PS3 Questions
==========
Written By: EJ Rainville, Spring 2021

Add your answers to this file in plain text after each question.
Leave a blank line between the text of the question and the text of
your answer.

argv
----

1. What does `argv[0]` always contain?

The variable 'argv[0]' always contains the call to the executable file. Therefore when you type the command to run an executable such as './repeat.exe' then argv[0] will be equal to './repeat.exe'.

2. Which entry of `argv` holds the first argument passed to the program?

The first argument that is passed to the program is contained in 'argv[1]' which is the second index of the array of pointers of argv.

3. Which entry of `argv` holds the second argument passed to the program?

The second argument that is passed to the program is contained in 'argv[2]' which is the third index in the array of pointers of argv.

4. How would you print just the last argument passed to a program?

You could print just the last argument passed to a program by subracting one index from the number of arguments passed in that is saved in 'argc'. Therefore, to print the last argument you could use a command such as 'std::cout << "Last argument is : " << argv[argc-1] << std::endl;'.

float vs double
----

5. What is the difference (ratio) in execution times between single and double precision for   *construction with no optimization*? Explain.

When comparing the execution time between floats and double precision

without optimization for construction, for a problem size of 10,000,000 the ratio between single and double precision execution time is 0.87. When using a larger problem of size 20,000,000, the ratio between single and double precision executing time is 0.91. Here we see that the time to compute with floats is faster than with doubles which is expected since the floats contain less information and therefore more can be stored in a cache prior to execution thus reducing time in retreiving the information from memory.

6. What is the difference (ratio) in execution times between single and double precision for *multiplication with no optimization*? Explain.

When comparing the execution time between floats and double precision without optimization for multiplication, for a problem size of 20,000,000 the ratio between single and double precision execution time is 0.96. When using a larger problem of size 50,000,000, the ratio between single and double precision executing time is 0.7. Again, we see that the computation with floats is faster than doubles since they take less space and therefore more values can be pipelined into the CPU, we also see for the larger problem that the difference in computation time is more significant that the smaller problem since the number of operations required is N^3 and therefore savings in a small problem are less significant than in a large problem.

7. What is the difference (ratio) in execution times between single and double precision for *construction with optimization*? Explain.

When comparing the execution time between floats and double precision with optimization for construction, for a problem size of 20,000,000 the ratio between single and double precision execution time is 0.46. When using a larger problem of size 50,000,000, the ratio between single and double precision executing time is 0.72. For both problem sizes, we see again that the float is faster than the double computation since more of these values can be loaded into the CPU cache and therefore less time is spent retreiving data from memory for computation.

8. What is the difference (ratio) in execution times between single and double precision for *multiplication with optimization*? Explain.

When comparing the execution time between floats and double precision with optimization for multiplication, for a problem size of 20,000,000 the ratio between single and double precision execution time is 0.45. When using a larger problem of size 50,000,000, the ratio between

single and double precision executing time is 0.69. As we saw before the float computations are faster than the double computations for the same size problem and it is directly related to the amount of memory that can be stored in short term memory caches that can be quickly accessed.

9. What is the difference (ratio) in execution times for double precision multiplication with and without optimization? Explain.

The difference between the optimized and not optimized computation times for a matrix multiplication were significant. For a problem size of 20,000,000 the ratio of optimized to not optimized matrix multiplication is 0.49 meaning the optimization cuts the computation time in half. When looking at a larger problem size of 50,000,000 we see an even more significant difference between the computational times with a ratio between optimized and not optimized of 0.23 which means the optimized case is even more significantly faster for the larger problems. This optimization is the fastest running time; however, it takes the longest to compile (from c++ manual entry) so there are tradeoffs with this flag.

All-Pairs
---

15. What do you observe about the different approaches to doing the similarity computation? Which algorithm (optimizations) are most effective? Does it pay to make a transpose of A vs a copy of A vs just passing in A itself. What about passing in A twice vs passing it in once (mult_trans_3 vs mult_trans_4)?

One of the first features that we notice about these algorithms is that for all problems with more than 128 images, the mult_0(A,B), mult_1(A,B), m_t_0(A,C) and m_t_0(A,A) were all too slow and gave outputs of -1 showing that these were all very inefficient algorithms as expected. The most effective algorithms were mult_3(A,B), m_t_3(A,C) and m_t_3(A,A) since they each had the largest GFlop/s values in their respective categories. It is fastest to just pass in A itself directly since making copies takes extra memory that can then take time to store and retrieve thus reducing the efficiency of the algorithm and we can see this in the measurements of GFlop/s of m_t_4(A) is the highest value consistently across each algorithm.

16. What is the best performance over all the algorithms that you observed for the case of 1024 images? What would the execution time

be for 10 times as many images? For 60 times as many images? (Hint: the answer is not cubic but still potentially a problem.) What if we wanted to do, say 56 by 56 images instead of 28 by 28?

The best performing algorithm for the case with 1024 images was m_t_3(A,A) that utilized hoisting, tiling and blocking to increase the number of operations that could be done consecutively with a value of 16.6077 GFlop/s. From the previous section we know that a matrix multiplication should be effectively N^3 operations; however, the execution time should be operations/(computation rate) which would mean for this problem the execution time should be approximately 64.6 milliseconds and assuming this computation rate stays similar the execution time should be approximately 64653 milliseconds which is about 1000 times more than the current problem. If we increase to 60 times more images with the same computation rate we expect an execution time of 13965102 milliseconds which is about 216178 times more than this problem. If we increase the image size from 28x28 to 56x56 and we used 1024 images still we would now expect an execution time of 1857 millisecond which is an increase by a factor 28.7 from the execution time for 28x28 pixel images which is very significant.

About PS3
---

17. The most important thing I learned from this assignment was ...
The most important thing that I learned in this assignment was how much using a float value and a double value can have a huge impact on the computational time for a problem so knowing the type of precision needed can be important to large computations.

18. One thing I am still not clear on is ...
I am still not very clear on exactly what the -03 optimization flag is doing to optimize the programs execution.