We made a few changes to the project since the initial proposal. Instead of using Primm's or Kruskal's algorithms for the traversal of the graph, we used Djisktra's algorithm, and Breadth First Search, to find the shortest paths from one song to the next.

The Breadth First Search algorithm will run in O(V+E) time in the worst case. When the algorithm runs, it has the possibility to go through each vertice in the graph if it does not find the target song before the end. As it traverses the graph, it is placing the vertices into two separate maps, one to keep track of what was visited, and another to keep track of each vertex's previous vertex in the search, which are both O(1) operations. The map uses the song ID's as keys, and stores the previous node as the value Once the search is done, the history of the target song in the map is iterated through to get all the names of the songs that came before it in the search up until the source song, and then prints this in reverse to show the connection from source to target.

The Djistktra's algorithm search takes O(ElogV) time to complete in the worst case, and is often shown to be slower than the BFS algorithm.The Djikstra's algorithm has to iterate through the entire graph every time first before it actually creates the connection/path of vertices that are displayed.

All of the spotify api functions take up most of the time of the program, as they are used to create the graph. If E corresponds to the number of artists, and V corresponds to the songs, we search through E for every V it is connected to (finding all

songs made by an artist), and then search through all E that are connected to every V (all artists that were featured in the songs made by the artist).

The actual insertion of these elements into the graph is O(1), as we are inserting a pair (song, artist) into a list that is the value of a part of a map, where the keys are the song IDs.