**Euler 116.**

The solution of this problem is done recursively though the optimal solutions would be done iterative. The work being done in every recursive call is very simple opposed to quicksort. When the work in every recursive call is simple and similar, some sort of compiler optimization will most likely happen if supported by the language. This is a good way to test Julia’s tail recursion optimization against Java's, and comparing with Python and C++. The result of the benchmarking is as the following:

**Results:**

It is clear that the graphs are exponential increasing. Another thing to notice is that the input is only increasing by one but is still making a huge difference in the run time. One of the reasons is that the problem is solved with recursion, for every extra one bit of space added to the black box a lot more recursive calls will have to be made.

The difference in time between the four languages are as expected. Python does not support any form of tail recursion optimization and the result is really slow. Java and Julia on the other hand does a good job at optimizing the recursive calls and is actually faster than C++ - keep in mind that no compiler flags were used in C++, so the default optimization level is used.

Julia and Java is close but Java is a bit faster, which is expected because of the fact that Java has been in development much longer than Julia and has a lot more optimization than Julia.