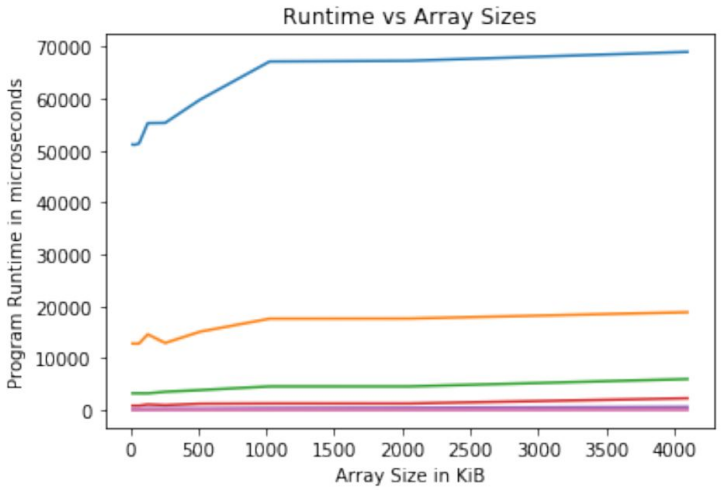
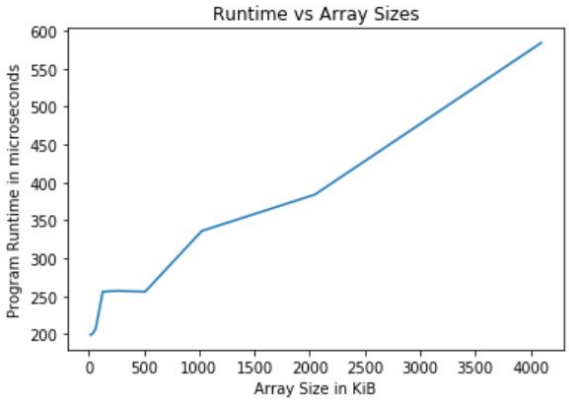
CODE: See **project1_1.c** for more details.
PLOTS: The plots were created using Python. See **project1_1_plots.pdf** for more details. The data was printed to text files from running the C code, then read into the Python code.

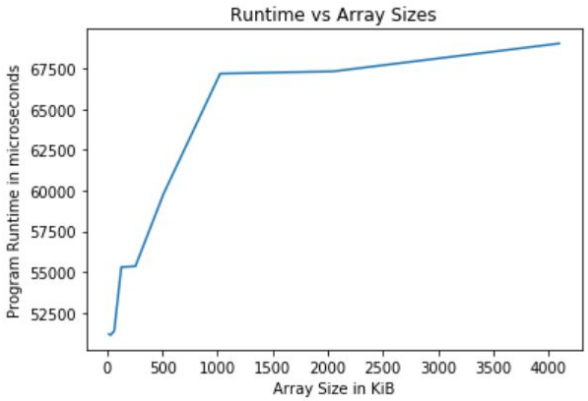
Part 1: Cache Size

To avoid segmentation fault, the maximum array size tested was 4 MiB, which has 1024^2 array elements (each integer is size 4 bytes). In order to test cache size, we want to reuse data by accessing the same array elements over and over again. The first plot shown has multiple lines; each line uses a different "stride" length, which indicates the number of array elements accessed again and again. Smaller stride indicates more array elements, which resulted in greater runtime. This was done so we can view trends across different strides for more data collection. The instructions that were timed are merely array access instructions and don't involve any computations.



After getting a sense of general trends, we focused on individual datasets to determine cache size. On the left, the first peak is around 64 KiB, which then plateaus until it reaches 512 KiB. Based on this information, we can guess our **L1 cache size to be 512 KiB** because there is no significant change in runtime between 64 and 512 KiB, indicating that data is being stored for faster access in the L1 cache. Other drops occur around 1024 KiB and 2048 KiB. Combined with the plot on the right side, where we can see another clear plateau around 1024 KiB, **we can conclude that an L2 cache does exist.**





Our answer of L1 cache = 512 KiB matches the L1 cache size given to us through the lscpu command. Although we did not find the size of L2 cache, because our array size appeared to be limited at 4 MiB, this matches the below information because otherwise we would have needed to use **malloc** for additional space.

```
slc265@engsoft:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   48 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              1
Core(s) per socket:              4
Socket(s):                       2
NUMA node(s):                    2
Vendor ID:                       AuthenticAMD
CPU family:                      16
Model:                           4
Model name:                      Quad-Core AMD Opteron(tm) Processor 2376
Stepping:                        2
CPU MHz:                         800.000
CPU max MHz:                     2300.0000
CPU min MHz:                     800.0000
BogoMIPS:                        4599.88
Virtualization:                  AMD-V
L1d cache:                       512 KiB
L1i cache:                       512 KiB
L2 cache:                        4 MiB
L3 cache:                        12 MiB
NUMA node0 CPU(s):               0-3
NUMA node1 CPU(s):               4-7
```

Part 2: Memory Bandwidth

To test memory bandwidth, we access a large array of memory. We know to make this array large enough to exceed cache size, which we obtained from the first part of this question. The initial plateau until 512 KiB here also indicates that our L1 cache is 512 KiB. If we divide the size in memory by run time, we can calculate the bandwidth. At its maximum up to 4 MiB, we have a bandwidth of about **6.94 GiB/s**.