Henry Yu
CPSC 335-11
Project 1

Algorithm 1:

Pseudocode:

```
Function seat_swaps(row<vector>):
        count = 0
        for i from 0 to length(row) - 2, i increments by 2
                if row[i] + 1 != row[i + 1] or row[i] - 1 != row[i + 1]:   // Check if each pair is a
                couple
                        for j from i + 1 to length(row) - 1:
                if row[i] + 1 == row[i + 1] or row[i] - 1 == row[i + 1]:   // Find matching person
                        swap(row[i + 1], row[j])   // Swap match
                        count++
                break   // Prevents over incrementation
        return count
```

Efficiency Analysis:
I will analyze this algorithm using the step count method:
There are an *n* total number of individuals, there are an *m* number of couples, the relationship between *n* and *m* being *m* = *n*/2.
The total amount of steps in this algorithm is *m* * *n* / 2, this is because the outer loop of the algorithm must *at least* run *m* number of times, as we must check through every couple pairing, while the inside loop is checking two people at a time reducing the amount of loops to the number of people *n* / 2. So for *m* loops through the outer loop, two people are compared an *n / 2* amount of times, therefore the total amount of steps is represented by *m* * *n* / 2 or *mn*/2. Because both *n* and *m* are linear in complexity the resulting algorithm is O(n) complexity.

Algorithm 2:
Pseudocode:

```
Function optimal_start_city(city_distances<vector>, city_fuel<vector>, car_mpg):
        n = total number of cities in vector
                for start from 0 to n, start increments by 1's
                        check = 1 // used as a flag to bail if starting city is not a valid start
                        car_fuel = 0
                        for i from 0 to n
                                j = (i + start) % 2 // index of current city
                                fuel = city_fuel[j]
```

```
            distance = city_distance[j]
            car_fuel += fuel * car_mpg - distance //subtract fuel for driving
            if car_fuel < 0
                check = 0 // sets bailout condition for invalid city
        if check == 1
            return start
```

Efficiency Analysis:

I will be using step count analysis to determine the efficiency class of this algorithm. Let *n* = the number of cities in our test vector. Both the outside loop and inside loop of my algorithm loops at least *n* number of times. Which means the worst case scenario for this algorithm is that the algorithm runs an *n*-squared number of times. But because both the inside and outside loop of the algorithm has a complexity of O(n) the resulting algorithm is also O(n) complexity