✅ Queues

📌 Problem Statement:

A truck wants to complete a circular tour of petrol pumps.

- Each petrol pump has `petrol` liters and is at a certain distance from the next pump.
- The truck consumes 1 liter of petrol per unit distance.
- Find the **starting petrol pump index** from which the truck can complete the full circle without running out of petrol.

📥 Example:

```
Input:
petrolpumps = [(4, 6), (6, 5), (7, 3), (4, 5)]

Output:
1

Explanation:
Starting at pump 1 allows completing the circle.
```

🔁 Brute Force:

- Try starting from each pump.
- Check if truck completes circle.
- O(n^2) time, inefficient.

🚀 Optimal Approach:

- Use a greedy approach with two pointers.
- Keep track of current petrol and deficit.
- If current petrol becomes negative, move start to next pump and add deficit.
- At the end, if total petrol >= total distance, starting point found.

✅ Java Code:

```java
public class TruckTour {
    public static int truckTour(int[][] pumps) {
        int n = pumps.length;
        int start = 0;
        int petrol = 0;
        int deficit = 0;
        for (int i = 0; i < n; i++) {
            petrol += pumps[i][0] - pumps[i][1];  // petrol gained - distance
            if (petrol < 0) {
                deficit += petrol;
                petrol = 0;
                start = i + 1;
            }
        }

        return (petrol + deficit) >= 0 ? start : -1;
    }

    public static void main(String[] args) {
        int[][] pumps = {{4,6}, {6,5}, {7,3}, {4,5}};
        System.out.println(truckTour(pumps));  // Output: 1
    }
}
```

✅ Output:

```
1
```

✅ Queues / Sorting – Jim and the Orders

📌 Problem Statement:

Jim runs a restaurant. Customers place orders at different times, and each order takes some time to prepare.

- You need to output the order in which Jim will serve the customers.
- Customers are served by the order of their completion time (order time + preparation time).
- If two orders complete at the same time, serve the customer with the smaller customer ID first.

📥 Example:

```
Input:
orders = [[1, 3], [2, 3], [3, 3]]

Output:
1 2 3

Explanation:
Completion times: 4, 5, 6 → served in order 1, 2, 3
```

🔁 Brute Force:

- Calculate completion times.
- Sort by completion time and customer ID.
- O(n log n) time.

🚀 Optimal Approach:

- Same as brute force because sorting is efficient for this problem.

✅ Java Code:

```java
import java.util.*;

public class JimAndTheOrders {
    public static int[] jimOrders(int[][] orders) {
        int n = orders.length;
        int[][] completion = new int[n][2]; // [completion_time, customer_id]

        for (int i = 0; i < n; i++) {
            completion[i][0] = orders[i][0] + orders[i][1];
            completion[i][1] = i + 1;  // customer IDs are 1-based
        }

        Arrays.sort(completion, (a, b) -> {
            if (a[0] != b[0]) return a[0] - b[0];
            else return a[1] - b[1];
        });

        int[] result = new int[n];
        for (int i = 0; i < n; i++) {
            result[i] = completion[i][1];
        }
        return result;
    }

    public static void main(String[] args) {
        int[][] orders = {{1, 3}, {2, 3}, {3, 3}};
        int[] result = jimOrders(orders);
        for (int id : result) {
            System.out.print(id + " ");
        }
        // Output: 1 2 3
    }
}
```

✅ Output:

```
1 2 3
```