Hotel Normalization Answers

a. Examples of anomalies: In order to identify insert/delete anomalies we need to first identify the primary key which is (guestNo, dateFrom) or (guestNo, dateTo). Note that you could also choose (hotelNo, roomNo, dateFrom) or (hotelNo, roomNo, dateFrom) as primary keys and then your insert/update anomalies would be somewhat different.
   1. insert: If we want to insert a new hotel, there is no way to do so unless we insert NULLs for guestNo and dateFrom, which is a primary key
   2. insert: if we want to insert a new guest, there is no way to do so unless the guest also makes a reservation
   3. insert: if we want to insert a new room at a hotel, there is no way to do so unless a guest also makes a reservation for that room or we insert NULLs for guestNo and dateFrom, which is the primary key
   4. update: if we change the address of a hotel, we must change it everywhere
   5. update: if we change the address of a guest, we must change it everywhere
   6. delete: if we delete all entries related to a guest, we lose the guest's address
   7. delete: if we delete all reservations at a hotel, we lose the hotel's information
   8. delete: if we delete all reservations for a room at a hotel, we lose the room's information

b. The functional dependencies can be derived directly from the 5 assumptions you were given:
   1. HotelNo -> HotelName HotelZip
   2. GuestNo -> GuestName GuestZip
   3. RoomNo, HotelNo -> RoomType RoomPrice
   4. GuestNo, DateFrom -> HotelNo, DateTo, RoomNo
   5. GuestNo, DateTo -> HotelNo, DateFrom, RoomNo
   6. HotelZip -> HotelCity
   7. GuestZip -> GuestCity
   8. HotelNo, RoomNo, DateFrom -> GuestNo, DateTo
   9. HotelNo, RoomNo, DateTo -> GuestNo, DateFrom

c. The candidate keys for the relation are:
   1. GuestNo, DateFrom
   2. GuestNo, DateTo
   3. HotelNo, RoomNo, DateFrom
   4. HotelNo, RoomNo, DateTo

   I will use (GuestNo, DateFrom) as the primary key because it is the most intuitive and the most compact.

d. To go from 1st to 2nd normal forms we use functional dependencies that are partial dependencies. There is only one such dependency:

```
 GuestNo -> GuestName GuestZip
```

   and then there is a transitive dependency of GuestZip -> GuestCity that forces GuestCity to accompany GuestZip to any new relation, so we decompose our relation into two relations as follows:

   GuestInfo(<u>GuestNo</u>, GuestName, GuestZip, GuestCity)
   Booking(<u>GuestNo</u>, <u>DateFrom</u>, DateTo, HotelNo, HotelName, HotelCity, HotelZip, RoomNo, RoomType, RoomPrice)

e. To go from 2nd to 3rd normal forms we use the following transitive dependencies:
   1. HotelNo -> HotelName HotelZip

2. RoomNo -> RoomType RoomPrice
3. HotelZip -> HotelCity
4. GuestZip -> GuestCity

Because the two zipcode functional dependencies really identify the same relationship of a zip code determining a city, I created just one relation from these two dependencies. If you did not realize that these two dependencies were expressing the same relationship then you would create two different relations. Note however that you are creating redundant information when you do so, which could lead to inconsistency in your database:

```
GuestInfo(GuestNo, GuestName, GuestZip)
Booking(GuestNo, DateFrom, DateTo, HotelNo, RoomNo)
RoomInfo(RoomNo, HotelNo, RoomType, RoomPrice)
HotelInfo(HotelNo, HotelName, HotelZip)
ZipCode(ZipCode, City)
```

Note that we did nothing with dependency 5, which represents an alternative candidate key for the Booking relation. However, note that the integrity constraint represented by this functional dependency has been preserved in the final Booking relation, because all five attributes in that functional dependency are present in the Booking relation.

f. partial dependency
g. transitive dependency