The first step in our hiring process is the following: at the bottom of this e-mail you will find one short programming problem that is designed to take about 2 hours to solve (total time). We understand your time is valuable and you have other commitments, so work on it whenever is best for your schedule - just send us an e-mail reply to let us know when we can expect your solutions. When you return the solutions, we will need to know how long you spent on it –Ideally we are looking for no longer than 2 hours and trust in your integrity to relay the true amount of time spent – if not, the test would be totally counterproductive and would not reflect your true capabilities.

Please send us:

- your solution
- your estimates of the running time and memory consumption of each method
- the amount of time you spent solving the problem.

The code does NOT need to run or even compile - We are more interested in your choice of algorithm and data structures. Please optimize for speed (and use memory efficiently if it does not sacrifice speed). The problem is stated in Java, but if you find it more convenient, you may use C, C++, or Golang.

**Problem: Design/implement a system that manages seating for a Restaurant.**

Our restaurant has round tables. Tables come in different sizes that can accommodate 2, 3, 4, 5 or 6 people. People arrive at our restaurant in groups of 6 or less. People in the same group want to be seated at the same table. You can seat a group at any table that has enough empty seats for them. If it's not possible to accommodate them, they're willing to wait. Once they're seated, they can stay as long as they want and you cannot ask them to move to another table (i.e. you cannot move them to make space for another group). In terms of fairness of seating order: seat groups in the order they arrive, but seat opportunistically. For example: a group of 6 is waiting for a table and there are 4 empty seats at a table for 6; if a group of 2 arrives you may put them at the table for 6 but only if you have nowhere else to put them. This may mean that the group of 6 waits a long time, possibly until they become frustrated and leave.

```
public class Table {
    public final int size; //number of chairs around this table
}

public class CustomerGroup {
    public final int size; //number of people in the group
}

public class SeatingManager {

    /* Constructor */
```

```
    public SeatingManager(List<Table> tables);

    /* Group arrives and wants to be seated. */
    public void arrives(CustomerGroup group);

    /* Whether seated or not, the group leaves the restaurant. */
    public void leaves(CustomerGroup group);

    /* Return the table at which the group is seated, or null if
       they are not seated (whether they're waiting or already left). */
    public Table locate(CustomerGroup group);
}
```

Choose the data structures you need for the SeatingManager and then implement the constructor and public methods (arrives, leaves, locate). You may modify these classes in any way you like, and you may add other