import java.util.\*;

class Process {

    int pid;          // Process ID

    int arrivalTime;  // Arrival Time

    int burstTime;    // Burst Time

    int priority;     // Priority (higher 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.priority = priority;

    }

}

public class PriorityNonPreemptive {

    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

        };

        // 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;

        boolean[] isCompleted = new boolean[processes.length];

        while (completedProcesses < processes.length) {

            int index = -1;

            int highestPriority = Integer.MIN\_VALUE;

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

            for (int i = 0; i < processes.length; i++) {

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

                    highestPriority = processes[i].priority;

                    index = i;

                }

            }

            // If no process is found (i.e., all processes have not arrived yet), increment time

            if (index == -1) {

                currentTime++;

                continue;

            }

            // Process with highest priority starts execution

            Process p = processes[index];

            p.completionTime = currentTime + p.burstTime;

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

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

            // Mark the process as completed

            isCompleted[index] = true;

            completedProcesses++;

            currentTime = p.completionTime;

            // Accumulate total turnaround time and waiting time

            totalTurnaroundTime += p.turnaroundTime;

            totalWaitingTime += p.waitingTime;

        }

        // 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 / processes.length;

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

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

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

    }

}