Compile and run the “branching_student.cu” code. (“nvcc -o branching_student branching_student.cu”)

This code will take a large loop of operations and run it inside of a CUDA kernel. Each thread will be assigned a task, and depending on the thread ID, either all threads will be assigned the same task, or some threads will be assigned different tasks. This will be done in sets of threads of size nBreak, and work will be assigned vie a modulus statement, so that threads 0 to nBreak-1 will get one set of instructions, nBreak to 2 nBreak-1 will get another set of instructions, and so on.

The code is designed to run for nBreak ranging from 8 to 32, with most GPUs having a warp size of 32.

1. Run the code “./branching_student 32” with nBreak=32. Run it repeatedly for decreasing nBreak, and make a plot of runtime as a function of the work set size in the GPU kernel.
2. Do you see evidence for branching having an impact on efficiency of a GPU kernel? If so, how does the warp size affect this? Explain in detail, referring back to the plot you made in example 1.