import java.util.\*;

class Process {

    int pid;          // Process ID

    int arrivalTime;  // Arrival Time

    int burstTime;    // Burst Time

    int remainingBurstTime; // Remaining Burst Time

    int priority;     // Priority (lower value indicates higher priority)

    int completionTime; // Completion Time

    int turnaroundTime; // Turnaround Time

    int waitingTime;    // Waiting Time

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

        this.pid = pid;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.remainingBurstTime = burstTime; // Initially, the remaining burst time is equal to burst time

        this.priority = priority;

    }

}

public class PriorityPreemptive {

    public static void main(String[] args) {

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

        Process[] processes = {

            new Process(1, 0, 8, 2),   // Process ID 1, Arrival Time 0, Burst Time 8, Priority 2

            new Process(2, 1, 4, 1),   // Process ID 2, Arrival Time 1, Burst Time 4, Priority 1

            new Process(3, 2, 9, 3),   // Process ID 3, Arrival Time 2, Burst Time 9, Priority 3

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

        };

        // Sort the processes based on arrival time

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

        int currentTime = 0;

        int completedProcesses = 0;

        int totalTurnaroundTime = 0;

        int totalWaitingTime = 0;

        int n = processes.length;

        boolean[] isCompleted = new boolean[n];

        // Priority Preemptive Scheduling

        while (completedProcesses < n) {

            int index = -1;

            int highestPriority = Integer.MAX\_VALUE;

            // Find the process with the highest priority among the arrived processes

            for (int i = 0; i < n; i++) {

                if (!isCompleted[i] && processes[i].arrivalTime <= currentTime && processes[i].priority < highestPriority) {

                    highestPriority = processes[i].priority;

                    index = i;

                }

            }

            if (index == -1) {

                // If no process is ready to run, increment time

                currentTime++;

                continue;

            }

            Process p = processes[index];

            // Process the selected process

            p.remainingBurstTime--;

            // If the process is completed

            if (p.remainingBurstTime == 0) {

                p.completionTime = currentTime + 1;

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

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

                isCompleted[index] = true;

                completedProcesses++;

                totalTurnaroundTime += p.turnaroundTime;

                totalWaitingTime += p.waitingTime;

            }

            currentTime++; // Increment time by 1 unit

        }

        // Output the results

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

        for (Process p : processes) {

            System.out.printf("%-4d %-13d %-11d %-9d %-16d %-17d %-13d\n", p.pid, p.arrivalTime, p.burstTime, p.priority, p.completionTime, p.turnaroundTime, p.waitingTime);

        }

        // Calculate average turnaround time and waiting time

        double avgTAT = (double) totalTurnaroundTime / n;

        double avgWT = (double) totalWaitingTime / n;

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

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

    }

}