Austin Brummett
Project 7 CS375
Due: November 20, 2018

Algorithm:

My algorithm focuses on a few things:

- Setting a maximum number of threads if N is greater than 10
  - 10 is max. number of threads
    - performance dropped off after 10
  - Otherwise number of threads == N
- A counter variable called step is set globally
  - in my thread function
    - thread counter c = step++
      - c tracks which thread is doing the process
      - the statement i=c*N/numThreads
        - tells which portions of the array each thread should take care of
        - sums everything into the sum variable and then puts it in the correct position in the array

Results:

Single Threaded

Trial \ N value	10	100	1000
1	.004	.013	5.518
2	.004	.013	5.871
3	.004	.013	5.533
4	.004	.013	5.558
5	.004	.013	5.823
Avg Time(s)	.004	.013	5.6606

Multi-Threaded

Trial \ N value	10	100	1000
1	.005	.007	1.196
2	.005	.007	1.212
3	.005	.006	1.223
4	.005	.007	1.208
5	.005	.008	1.221
Avg Time(s)	.005	.007	1.212

Results Discussion:

Based on my results and testing, single threaded processes were more efficient when N is less than 50, but as N increased above that the multi-threaded started to outperform the single threaded multiplication.

For N = 10:

Single threading was 1.25x faster

For N = 100:

Multi-Threaded is 1.86x faster

For N = 1000:

Multi-Threaded is 4.67x faster