1 Should I give a ride? (10 marks)

As a smart student you are always trying to optimise the driving time to your early morning algorithms lectures so that you can sleep more. You will be leveraging your algorithms skills to get an optimal solution.

Some of the roads in your city have carpool lanes that can only be used if there are at least 2 persons in the car, so you have to decide if you will be giving a ride to a fellow student or not. You have access to precise travel time information among key locations in your city, both with single car occupancy and with 2 or more persons in the car. And you have a list of locations in which there are students looking for a ride to the same destination you are going. You can assume that the potential passengers are always on time at the agreed location and there will be no additional time incurred for getting them into the car. You will either pickup passenger(s) with the same destination as you, or go alone the whole trip. Your absolute priority during those very early morning hours is maximising your sleeping time, so you are looking for the shortest total driving time and will not give a ride if that increases the total driving time.

You are a law-abiding citizen and would never drive in a carpool lane while you are alone in the car!

There are $|L|$ key location points represented by $0, 1, \ldots, |L| - 1$. Your algorithm will take as input the following: a departure location `start` $\in \{0, 1, \ldots, |L| - 1\}$, a destination location `end` $\in \{0, 1, \ldots, |L| - 1\}$, a list `passengers` of locations where there are potential passengers, and a list of roads `roads` with the corresponding travel times. `passengers` is a list of integers such that each integer i in it is such that $i \in \{0, 1, \ldots, |L| - 1\}$ and indicates that there are potential passengers at location i looking for a ride to your destination. The list of roads `roads` is represented as a list of tuples (a, b, c, d) where:

- $a \in \{0, 1, \ldots, |L| - 1\}$ is the starting location of the road.
- $b \in \{0, 1, \ldots, |L| - 1\}$ is the ending location of the road.
- c is a positive integer representing how many minutes you would spend to drive from a to b on that road if you are alone in the car.
- d is a positive integer representing how many minutes you would spend to drive from a to b on that road if you are there are 2 or more persons in the car.

Regarding those inputs:

- For any tuple in (a, b, c, d) in `roads`, you can assume that $d \leq c$ (as you can still use the non-carpool lanes when there are passengers in the car). For some tuple (a, b, c, d), it might be the case that $c = d$ (as some roads might not have carpool lanes, or the carpools lanes might not be improving the actual travel time on that road).
- You can assume that no roads will have only carpool lanes. I.e., if there is a road from a to b, it will always be possible to travel on it even if you are alone in the car. Therefore for any tuple (a, b, c, d) in `roads` the values c and d are well-defined.
- You can assume that for every location $\{0, 1, \ldots, |L| - 1\}$ there is at least one road that begins or finishes there.
- You can assume that there is a route to go from `start` to `end` and that `start` $\neq$ `end`.
- The locations specified by `start` and `end` will not have potential passengers.
- You cannot assume that the list `passengers` is given to you in any specific order, but you can assume that there will be no repeated integers in `passengers`.

- You cannot assume that the list of tuples in `roads` are given to you in any specific order.
- You cannot assume that the roads are 2-way roads.
- The set of locations P specified in `passengers` can constitute any subset of $\{0, 1, \ldots, |L|-1\} \setminus \{\texttt{start}, \texttt{end}\}$, therefore you can neither assume $|P|$ is a constant nor assume that $P = \Theta(|L|)$.
- The number of roads $|R|$ might be significantly less than $|L|^2$, therefore you should not assume that $|R| = \Theta(|L|^2)$.

You should implement a function `optimalRoute(start, end, passengers, roads)` that returns one optimal route to go from `start` to `end` with the minimum possible total travel time. Your function should return the optimal route as a list of integers. If there are multiple route achieving the optimal time, you can return any one of them.

1.1 Complexity

Given an input with $|L|$ key locations and $|R|$ roads, your solution should have time complexity $O(|R| \log |L|)$ and auxiliary space complexity $O(|L| + |R|)$.

Note that the number $|P|$ of locations where there are potential passengers looking for a ride to your destination is not stated in the complexity. If your algorithm has time complexity $\Omega(|P||R| \log |L|)$, you will be losing a very significant amount of marks for this question.

1.2 Example

Consider the example below:

```
# Example
start = 0
end = 4
# The locations where there are potential passengers
passengers = [2, 1]
# The roads represented as a list of tuple
roads = [(0, 3, 5, 3), (3, 4, 35, 15), (3, 2, 2, 2), (4, 0, 15, 10),
(2, 4, 30, 25), (2, 0, 2, 2), (0, 1, 10, 10), (1, 4, 30, 20)]

# Your function should return the optimal route (which takes 27 minutes).
>>> optimalRoute(start, end, passengers, roads)
[0, 3, 2, 0, 3, 4]
```

Warning

For all assignments in this unit, you may **not** use python **dictionaries** or **sets**. This is because the complexity requirements for the assignment are all deterministic worst-case requirements, and dictionaries/sets are based on hash tables, for which it is difficult to determine the deterministic worst-case behaviour.

Please ensure that you carefully check the complexity of each in-built python function and data structure that you use, as many of them make the complexities of your algorithms worse. Common examples which cause students to lose marks are **list slicing**, inserting or deleting elements **in the middle or front of a list** (linear time), using the `in` keyword to **check for membership** of an iterable (linear time), or building a string using **repeated concatenation** of characters. Note that use of these functions/techniques is **not forbidden**, however you should exercise care when using them.

Please be reasonable with your submissions and follow the coding practices you've been taught in prior units (for example, modularising functions, type hinting, appropriate spacing). While not an otherwise stated requirement, extremely inefficient or convoluted code will result in mark deductions.

Abiding by the complexity requirements is extremely important and failure to do so will be penalised heavily. It may well be worth it to submit a half-working solution within the complexity bounds than a completely working solution that is not within the complexity bounds.

These are just a few examples, so be careful. **Remember that you are responsible for the complexity of every line of code you write!**