Nicky Ren
Lance Ding
David Choi

1.1

1.1.1

Single Source Problem

	tinyEWD.txt	rome.txt	10000EWD.txt
Dijkstra	51.0 ms	191.0 ms	597.0 ms
BellmanFord	50.0 ms	195.0 ms	604.0 ms

	tinyEWD.txt	ROME.txt	10000EWD.txt
Dijkstra	51	191	597
Bellman Ford	50	195	604
Relative difference (% Dijkstra)	-1.960784314	2.094240838	1.172529313

1.1.2

Pairs Problem

	tinyEWD.txt	rome.txt	10000EWD.txt
Dijkstra	55.0 ms	43444.0 ms	449406.0 ms
BellmanFord	59.0 ms	44539.0 ms	386510.0 ms

	tinyEWD.txt	ROME.txt	10000EWD.txt
Dijkstra	55	43444	449406
Bellman Ford	59	44539	386510
Relative difference (% Dijkstra)	7.272727273	2.520486143	-13.99536277

1.2

Analysis of Runtimes

Upon analyzing the runtimes of the algorithms, our team discovered that with the exception of the Pairs Problem on the largest dataset (10000EWD.txt), the two algorithms performed comparably through all tests, varying by amounts small enough that they may be explainable through background variation in the system running the code. Looking at our data tables, we can see that only 2 of the tests captured a significant ($|\Delta| \geq 0.05$) difference, and on one of them the large relative difference is probably caused by the nature of constant variation scaling to larger effects on data with smaller magnitudes (e.g. 1ms matters a lot more to a situation where the entire runtime is 10ms compared to one where the runtime is 1000ms). Therefore we conjecture that most of the variation of the runtime is caused by random variation and not by any systematic difference between the performance in the algorithms with the exception of the 10000EWD dataset.

Focusing on the discrepancy in runtimes on the 10000EWD dataset, we discussed and came up with several possible contributing factors that caused the Bellman Ford algorithm to run significantly faster than Dijkstra's algorithm. We speculate that the specific queue (regular FIFO queue) based implementation of the Bellman Ford algorithm may have contributed to the faster runtime of said algorithm in relation to Dijkstra's algorithm in larger datasets. The queue data structure allows the vertices to be processed in a specific, predictable order, which may allow java to more efficiently manage and allocate memory. Additionally, our use of a priority queue in the implementation of Dijkstra's algorithm may have also contributed to the slowness of the algorithm on larger datasets. Since the priority queue used to decide the next vertex to travel to requires maintenance similar to that of a heap, more complicated memory allocation and processing processes may be required, and java may in turn take more time to execute the code.

On the other hand, if we rule out random variation as a cause of the runtime discrepancy between Bellman Ford and Dijkstra's algorithms (since Dijkstra's algorithm does seem to work faster over each test by a very small margin), Dijkstra's working more efficiently can be due to the algorithm's fixed time complexity. The Bellman Ford algorithm exhibiting an average runtime of O(E + V) and a worst case runtime of O(E * V), while Dijkstra's has a more constant runtime of O(E log V). This discrepancy arises from the inherent nature of the algorithms as we talked about earlier. Additionally, Bellman-Ford's worst-case scenario involves negative edge cycles, but with Lance's summary table that he wrote to generate a report for the distinctive datasets before

running the algorithm confirms the absence of negative edges in the datasets first. In this case, ensuring that Bellman Ford never reaches its worst-case time complexity may result in a more efficient algorithm.

However, it's important to note the limitations of our conclusions. Even though our data presented a case for Bellman Ford being faster, there are a few caveats to consider when comparing runtimes. Since we are taking a very small sample of a few runs to compare the runtimes, we cannot conclude anything conclusive about the runtimes of the algorithms. It is likely that we can expect similar results but without extensive testing that would likely take up a lot of time and memory, we cannot for certain conclude any results. Another consideration to take into account is that since we are timing the algorithms using the computer and not analytically, there are many different variables from the computer itself that have to be accounted for. There can be various background processes happening at different times that could account for a difference in runtimes leading to some inaccuracies. We again want to emphasize that the differences between our runs of the Dijkstra and Bellman Ford algorithm could be attributed as variance in computer processing. In addition, while testing the algorithms we noticed that there was a large variance between the runtimes on different machines. The algorithms ran significantly faster on Lance’s desktop compared to David’s laptop in which the two computers ran on a different architecture. In our efforts to reduce any biases, we decided to try to remove as many background programs and processes as possible and to keep the set of all dataset times separate from each computer and run instance (to group the 12 runtime outputs together for each set). With all these considerations we believe there is a statistical correlation between the algorithms and the runtimes but we cannot concretely conclude without further exploration.