ASSIGNMENT 4

ABHAY SHANKAR K: CS21BTECH11001

1. Overview

- There are m passenger threads and n car threads, and each thread waits on the availability of a thread of the other kind.
- A counting sempahore is used for the passenger wait, whereas the car threads busy-wait.
- When each thread finishes operation, it appends all the benchmarking data onto a global vector, and once all threads finish, this data is averaged and written to a csv file.
- The enclosed python scripts read from this file and generate graphs. Building the enclosed latex file generates the pdf report.

2. Contents

2.1. Classes.

- Params: Contains all the user-defined parameters, taken from `inp-params.txt`
- Fleet: Represents the collection of cars in the park.
  - `acquire()`: 'Polls' the car threads by iterating through the `status` deque. When a car is `Waiting`, changes it's state to `Service` atomically and returns the index.
  - `release()`: Takes the index of a car thread and sets it's status to `Cooldown`
  - `ready()`: Sets status to `Waiting`.
  - The Constructor and Destructor spawn and join threads respectively. Furthermore, the Constructor opens the `glob` semaphore and the Destructor closes it.
- Crowd: Represents the collection of passengers in the park.
  - `leave()`: Decrements atomic variable indicating # passengers in museum.
  - `alive()`: Returns whether any passengers left. Used for Car thread termination.
  - The Constructor and Destructor spawn and join threads respectively.
- Time: Used for calculating the average ride time (as required by the question).
  - `exit()`: Records end of passenger thread.
  - `power_off()`: Records end of car thread.
  - `out()`: Writes average ride time to file.
  - The Constructor reserves space for the `vector<chrono::duration<double>>` and opens two semaphores. The Destructor unlinks the semaphores.
- CarStatus: Enum representing the three states of a car.

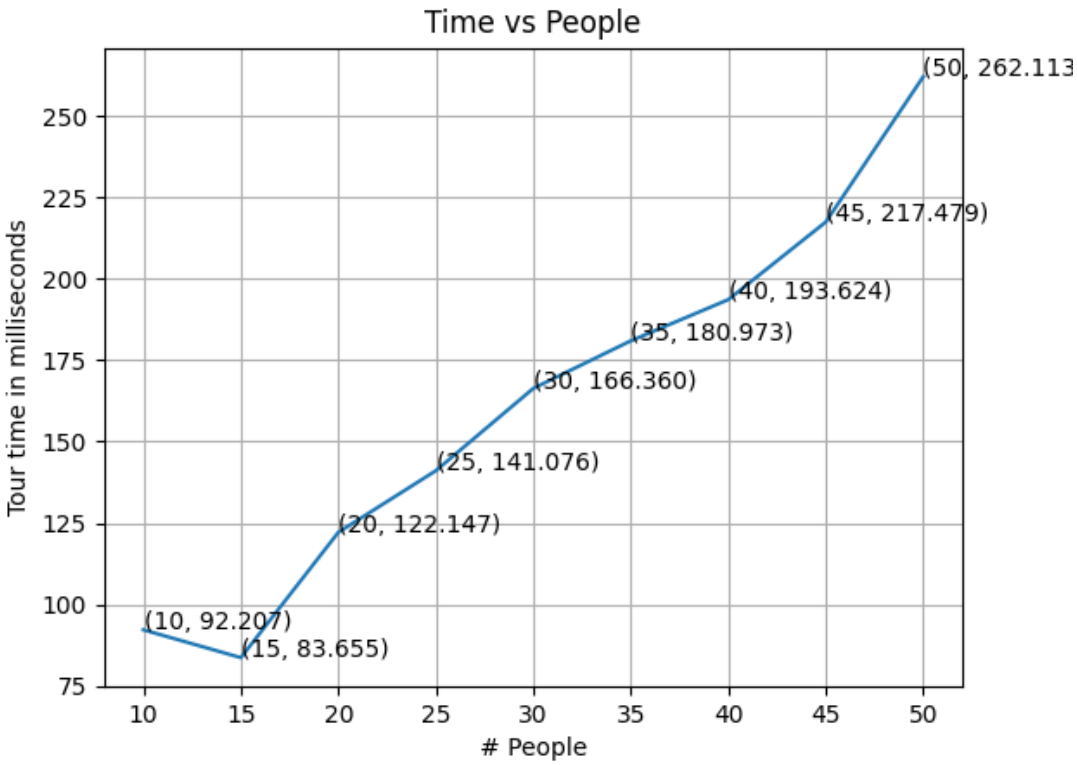
2.2. **Functions.**

- `cars_init()`: Starting point for Car threads. Mostly passive, resulting in scheduling abnormalities causing a sporadic segmentation fault (fixed). Logs readiness to file, busy waits until status changes to `Service`, then logs occupancy to file, busy waits until status changes to `Cooldown`, then sleeps. Repeat while any passenger thread is active.
- `passengers_init()`: Starting point for Passenger threads. For k iterations,
  - Wander: sleep for a random duration.
  - Perform request and acquire a Car.
  - Yield - fix to the aforementioned segfault.
  - Ride: sleep for a random duration.
  - Release the Car.

  with appropriate IO as mentioned in the question statement.
- `now()`: Returns current system time.

3. Graphs

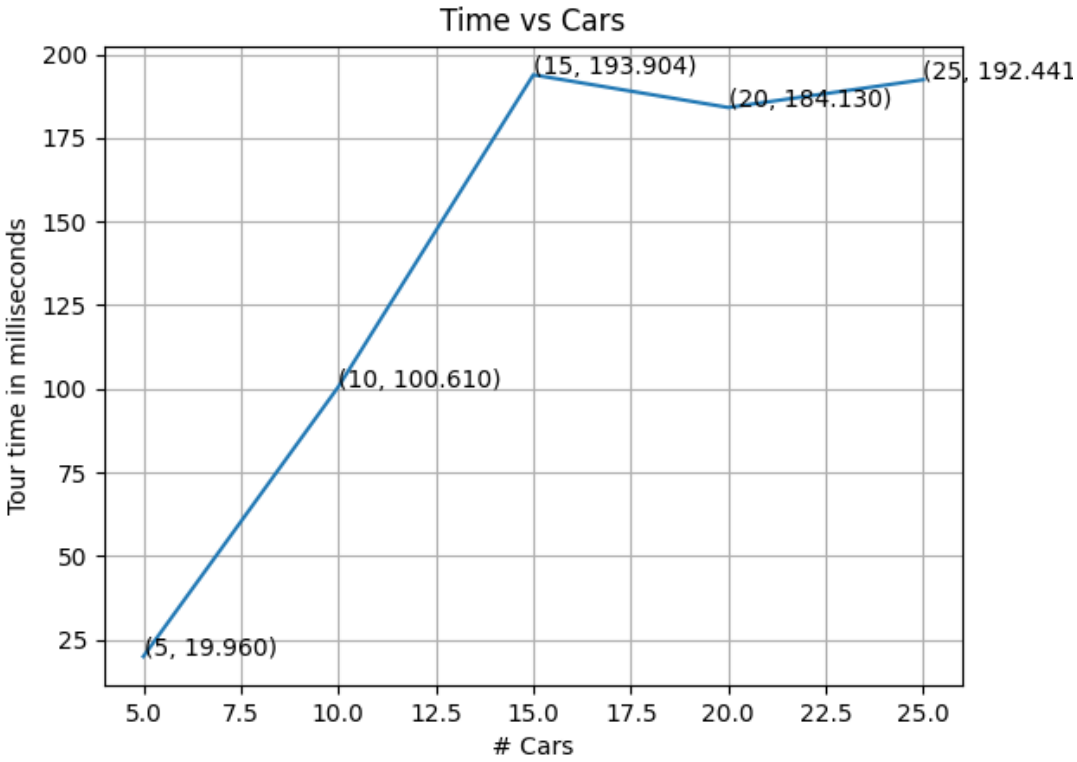
3.1. **Vary Passengers.** .



The graph is as one would expect, with a clear linear increase in time as the number of passengers increase due to contention for cars as well as for the cores.

Due to the compounding effects of contention, the slope of the graph increases slightly for higher # threads.

3.2. **Vary Cars.** .



The graph is somewhat atypical, because while an increase in # car threads would decrease contention among passengers for cars, it increases contention among threads for cores, and clearly the latter effect dominates for small # threads - the time taken increases linearly - but balances out for larger # threads.

In earlier versions, this was not the case, and tour time showed a marked decline with increase in # cars. However, the `yield` call - required to prevent starvation and subsequent segmentation faults - in `passenger_init()` reduced passenger contention, and thus we have this graph.