import java.util.\*;

class Process {

    int pid;    // Process ID

    int arrivalTime;   // Arrival Time

    int burstTime;     // Burst Time

    int remainingBurstTime; // Remaining Burst Time

    int completionTime; // Completion Time

    int turnaroundTime; // Turnaround Time

    int waitingTime;    // Waiting Time

    public Process(int pid, int arrivalTime, int burstTime) {

        this.pid = pid;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.remainingBurstTime = burstTime; // initially, remaining burst time is the same as burst time

    }

}

public class RoundRobin {

    public static void main(String[] args) {

        // Hardcoded inputs for processes (Process ID, Arrival Time, Burst Time)

        Process[] processes = {

            new Process(1, 0, 5),  // Process ID 1, Arrival Time 0, Burst Time 5

            new Process(2, 1, 3),  // Process ID 2, Arrival Time 1, Burst Time 3

            new Process(3, 2, 1),  // Process ID 3, Arrival Time 2, Burst Time 1

            new Process(4, 3, 2),  // Process ID 4, Arrival Time 3, Burst Time 2

            new Process(5, 4, 3)   // Process ID 5, Arrival Time 4, Burst Time 3

        };

        int timeQuantum = 2;  // Time quantum for Round Robin

        // Sorting processes by arrival time

        Arrays.sort(processes, Comparator.comparingInt(p -> p.arrivalTime));

        int currentTime = 0;

        int completedProcesses = 0;

        int totalTurnaroundTime = 0;

        int totalWaitingTime = 0;

        Queue<Process> readyQueue = new LinkedList<>();

        int i = 0;

        // Round Robin scheduling

        while (completedProcesses < processes.length) {

            // Add all processes that have arrived by the current time to the ready queue

            while (i < processes.length && processes[i].arrivalTime <= currentTime) {

                readyQueue.add(processes[i]);

                i++;

            }

            if (readyQueue.isEmpty()) {

                // No processes are ready to run, increment time and check again

                currentTime++;

                continue;

            }

            // Get the process from the ready queue

            Process p = readyQueue.poll();

            // If the burst time is less than or equal to time quantum, complete the process

            if (p.remainingBurstTime <= timeQuantum) {

                currentTime += p.remainingBurstTime;

                p.remainingBurstTime = 0;

                p.completionTime = currentTime;

                completedProcesses++;

            } else {

                // Otherwise, process for the full time quantum

                currentTime += timeQuantum;

                p.remainingBurstTime -= timeQuantum;

                readyQueue.add(p);  // Add process back to the ready queue

            }

            // If the process is completed, calculate turnaround time and waiting time

            if (p.remainingBurstTime == 0) {

                p.turnaroundTime = p.completionTime - p.arrivalTime;

                p.waitingTime = p.turnaroundTime - p.burstTime;

                totalTurnaroundTime += p.turnaroundTime;

                totalWaitingTime += p.waitingTime;

            }

        }

        // Output the results

        System.out.println("PID  Arrival Time  Burst Time  Completion Time  Turnaround Time  Waiting Time");

        for (Process p : processes) {

            System.out.printf("%-4d %-13d %-11d %-16d %-17d %-13d\n", p.pid, p.arrivalTime, p.burstTime, p.completionTime, p.turnaroundTime, p.waitingTime);

        }

        // Calculate average turnaround time and waiting time

        double avgTAT = (double) totalTurnaroundTime / processes.length;

        double avgWT = (double) totalWaitingTime / processes.length;

        System.out.println("\nAverage Turnaround Time: " + avgTAT);

        System.out.println("Average Waiting Time: " + avgWT);

    }

}