**Algorithms & Data Structures II Design Document**

**Shortest Path:**

For the shortest path I decided to use the Dijkstra algorithm to find the path from the source to the destination. Since we are only interested in the cheapest path from a single source, it is a suitable algorithm compared to Floyd-Warshall which looks for every path from every source which is unnecessary calculation for us. The time it would take to find the path is at a constant *E* log(*V*) for a typical case and the worst case as well and its space complexity takes up *V* space, so the wait time for the user will be similar regards the path the user will end up with. Bellman-Ford was another choice I thought of, its space complexity is the same as Dijkstra but the typical and worst case time is *E V*. When implementing Dijkstra, I thought it would be cleaner to implement it by creating the Edge and Vertex as their own classes compared to my previous first time implementation of Dijkstra. I used an ArrayList to save and sort different values and inputs for this algorithm since it is easy to use in java and has good features that are useful for different operations. Its sort function uses either the Mergesort or the Timsort, both of which have the worst-case time complexity of nlog(n) which is pretty good considering the input files we are dealing with are very large. The ShortestPath class has 2 methods besides the Dijkstra calculation which is required for the project, one of them calculates the distance, which will be the cost, and the other one records and then prints the path it takes. That way it’s easier to calculate separate results from the ShortestPath class. For stop ID’s I decided to try out using one of the java features which is a HashMap function to store and retrieve bus stop ID’s since it has a constant time of O(1) to store the stop ID’s and worst case O(n) to get the stop ID’s.

**Bus stop name searching:**

For the bus stop name searching I implemented the TST class as required from the project description. Credit to Princeton for main template of the TST class that already provides most of the main features. Since we are dealing with strings, I made it the way so that will work with strings. The insertion and search hit have a typical cost of *L* + ln *N* which isn’t the best compared to others like BST or R-way Trie, but it does have a space complexity of 4*N*, making it more space efficient compared to some other string search algorithms. I implemented a StringBuilder to manipulate the keyword flagstops in the names to make name searching by the user to be more meaningful and easier to find when it comes to a specific name. Then I create another StringBuilder to construct the print and display of the information to be more cleaner for the user and generally easier to read. StringBuilder has an insert time complexity of O(1), so considering how large amount of information that can be printed, this is very beneficial for the program to be more time efficient.

**Printing the arrival times:**

For printing the arrival times, I did a bit of research on the best way that I can quickly read the lines from stop\_times.txt file, looking for the ones that match the format of the time. So I made use of the java.nio.file library and made use of the functions to read the text file and obtaining the valid bus stop times and storing them in the ArrayList. Then I have the main function where I call it in the user interface to print out the messages and ask the user to give the time in a certain format. Once the time is given by the user, I check if the time is valid and if it matches in any of the times that were read from the given text file. Any times that match I store all the information with the matching time in the result ArrayList. Once it searched all of the times that matched with the times that were found from a text file, the other function then takes in the ArrayList result and prints out all the information in the sorted order that match the arrival times. The sorting is done using the method Collections.sort(). The sorting algorithm in this case also uses either Timsort or Mergesort with space on O(n) and worst case time complexity of nlog(n). There is also a minor function that parses a String into an int.

**User Interface:**

As for the interface, I made it simple enough for the user to be able to navigate through a choice of different functions the user wants to use. Most of them is typing in a number from 1 to 3 to choose a function they want to use, or they can type 0 or ‘quit’ to exit the program.