In order to make the concurrent link-based stack a lock-free one, we first insert the stack with a fixed number of nodes to avoid errors when randomly deciding an operation such as push or pop. We then pre-allocate a fixed number of nodes to be used later and a fixed amount of random numbers that will determine the whether a thread should push, pop, or return the size of the stack. Then we create the four threads, and each are timed. Each thread will be given access to the nodes created, set of random numbers, and the stack reference. We run each thread and they will each perform their operations based on the random number drawn from the reference. Because our goal is to provide a lock-free algorithm, one viable approach would be to convert the necessary variables to atomic (i.e. the variables that keep track of the head pointer, the size, and the number of operations of the stack). Doing so, we guarantee an environment where the stack can be modified in a thread-safe manner and that it can generally be more efficient in performance than its lock-based counterparts. If we didn’t make use of atomic or volatile variables in a lock-free setting, then our code is prone to race conditions, duplicate operations, and other thread-unsafe consequences. Finally, we compare results by varying the ratio between the operations and plot them on a graph (check excel file).