Estimating and Timing Algorithms

Purpose:

to extend skills with estimating and verifying $\Theta(\square)$ behaviour.

Due Date:

This assignment is due Wednesday, 2021-10-06 *at the beginning of lecture*.

Stopwatches

Use the `StopWatch` class that you created for Lab Assignment 1.

Re-read the "Instructions on Plotting and Timing" on Casperson's web-site.

Question 2.7 from *Weiss*

Here are the directions from Problem 2.7 in *Weiss*.

> For each of the following programming fragments:
>
> (a) Give an analysis of the running time (big Θ will do).
>
> (b) Implement the code in Java and give the running times for several values of n.
>
> (c) Compare your analysis with the running times.

The program fragments appear on the following page. For your convenience, the algorithms are also inside methods in the `Sums.java` file attached to this assignment. Cut and paste code as you wish.

For these problems, it should be clear that the best-, average-, and worst- case times are the same up to experimental noise.

```
    for(int i=0; i<n; i++)
        sum++;
```

Figure 1: 2.7(1)

```
    long sum = 0;
    for(int i=0; i<n; i++)
        for(int j=0;j<n;j++)
            sum++;
```

Figure 2: 2.7(2)

```
    long sum = 0;
    for(int i=0; i<n; i++)
        for(int j=0;j<n*n;j++)
            sum++;
```

Figure 3: 2.7(3)

```
    long sum = 0;
    for(int i=0; i<n; i++)
        for(int j=0;j<i;j++)
            sum++;
```

Figure 4: 2.7(4)

```
    long sum = 0;
    for(int i=0; i<n; i++)
        for(int j=0;j<i*i;j++)
            for(int k=0;k<j;k++)
```

Figure 5: 2.7(5)

```
    long sum = 0;
    for(int i=0; i<n; i++)
        for(int j=0;j<i*i;j++)
            if (j % i == 0)
                for(int k=0;k<j;k++)
                    sum++;
```

Figure 6: 2.7(6)

For many of the problems the running time is likely polynomial in n. If $T(n) = cn^k$ for some constant c and some integer $k > 0$. In this case plotting $\log T(n)$ versus $\log n$ should give you a straight line with slope k and y-intercept $\log c$.

For each problem, choose your maximal $n_{\max}$-value so as to get an easily measurable time, preferably a nice round number, then choose equal increments from $n = 0$ up to $n = n_{\max}$ at which to run measurements.

For part (a), do your best to guess what the $\Theta(\square)$ behaviour will be. Some of thse are quite tricky.

⇒ For part (b) for each problem, hand in your JAVA-code, and a graph of the running times as in Lab 1. If you wish, you may write one program to measure all of your running times. However, do *not* write one long ugly `main` (Try to keep individual methods to under 25 lines of code.)

⇒ For each problem combine parts (a) and (c). If data from part (b) confirms your guess from part (a), say how it does so. If the data appear to show a different $\Theta(\square)$ behaviour, explain what you think is going on.