Restaurant Reservations

Sashu’s restaurant chain is having difficulty managing their seating. A given restaurant has several tables, with either 2, 4, 6, or 8 seats. When a party wants to reserve a table, you want to seat them at the smallest available table. Only parties of up to 8 maximum are allowed. If no table is available, that party will have to wait to the side until a large enough table is available.

When a table is cleaned and ready for a new party, it must be filled by the earliest party waiting in line, based on the time of day when they arrived. However, priority will be given to the parties that cannot sit at a smaller table.

For example: Suppose Party A is 2 people, Party B is 3 people, and Party C is 4 people. They arrive in order A, B, C. All are waiting to be seated. Then, a table of four seats is cleared. Party B should be seated, even if Party A arrived first, as Party B cannot use a 2 person table. Party C is not seated either, as B arrived earlier.

You can assume that no two parties will arrive simultaneously.

You are tasked with designing a system that will allow a restaurant employee to manage tables at any of the restaurants, implementing these methods (Please do not write pseudocode; we expect you to convey your design in English).

```java
public class Restaurant {

   // construct a restaurant model that has
   // "two" tables with 2 seats, "four" tables
   // with 4 seats, "six" tables with 6 seats,
   // and "eight" tables with 8 seats
   public createRestaurant(int two, int four,
                           int six, int eight) {}

   // Seat a party of "partySize" people if possible.
   // Place in line for a table otherwise
   public void reserveTable(int partySize) {}

   // Empty and clear table of capacity “seats”.
   // Fill the table if possible.
   public void clearTable(int seats) {}

}
```

1. What ADT(s) best fit the required functionality for this scenario?
   List

2. Describe in 1 to 2 sentences why this ADT is a good fit for this scenario, including any assumptions about how users will behave in this scenario.
   This ADT will allow us to track how many people are waiting in line for a table. The order people arrived matters. We can use int variables to track how many tables of each type are available and filled.

3. What Data Structure implementation(s) of your chosen ADT is optimal for this scenario?
   ArrayList (Array)

4. Implementation and Optimization

   a. Describe how your chosen data structure would be used to implement each of the functions of the program. (A few sentences per function).

      - For createRestaurant(), we create four int variables, one for each size of table, and one ArrayList for the waiting line.

      - For reserveTable(), we first check the amount of free tables for this party's closest size group. If any are free, the party is seated, and that table variable is decremented. If they aren't we check all larger size groups in the same manner. If none are available, we add this party to the waitlist, at the end.

      - For clearTable(), we go through the list of people waiting in line, using a temp variable to hold the party we want to seat. If we find a party that matches the table size or is only 1 less, we seat them. Otherwise, we seat the largest and earliest party that can fit. If no parties are available we increment a table variable.

   b. Describe which functions of the program are optimized by your selected data structure, e.g. What is the most efficient task your ADT accomplishes?
      - This data structure optimizes the reserveTable() function, as it should run in constant time. This is because we only must check at most four table variables, as opposed to the number of people in the waiting list.

   c. Describe which functions (if any) of the program are not optimized by your selected data structure, e.g. What tasks are not fully efficient with your ADT?
      - This data structure does not optimize the clearTable(), as we have to potentially loop over the entire waitlist to see if there is someone who can be seated.

5. Describe the best-case scenario for your design, including discussion of how this impacts each of the programs's functions.

   Best case scenario is when the party at the front of the waiting queue is the same size as the table that has just been cleared. This makes the traversal through the line of people as short as possible

6. What is the Tight Big O of the best-case scenario for each of the functions you described in part 5?

   For both functions, the runtime is O(1).

7. Describe the worst-case scenario for your design, including discussion of how this impacts each of the program's functions.

   The worst-case scenario for this design is when the optimal party size is at the very back of the queue of people. This would cause the clearTable function to traverse through the whole list.

8. What is the Tight Big O of the worst-case scenario for each of the functions you described in part 5?

   For reserveTable(), the runtime remains constant
   For clearTable(), the runtime is O(n), where n is the size of the line of people waiting.