Our Test Cases:

Sample1: This sample simply tests the main algorithm of the program. It represents the free time of the day thus the waiting list will be not tested here. It mainly checks the simulator function. (Easy)

Sample2: This sample is more intense than sample 1. It shows how the interface interact when many order come at the same time and how it react to every time step. (Medium)

- Timestep 8 will clearly show how the orders are handled in each timestep and how the waiting list is used.
- Timestep 9 will check how the cancellation and promotion event should work in this phase. For example, orders 8&5 were cancelled so you can see that they were removed from the wait list. A promotion event was also called for order 11 however it was ignored.

Sample 3: This sample represents the most crowded time of the day, where many orders come at the same time. This sample is the best to fully test the code. As for sample 2 it tests our function fully + in this sample it shows how cancellation cannot be executed if an order is already in the In-Service list. This can be checked at time step 11 were order 12 will not be cancelled since it is in the In-Service list. (Extreme/Hard)

➢ Linked List:

Function	Complexity	Pros & Cons
InsertEnd	O(n)	• Equivalent to the queue but with easier traversal. • All list traversal to the end node.
DeleteFirst	O(1)	• Similar to the dequeue but with the linked list's features.
DeleteNode	O(n)	• Advantage of picking the required node. • All list traversal.

➢ Queue:

Function	Complexity	Pros & Cons
enqueue	O(1)	• Easy insertion.
dequeue	O(1)	• Simple deletion process
peekFront	O(1)	• gets the front of the queue without modifying it. • Only allows you to see the first node.
toArray	O(n+n)=O(n)	• Provides Linked List's traversal advantage.

Data Structure used:

1. **WaitingVIP** – *Sorted Queue*: As mentioned in the documents, VIP will be taken according to several factor not just the arrival time like money and dish size.

2. **WaitingNormal** – *Linked List*: Unlike VIP, no need to sort it since normal orders will be taken by their arrival time; however, looking into phase 2 normal events may later be promoted thus queue will not be a suitable choice as traversing for adding, cancellation will longer than a list.

3. **WaitingVegan** – *Queue*: Vegan orders won't be promoted nor cancelled so traversing wouldn't be needed thus a queue is suitable as FIFO.

4. **Events-Arrival/Cancel/Promo event** – *Queue*: As mentioned in vegan waiting, there is no traversing to cancel/add an event, just retrieve the events.

5. **In-Service/Finished** – *Linked Lists*: These are easier to traverse through than queues as we will add/remove an order in them for every time step.

6. **AllOrders** – *Queue*: Or we can name it our orders drawing queue; in order to draw all the orders sorted either in the waiting, serving, or finished region. The concept of FIFO applies on real life simulation and our project thus a queue was needed.

7. **AllCookQueue** – *Queue:* Our cooks drawing queue; simply by updating the interface with all the available cooks. No need for insertion or deletion just addition to the drawing List.

8. **VIPCookQueue / NormalCookQueue / VeganCookQueue** – *Queue*: Separate lists for each type of cook; in order to check availability and perform assignment without checking the cook's type each time. In addition to, the advantage of easy insertion that a queue provides.

*Note:

We updated the interface counters for the available cooks of each type, regardless that there was no cook assignment or check availability.