import java.util.\*;

class Process {

int id;

int arrivalTime;

int burstTime;

int remainingTime;

int completionTime;

int turnaroundTime;

int waitingTime;

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

this.id = id;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.remainingTime = burstTime;

}

}

public class SchedulingAlgorithms {

// First Come First Serve (FCFS)

private static void FCFS(List<Process> processes) {

int currentTime = 0;

System.out.println("\nFCFS Scheduling:");

System.out.printf("%-10s %-12s %-10s %-15s %-15s %-15s\n", "Process", "Arrival", "Burst", "Completion", "Turnaround", "Waiting");

for (Process p : processes) {

if (currentTime < p.arrivalTime) {

currentTime = p.arrivalTime; // Wait for the process to arrive

}

currentTime += p.burstTime;

p.completionTime = currentTime;

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

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

System.out.printf("%-10d %-12d %-10d %-15d %-15d %-15d\n", p.id, p.arrivalTime, p.burstTime, p.completionTime, p.turnaroundTime, p.waitingTime);

}

}

// Shortest Job First (Preemptive)

private static void SJFPreemptive(List<Process> processes) {

int currentTime = 0;

int completed = 0;

int n = processes.size();

List<Process> waitingQueue = new ArrayList<>();

Set<Process> completedProcesses = new HashSet<>(); // To track completed processes

System.out.println("\nSJF Preemptive Scheduling:");

System.out.printf("%-10s %-12s %-10s %-15s %-15s %-15s\n", "Process", "Arrival", "Burst", "Completion", "Turnaround", "Waiting");

while (completed < n) {

// Add processes to the waiting queue based on arrival time

for (Process p : processes) {

if (p.arrivalTime <= currentTime && p.remainingTime > 0 && !completedProcesses.contains(p) && !waitingQueue.contains(p)) {

waitingQueue.add(p);

}

}

if (waitingQueue.isEmpty()) {

currentTime++; // No process is available, move time forward

continue;

}

// Find the process with the smallest remaining time

Process shortestJob = waitingQueue.get(0);

for (Process p : waitingQueue) {

if (p.remainingTime < shortestJob.remainingTime) {

shortestJob = p;

}

}

// Execute the process for one time unit

shortestJob.remainingTime--;

// If the process is completed

if (shortestJob.remainingTime == 0) {

completed++;

shortestJob.completionTime = currentTime + 1;

shortestJob.turnaroundTime = shortestJob.completionTime - shortestJob.arrivalTime;

shortestJob.waitingTime = shortestJob.turnaroundTime - shortestJob.burstTime;

completedProcesses.add(shortestJob); // Mark process as completed

waitingQueue.remove(shortestJob);

System.out.printf("%-10d %-12d %-10d %-15d %-15d %-15d\n", shortestJob.id, shortestJob.arrivalTime, shortestJob.burstTime, shortestJob.completionTime, shortestJob.turnaroundTime, shortestJob.waitingTime);

}

currentTime++; // Increment time

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

List<Process> processes = new ArrayList<>();

// Input number of processes

System.out.print("Enter the number of processes: ");

int numberOfProcesses = scanner.nextInt();

// Input arrival time and burst time for each process

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

System.out.print("Enter arrival time for process " + (i + 1) + ": ");

int arrivalTime = scanner.nextInt();

System.out.print("Enter burst time for process " + (i + 1) + ": ");

int burstTime = scanner.nextInt();

processes.add(new Process(i + 1, arrivalTime, burstTime));

}

// Sort processes based on arrival time (important for both algorithms)

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

System.out.println("Select Scheduling Algorithm: \n1. FCFS \n2. SJF (Preemptive)");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

FCFS(processes);

break;

case 2:

SJFPreemptive(processes);

break;

default:

System.out.println("Invalid choice!");

}

scanner.close();

}

}