPrime(int n) that returns true if n is a prime number and false otherwise.

1.2 Test the method, by adding suitable test code to Main.java.

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

```java
public void run(String[] args)
{
    // Test the log method
    int logResult = Arithmetic.log(n:16);
    System.out.println("log(16) = " + logResult); // Expected: 4

    // Test the isPrime method
    boolean isPrimeResult = Arithmetic.isPrime(n:17);
    System.out.println("isPrime(17) = " + isPrimeResult); // Expected: true

    isPrimeResult = Arithmetic.isPrime(n:18);
    System.out.println("isPrime(18) = " + isPrimeResult); // Expected: false>
}
```

```
log(16) = 4
isPrime(17) = true
isPrime(18) = false
```

1.3 What is the asymptotic complexity of your method? Explain your answer briefly.

O(n)

Because we need to iterate till the end of the array of n elements in the worst case.

1.4 Can you improve your method to make it run faster?. What is the best asymptotic complexity you can achieve?

Our math knowledge is too poor for it, but we googled a bit and found that we can iterate only till Math.sqrt(n), that will give 0(sqrt(n))

```
for(int i = 2; i <=Math.sqrt(n); i++)
```

Combining this approach with a recursive implementation it could look like this. This code was generated with help of the Copilot Assistant

```
    public static boolean isPrime(int n) {
        return isPrimeHelper(n, 2);
    }

    /**
     * Helper method for the recursive isPrime check.
     */
    private static boolean isPrimeHelper(int n, int divisor) {
        if (n <= 1) {
            return false;
        }
        if (divisor > Math.sqrt(n)) {
            return true;
        }
        if (n % divisor == 0) {
            return false;
        }
        return isPrimeHelper(n, divisor + 1);
    }
```

2.

2.1 Write a method int sumOfMultiplesOn(int n) that calculates the sum of all positive integers i ≤ n which are divisible by at least one of 3 or 5. This version should simply loop over the candidate i.

```
public static int sumOfMultiplesOn(int n){
       int sum = 0;
       for(int i = 0; i <= n; i++){
           if(i%3 ==0 || i%5 == 0){
               sum = sum + i;
           }
       }
       return sum;
}
```

2.2 Write a method int sumOfMultiplesO1(int n) that does the same as sumOfMultiplesOn but has constant asymptotic complexity. Use the algorithm for summing up the multiples of a given integer presented in the lecture.

```
public static int sumOfMultiplesO1(int n) {
   int sum3 = sum(n, 3);
   int sum5 = sum(n, 5);
   int sum15 = sum(n, 15);

   return sum3 + sum5 - sum15;
}

private static int sum(int n, int x) {
   int k = n / x;
   return x * k * (k + 1) / 2;
}
```