Question 1

Part 4

To compare and analyze the performance and efficiency of the three functions Serial, GPU Per Particle, and GPU Per Pair, Table 1 shows the execution times of the functions with N values ranging from 8K to 20K. The serial CPU baseline algorithm has approximately an $O(N^2)$ complexity. In contrast, part 2 algorithm (GPU Per Particle) employs and utilizes the GPU to achieve speedup. The total number of calculations remains approximately the same, in GPU Per Particle method the computations on the GPU are distributed across its cores. So, each GPU thread handles approximately O(N) work to compare its particle with others concurrently (excluding itself), and then followed by a parallel reduction step, which results in faster execution.

Function	**#N**	**Time (seconds)**
Serial_CPU	8000	0.249650
	9000	0.311938
	15000	0.865740
	20000	1.538474
GPU_Per_Particle	8000	0.001668
	9000	0.001882
	15000	0.003493
	20000	0.004625
GPU_Per_Pair	8000	0.355984
	9000	0.471139
	15000	1.669346
	20000	3.697619

Table 1: Comparison of time using different functions with varying N.

The GPU_Per_Pair method is twice as slow as the serial CPU baseline, and the GPU_Per_Particle method is faster. This is logical because GPU_Per_Pair needs a substantial memory allocation and suffers from a memory bandwidth bottleneck. Writing global data is intensive, and also frequent global memory access are much slower than GPU computations

which likely creates a performance bottleneck. Additional time is also lost because of the reduction and management overheads.

These results shows a clear trade-off between the methods Part two (GPU_Per_Particle) balances parallel computation with manageable data size, while part three (GPU_Per_Pair) suffers from memory usage and higher communication overhead.

Part 5

The fundamental computation of the three algorithms scales to O(N$\hat{2}$). But by utilizing GPU implementations as in part 2 and part 3, they can reduce the runtime.

If we think about algorithm redesigning we can use space partitioning methods like kd-trees or divide and conquer, this will avoid comparing all pairs because binning the particles spatially, limits the calculations of distance to nearby particles. For the reduction part faster atomic operations can be achieved by using scaled integers instead of distances or I can also try hierarchical reduction. For the GPU considerations, one thread per particle approach seems the most effective because the number of threads will be limited to N, however, each thread does O(N). On the other hand, one thread per pair maximize the effectiveness of parallel computing but the additional memory, management overhead and kernel launching is faced when N is large.