Temporal complexity

```
public int calculatePriorityEntry(Passenger passenger, int arrivalTime) {
   int priority = 0; // 1
   int coefficient = 100000; // 1
   String rowString = passenger.getSeat(); // 1
   int seat = Integer.parseInt(rowString.substring(1)); // 1
   priority += coefficient * seat; // 1
   coefficient = 1; // 1
   if (passenger.isFirstClass()) { // 1
      if (passenger.isSpecialAttention()) { // 1
         priority += coefficient * (passenger.getAge() >= 60 ? 5000 : 2500); // 1
      } else if (passenger.getAge() >= 60) { // 1
         priority += coefficient * 2500; // 1
      }
      priority += coefficient * passenger.getAccumulatedMiles(); // 1
   }
   coefficient = 5; // 1
   priority -= coefficient * arrivalTime; // 1
   return priority; // 1
}
```

Since there are if-else structures in the method, for the calculation of the time complexity we take only the most complex part.

TOTAL time complexity: 1+1+1+1+1+1+1+1+1+1+1+1+1 = 13
Time complexity in big O notation: *O(1)*

```
public void readGson() {
   Gson gson = new Gson(); // 1
   int NUMBER_OF_SEATS = 48; // 1
   passengers = new Hash<>(NUMBER_OF_SEATS); // 1
   File projectDir = new File(System.getProperty("user.dir")); // 1
   File dataDirectory = new File(projectDir + "\\data"); // 1
   File passengerInformation = new File(dataDirectory + "\\data.json"); // 1

   if (!dataDirectory.exists()) { // 1
      dataDirectory.mkdir(); // 1
   }
   try {
      if (!passengerInformation.exists()) { // 1
         passengerInformation.createNewFile(); // 1
      }
      String json = new String(java.nio.file.Files.readAllBytes(passengerInformation.toPath())); // n. The complexity of reading all
bytes from the file depends on the size n of the file.
      Passenger[] passengersJson = gson.fromJson(json, Passenger[].class); // n. JSON deserialization has a linear complexity that
depends on the size of the JSON.
      for (Passenger passenger : passengersJson) { // n + 1
         passengers.insert(passenger.getIdentification(), passenger); // n
      }
   } catch (IOException e) { // 1
      e.printStackTrace(); // 1
   } catch (HashException he) { // 1
      System.err.println(he.getMessage()); // 1
   }
}
```

The worst case complexity occurs when the method has to load n elements from the Json file into the program.
TOTAL time complexity: 1+1+1+1+1+1+1+1+n+n+n+1+n = 4n+9
Time complexity in big O notation: *O(n)*

Spatial complexity

calculatePriorityEntry method

TYPE	VARIABLE	LENGHT OF 1 ATOMIC VALUE	AMOUNT OF ATOMIC VALUES
Input	passenger	32 bits	1
	arrivalTime	32 bits	1
Auxiliary	coefficient	32 bits	1
	NUMBER_OF_SEATS	32 bits	1
	seat	32 bits	1
Output	priority	32 bits	1

Output + Auxiliary space complexity= 1+1+1+1 = 4
Total space complexity= Input + Auxiliary + Output = 1+1+1+1+1+1 = O(1)

readGson method

TYPE	VARIABLE	LENGHT OF 1 ATOMIC VALUE	AMOUNT OF ATOMIC VALUES
Input	none	0 bits	0
Auxiliary	gson	32 bits	1
	NUMBER_OF_SEATS	32 bits	1
	projectDir	32 bits	1
	dataDirectory	32 bits	1
	passengerInformation	32 bits	1
	json	32 bits	1
	passengersJson	32 bits	n
	passenger	32 bits	1
Output	passengers	32 bits	1

Output + Auxiliary space complexity= 1+1+1+1+1+1+1+n+1 = n + 8
Total space complexity= Input + Auxiliary + Output = 0+1+1+1+1+1+1+1+n+1 = O(n)