List	Data Structure	Complexity	Justification
VIP Orders	Priority Queue	Searching “For enqueuing”: O(N) Insert: O(1)	According to the priority equation, the order with the highest priority is enqueued first.
Normal Orders	Linked List	Searching “to cancel”: O(N) Insert: O(1)	We could have used a normal Queue, but this would have increased the complexity because of the order cancel event, we had to empty the original queue to find the cancelled order then enqueue the orders once again.
Vegan Orders	Queue	Insert: O(1)	There is no priority and there is no order cancellation. So, orders are enqueued in sequence of arrival.
VIP Cooks Normal Cooks Vegan Cooks	Queue	Insert: O(1)	At the start of the simulation, these lists are populated. After the assignment of cooks to orders they are moved to the in – service list then if any of them should have a break then he is moved to the Breaks’ lists then after finishing the break he is moved to this list again. They are separated because vegan orders must be served from vegan cooks so, we can’t make one queue for all free cooks.
VIP in Break Normal in Break Vegan in Break	Queue	Remove “Dequeue” :O(1) Add “Enqueue” :O(1)	If any cook finished his max number of orders then after we remove him from in – service list, we will enqueue him in his break queue. We separated each cook because of the service criteria.
Finished Orders	Queue		To print the finished orders in their order (FIFO).

In Service Orders and Cooks	Linked List	Insert: O(1) Delete: O(N)	We will implement a struct containing the in - service orders and their assigned cooks and then we will insert them in a linked list, due to different time steps and service time of each order.