LabReport08

1. **Problem:** The problem of this lab is to implement a class called OrderScheduler that can schedule and manage orders for a food delivery system. The OrderScheduler should be able to add orders, keep track of the current order being processed, advance time by one minute, calculate average waiting time for orders, and determine if all orders have been processed.

2. **Solution Description:** The code provides a solution by implementing an OrderScheduler class that uses a MinHeap data structure to prioritize orders based on their arrival time. Orders are added to the MinHeap and the current order being processed is tracked. The advanceOneMinute() method simulates the passage of time by updating the current minute, checking if the current order is done cooking, and if so, removing it from the MinHeap and updating the waiting time. If the current order is null, the next order from the MinHeap is taken as the current order. The class also provides methods to get the current order, current minute, total orders, and average waiting time. The isDone() method checks if all orders have been processed.

3. **Problems Encountered:** There were no syntax, run-time, or logic errors encountered while implementing the solution.

4. The Big O complexity of inserting one element into a heap is O(log n), where n is the number of elements in the heap. This is because the element needs to be inserted at the last leaf node of the heap and then may need to be swapped up the heap to maintain the heap property, which takes O(log n) time in the worst case.

5. The Big O complexity of heap sort is O(n log n), where n is the number of elements in the array to be sorted. This is because heap sort involves two main operations: building a heap (O(n)) and repeatedly extracting the maximum (or minimum) element and restoring the heap property (O(log n)), and this process is done n times.

LabReport08

6. **Max Heap structure:**

```
        8
      /    \
     4      9
    /  \   /  \
   1    2 16   12
```

7. **Max Heap structure:**

```
         7
      /     \
     8       16
    /  \     /  \
   12   21  15   21
```

8. The index of the left child of a node in an Array Heap with root index 0 can be calculated using the formula: **2 * parentIndex + 1.**

9. The index of the right child of a node in an Array Heap with root index 0 can be calculated using the formula: **2 * parentIndex + 2**.

10. The index of the parent of a node in an Array Heap with root index 0 can be calculated using the formula: **(index - 1) / 2**, where **index** is the index of the node for which we want to find the parent.