Results Anomalies

# JavaScript

Huge difference was spotted only for the **BSTinsert** benchmark test. The problem was in implementation of algorithm, and after rewriting the program using a recursive implementation of Binary Search Tree instead of loops the result became close to the other languages.

On the other side, **BSTremove** was working correctly, and after using the same recursive implementation instead of cyclic the result increased from in average **4 ms** to **10 ms** (as expected, less effective).

If we use the reversed version of array after insertion to remove nodes, the execution time drops from in average **10 ms** to **8 ms**.

# Swift

While being a compiled language and running natively, Swift shows bad results in almost all algorithms, performing even worse than JVM-running languages (Kotlin, Java) while expected to be close to C/C++. The possible problem may be in compiler since Swift is not really tuned to run outside Apple-made operating systems, it lacks most of the libraries available for macOS/iOS. There is a possibility that the Ubuntu/CentOS (the only available options for compiling Swift on Linux) compilers are not optimized which causes such bad results. If that is so, it’s impossible to compare Swift to other languages on Linux platform (which is used in our benchmarks).

Additionally, the mentioned Linux compiler does not offer an option to compile only to produce an executable. Swift program can be only launched with *swift* command which compiles and executes it right away. This leads to additional memory usage during the compilation process and therefore the memory result is not representative.