Assignment 6: Code Performance (40 Punkte)

Deadline: 25.06.2018

In Assignment 6 you should use the Java Microbenchmark Harness (JMH) to evaluate the performance of different **sorting algorithm implementations** and do your **own** implementations which are (hopefully) **faster** than the provided default implementation.

You are provided an eclipse maven project called PRSW2_UE6_Performance that is based on the JMH maven archetype for benchmarking.

The project contains one Java file called **`SortingBenchmark.java`** that implements a minimal harness for benchmarking the performance of sorting integer arrays. The harness builds a state object containing a two dimensional integer array where the second dimension is of a fixed size. The arrays that should be sorted are in the first dimension. So for a given size e.g. int[10][5] the algorithm needs to sort 10 integer arrays each of size 5. An initial implementation of the Bubblesort is outlined.For your implementation two skeleton methods (marked with **`TODO`**) are given.

Your tasks:

- You should implement a **different sorting algorithm** with the Java native interface (**JNI**). The skeleton that should call your JNI function is given in SortingBenchmark.java . You must **not** use library functions but do a simple sorting function on your own (e.g. QuickSort or MergeSort). Follow the process outlined in the slides to the JNI topic:
  - Generate the JNI header file for the class **JNISorter.java**
  - Implement it in C/C++
  - Compile your implementation to a shared lib
  - Load the lib in the static initializer of **JNISorter.java**
  - Call the method from the benchmark harness skeleton
- Then you should **evaluate** your implementation against the provided default implementation. Hopefully, it is faster! Generate a **PDF report** of your experiments covering the results of the JMH run. Please check in that PDF report in the eclipse project as well under a folder called ***report.***
- As a second task you should **try** to **beat the reference implementation** of Java, being Arrays.sort . You must implement a **second sorting** algorithm, however, this time you are allowed to use any JVM feature you are interested in. But, you are not allowed to use library functions yourself. However, you may use
  - Multiple threads to separate the workload into chunks that can be processed in parallel
  - Do further JNI investigation trying to optimize your sorting with C/C++ tricks
  - Combine threading with JNI
  - Do an optimized Java implementation (e.g. Radix Sort ?)

Hints:

- JNI: You can try to access the Java underlying integer array with *Get<>Critical,* however be aware of the implications that the returned array might not be the original Java array.
- JMH uses annotation processing to generate its benchmark harness classes, you need to run the maven archetype after updating your Java code (maven clean & maven install).
- If you modify Java constants you need to clean the maven archetype and install it again.
- If you fail to set your library path correctly for loading JNI shared library (e.g. on windows) you can use absolute paths.

Workflow (tested with Windows and Linux) :

1. Download sample project PR_SW2_UE8_Performance.zip
2. Uncompress
3. Import in IDE (e.g. eclipse)
4. Run maven clean
5. Run maven install
6. Run benchmarks with **`java -jar target/benchmarks.jar`** from within Eclipse project root folder (cmd line)