Good afternoon TA, my name is Yi Hao and today, together with my groupmates Sahithya, Saori and Joel, we will be presenting to you on our findings for Example Class 2.

This is the table of contents for today’s presentation.

Let us first begin by talking about what is Dijkstras Algorithm. Dijkstra’s is a path finding algorithm that tries to find the shortest path in a graph with non-negative edge weights. It can be implemented in 2 ways, via an Adjacency Matrix (with an Array Priority Queue) or an Adjacency List (with a minimizing Heap Priority Queue), both of which we will talk about later. To understand what exactly the algorithm does, let us look at a visual proof of it. At each iteration, Dijkstra’ Algorithm chooses the vertex with the next smallest distance and then relaxes the edge weights going out from it. This occurs until all vertex have been chosen once. The resulting distance represents the shortest distance.

For our testing, our group decided to standardize different input graphs. We concluded the best case would be a graph with only a single path, average being a graph with randomly chosen edges and worst to be a complete graph. This could be generalized to thinking in terms of edges where the best case has the minimum number of edges and the worst case having the maximum number of edges with the random case lying between both.

With that in mind, I will now talk about the Adjacency Matrix implementation. Let us first take a look at the pseudocode. Although there were different ways to implement an array priority queue, our group decided to utilize the arrays that contained the current shortest distance to a vertex and whether it had been visited before. At each iteration, minDistance extracts the next unvisited vertex with the shortest distance and then accordingly relax the edges adjacent to it. This occurs until all vertices have been visited.

Looking at the theoretical time complexity, the minDistance function occurs at thetha (V) as it iterates over each element once. Further more during the looping of each adjacent vertices, we will visit at most 1 to V-1 vertices. Since both of these occuration occur while there are vertices with unfinalized distance, we can generalize this to occur V times and hence the time complexity is big thetha (V^2).

After running our algorithm on all 3 different input graphs at 10 vertices, these were the results. We can clearly see that our expected worst case input took the longest while our best case input took the fastest. These results were further reinforced by the fact that the trend was maintained as the array size increased to 100, 1000 and 10,000. Keeping what we’ve discussed in mind, I will now pass the time to Sahithya to talk about the adjacency list implementation.