**Title Page (KG)**

## **1 Introduction**

## **Application Requirements (DC, SD, SL)**

The objective of this project is to develop an interface using techniques learnt in class which will act as a bus information system for Vancouver public transport. For the purpose of this project, we have used three text files including stops.txt, transfers.txt and stop\_times.txt. Using these input files, we have implemented the following functionalities: Finding shortest paths between 2 bus stops (as input by the user), returning the list of stops en route as well as the associated “cost”. Searching for a bus stop by full name or by the first few characters in the name, using a ternary search tree (TST), returning the full stop information for each stop matching the search criteria (which can be zero, one or more stops), Searching for all trips with a given arrival time, returning full details of all trips matching the criteria (zero, one or more), sorted by trip id and Finally Providing front interface enabling selection between the above features or an option to exit the programme, and enabling required user input.

## **Technical Approach (DC, SD, SL)**

## **2 Design**

## **Logic Flow Diagram (KG)**

## **3 Implementation**

## **Tools, Libraries, Platforms (ALL)**

## **User Interfaces (KG)**

## **Algorithms (DC, SD, SL)**

The following are the design decisions in which we discussed and made, relative to each of the three main features:

Part 1:

Part 2:

Part 3: This part entailed searching for all trips with a given arrival time, returning full details of all trips matching the criteria, sorted by trip ID. The data structure used in this part of the project includes the use of Array List. We choose to use an array list here as we are using an input file of over 1.7 million entries and array lists have a space complexity of O(N) . As well as that Array lists work well with Collections.sort() which I initially thought implemented quick sort but after research I found it implements Tim sort which is a hybrid stable sorting algorithm derived from merge sort and insertion sort. Collections.sort() also sorts elements presented in a specified list of collection in ascending order which is ideal as This part of the project the trips are needed to be sorted by ID. We found it ideal to use Collections.sort() as it is fast and has a guaranteed run time of N log N, it is stable meaning the order of stops is preserved.

## **Runtime Environment (KG)**

## **Conclusion**

We split the group work evenly amongst the team. We met frequently to discuss and review different parts of the project. The following is a description of each of the parts each member played in the group :

Sean Langan : Implemented part 1 & worked on the document .

Denisa Costinas: Implemented part 2 & worked on the document.

Sarah Dolan : Implemented part 3 & worked on the document.

Keira Gatt : Implemented part 4 & worked on the document.