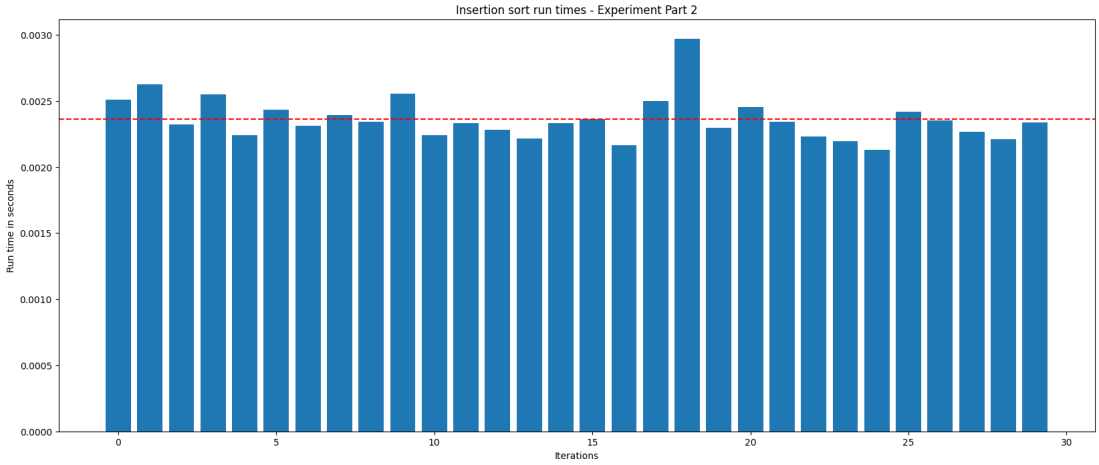
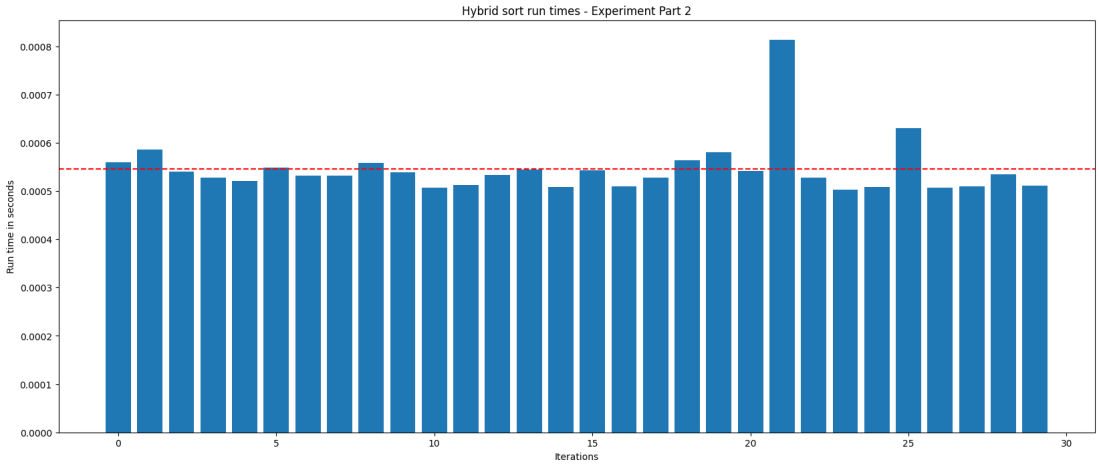
Part 2

a.





b. Traditional insertion sort run time's mean is 0.00236, while hybrid sort run time's mean is 0.000546. From both the graphs and the numbers, hybrid sort is clearly much faster.

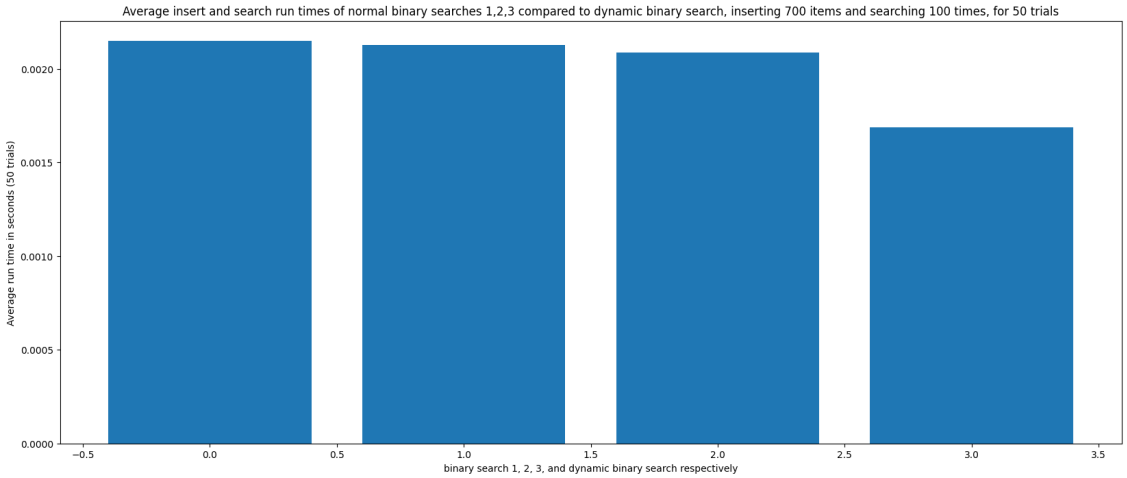
c. Hybrid sort's complexity is NlgN because the binary search to find the appropriate index of an item is always logarithmic; the correct place needs to be found for all items in a list of length N.

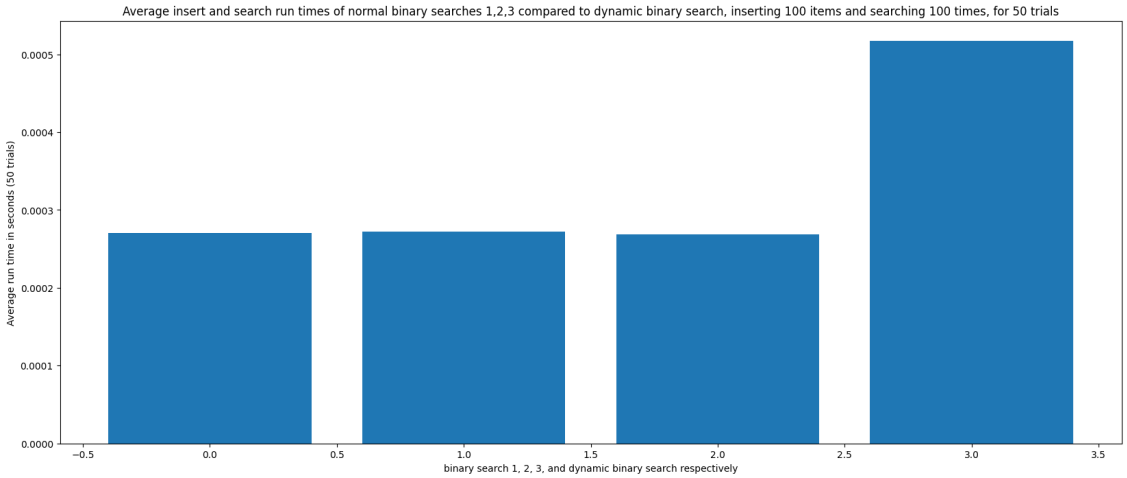
d. The experiment design is in the code, but the evidence of the comparison is logged into the table below, where hybrid sort is generally faster than insertion sort except in insertion sort's best case where insertion sort is essentially linear, but where hybrid sort lost the ability to take advantage of because it performs binary search on every element.

	Average	Worst	Best
Insertion	0.00407	0.00487	2.38e-05
Hybrid	0.000872	0.000675	0.000260

Part 4

a. The dynamic version of binary search uses a more complicated data structure to optimize inserts and deletes into a searchable dataset. My experiment focused on inserts and searches, and I found after varying the number of inserts while keeping the number of searches the same, and after varying the number of searches while keeping the number of inserts the same (not shown here), that when there are many searches but not so many inserts, the normal binary searches with the linear time insert function I wrote for them perform better, whereas if there are many inserts compared to the number of searches, the dynamic binary search is faster.





b. In other words, when the task that this more complicated data structure is optimized for is not performed often enough to justify the increased cost to other aspects, dynamic binary search is “overkill” and performs worse than traditional solutions.

c. Here, I present two graphs that most starkly show the difference in performance when the number of inserts is too low or high enough to make the dynamic solution outperform traditional ones. More are not shown in the interest of keeping this brief report brief. In the first, dynamic binary search displays a shorter average insert-and-search run time than all three traditional

binary searches, which all performed roughly the same, and in the second, the opposite is shown with the three traditional binary searches still roughly the same.

Part 5

a. Unknown iterates through every combination to find the shortest path lengths between all nodes. After googling all pairs shortest path algorithms, seeing a cubic one, and comparing with the pseudo-code, I discovered it's the Floyd-Warshall algorithm,.

b. Unknown has 3 for-loops, making it very clearly cubic.

c. It's not surprising, given that Dijkstra (shortest path from each node) implemented with list is quadratic, so times another V number of vertices (shortest path from every node) makes it cubic.

Part 6

Both algorithms take more time as the graphs get denser which makes sense as both of their complexity are partly dependent on the number of edges in the graph, but Prim's is consistently faster than Krushkal's, and on average. Prim's average is 7.74e-05, and Krushkal's is 9.91e-05.