import java.util.ArrayDeque;

import java.util.Queue;

import java.util.Scanner;

public class B2 {

    void fcfs(int n, int p[], int at[], int bt[], int wt[], int tat[], int ct[], int rt[]) {

        wt[0] = 0;

        ct[0] = bt[0];

        tat[0] = bt[0];

        rt[0] = 0;

        for (int i = 1; i < n; i++) {

            ct[i] = ct[i - 1] + bt[i];

            tat[i] = ct[i] - at[i];

            wt[i] = tat[i] - bt[i];

            rt[i] = wt[i];

        }

        int awt = 0;

        int atat = 0;

        int art = 0;

        for (int i = 0; i < n; i++) {

            awt += wt[i];

            atat += tat[i];

            art += rt[i];

        }

        float w = (float) awt / n;

        float t = (float) atat / n;

        float r = (float) art / n;

        display(n, p, at, bt, wt, tat, ct, rt);

        System.out.println("\nAverage Waiting Time - " + w + "\nAverage Turnaround Time - " + t + "\nAverage Response Time - " + r);

    }

    void roundRobin(int n, int p[], int at[], int bt[], int quantum) {

        int[] remainingTime = new int[n];

        int[] ct = new int[n];

        int[] tat = new int[n];

        int[] wt = new int[n];

        int[] rt = new int[n];

        boolean[] firstResponse = new boolean[n];

        int currentTime = 0;

        Queue<Integer> readyQueue = new ArrayDeque<>();

        for (int i = 0; i < n; i++) {

            remainingTime[i] = bt[i];

            firstResponse[i] = false;

        }

        for (int i = 0; i < n; i++) {

            if (at[i] <= currentTime) {

                readyQueue.offer(i);

            }

        }

        while (!readyQueue.isEmpty()) {

            int currentProcess = readyQueue.poll();

            if (!firstResponse[currentProcess]) {

                rt[currentProcess] = currentTime - at[currentProcess];

                firstResponse[currentProcess] = true;

            }

            int executionTime = Math.min(quantum, remainingTime[currentProcess]);

            remainingTime[currentProcess] -= executionTime;

            currentTime += executionTime;

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

                if (at[i] <= currentTime && remainingTime[i] > 0 && !readyQueue.contains(i) && i != currentProcess) {

                    readyQueue.offer(i);

                }

            }

            if (remainingTime[currentProcess] == 0) {

                ct[currentProcess] = currentTime;

                tat[currentProcess] = ct[currentProcess] - at[currentProcess];

                wt[currentProcess] = tat[currentProcess] - bt[currentProcess];

            } else {

                readyQueue.offer(currentProcess);

            }

        }

        int awt = 0;

        int atat = 0;

        int art = 0;

        for (int i = 0; i < n; i++) {

            awt += wt[i];

            atat += tat[i];

            art += rt[i];

        }

        float w = (float) awt / n;

        float t = (float) atat / n;

        float r = (float) art / n;

        display(n, p, at, bt, wt, tat, ct, rt);

        System.out.println("\nAverage Waiting Time - " + w + "\nAverage Turnaround Time - " + t + "\nAverage Response Time - " + r);

    }

    void priority(int n, int p[], int at[], int bt[], int wt[], int tat[], int ct[], int rt[]) {

        int completed = 0;

        int currentTime = 0;

        boolean[] visited = new boolean[n];

        boolean[] responseAssigned = new boolean[n];

        for (int i = 0; i < n; i++) {

            visited[i] = false;

            responseAssigned[i] = false;

            rt[i] = -1;

        }

        while (completed < n) {

            int highestPriorityProcess = -1;

            int highestPriority = Integer.MAX\_VALUE;

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

                if (!visited[i] && at[i] <= currentTime && p[i] < highestPriority) {

                    highestPriority = p[i];

                    highestPriorityProcess = i;

                }

            }

            if (highestPriorityProcess == -1) {

                currentTime++;

                continue;

            }

            if (!responseAssigned[highestPriorityProcess]) {

                rt[highestPriorityProcess] = currentTime - at[highestPriorityProcess];

                responseAssigned[highestPriorityProcess] = true;

            }

            ct[highestPriorityProcess] = currentTime + bt[highestPriorityProcess];

            tat[highestPriorityProcess] = ct[highestPriorityProcess] - at[highestPriorityProcess];

            wt[highestPriorityProcess] = tat[highestPriorityProcess] - bt[highestPriorityProcess];

            visited[highestPriorityProcess] = true;

            completed++;

            currentTime += bt[highestPriorityProcess];

        }

        int awt = 0, atat = 0, art = 0;

        for (int i = 0; i < n; i++) {

            awt += wt[i];

            atat += tat[i];

            art += rt[i];

        }

        float w = (float) awt / n;

        float t = (float) atat / n;

        float r = (float) art / n;

        display(n, p, at, bt, wt, tat, ct, rt);

        System.out.println("\nAverage Waiting Time : " + w);

        System.out.println("Average Turnaround Time : " + t);

        System.out.println("Average Response Time : " + r);

    }

    void sjf(int n, int p[], int at[], int bt[], int wt[], int tat[], int ct[], int rt[]) {

        int completed = 0;

        int currentTime = 0;

        boolean[] visited = new boolean[n];

        int[] remainingTime = new int[n];

        for (int i = 0; i < n; i++) {

            remainingTime[i] = bt[i];

            rt[i] = -1;

        }

        while (completed < n) {

            int shortestProcess = -1;

            int shortestBurst = Integer.MAX\_VALUE;

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

                if (!visited[i] && at[i] <= currentTime && remainingTime[i] < shortestBurst) {

                    shortestBurst = remainingTime[i];

                    shortestProcess = i;

                }

            }

            if (shortestProcess == -1) {

                currentTime++;

                continue;

            }

            if (rt[shortestProcess] == -1) {

                rt[shortestProcess] = currentTime - at[shortestProcess];

            }

            remainingTime[shortestProcess]--;

            currentTime++;

            if (remainingTime[shortestProcess] == 0) {

                ct[shortestProcess] = currentTime;

                tat[shortestProcess] = ct[shortestProcess] - at[shortestProcess];

                wt[shortestProcess] = tat[shortestProcess] - bt[shortestProcess];

                visited[shortestProcess] = true;

                completed++;

            }

        }

        int awt = 0;

        int atat = 0;

        int art = 0;

        for (int i = 0; i < n; i++) {

            awt += wt[i];

            atat += tat[i];

            art += rt[i];

        }

        float w = (float) awt / n;

        float t = (float) atat / n;

        float r = (float) art / n;

        display(n, p, at, bt, wt, tat, ct, rt);

        System.out.println("\nAverage Waiting Time : " + w + "\nAverage Turnaround Time : " + t + "\nAverage Response Time : " + r);

    }

    void display(int n, int p[], int at[], int bt[], int wt[], int tat[], int ct[], int rt[]) {

        System.out.println(" Process No. || Arr. Time || Burst Time || Compl. Time || Turnaround Time || Waiting Time || Response Time\n");

        for (int i = 0; i < n; i++) {

            System.out.printf("  " + p[i] + "  |  " + at[i] + "  |  " + bt[i] + "  |  " + ct[i] + "  |  " + tat[i] + "  |  " + wt[i] + "  |  " + rt[i] + "\n");

        }

    }

    public static void main(String[] args) {

        B2 a = new B2();

        int ch;

        try (Scanner sc = new Scanner(System.in)) {

            int n;

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

            n = sc.nextInt();

            int p[] = new int[n];

            int at[] = new int[n];

            int bt[] = new int[n];

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

                System.out.println("\nEnter process " + i + " Details");

                p[i] = (i + 1);

                System.out.println("Enter arrival time : ");

                at[i] = sc.nextInt();

                System.out.println("Enter burst time : ");

                bt[i] = sc.nextInt();

            }

            do {

                System.out.println("\n\nSelect Scheduling Algorithm - \n1.FCFS Algorithm(Non-Preemptive)\n2.Round Robin Algorithm(Preemptive)\n3.Priority Algorithm(Non-Preemptive)\n4.SJF Algorithm(Preemptive)\nEnter choice : ");

                ch = sc.nextInt();

                switch (ch) {

                    case 1:

                        a.fcfs(n, p, at, bt, new int[n], new int[n], new int[n], new int[n]);

                        break;

                    case 2:

                        int quantum;

                        System.out.println("Enter time quantum for Round Robin - ");

                        quantum = sc.nextInt();

                        a.roundRobin(n, p, at, bt, quantum);

                        break;

                    case 3:

                        int[] pr = new int[n];

                        System.out.println("Enter the Priorities : ");

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

                            pr[i] = sc.nextInt();

                        }

                        a.priority(n, pr, at, bt, new int[n], new int[n], new int[n], new int[n]);

                        break;

                    case 4:

                        a.sjf(n, p, at, bt, new int[n], new int[n], new int[n], new int[n]);

                        break;

                    default:

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

                        break;

                }

            } while (ch < 5);

        }

    }

}

/\*

FCFS -

 Process No. || Arr. Time || Burst Time || Compl. Time || Turnaround Time || Waiting Time || Response Time

  1  |  0  |  5  |  5  |  5  |  0  |  0

  2  |  1  |  3  |  8  |  7  |  4  |  4

  3  |  2  |  8  |  16  |  14  |  6  |  6

  4  |  3  |  6  |  22  |  19  |  13  |  13

Average Waiting Time - 5.75

Average Turnaround Time - 11.25

Average Response Time - 5.75

RR -

 Process No. || Arr. Time || Burst Time || Compl. Time || Turnaround Time || Waiting Time || Response Time

  1  |  0  |  5  |  12  |  12  |  7  |  0

  2  |  1  |  4  |  11  |  10  |  6  |  1

  3  |  2  |  2  |  6  |  4  |  2  |  2

  4  |  4  |  1  |  9  |  5  |  4  |  4

Average Waiting Time - 4.75

Average Turnaround Time - 7.75

Average Response Time - 1.75

PRIORITY -

 Priority || Arr. Time || Burst Time || Compl. Time || Turnaround Time || Waiting Time || Response Time

  2  |  0  |  3  |  3  |  3  |  0  |  0

  6  |  2  |  5  |  18  |  16  |  11  |  11

  3  |  1  |  4  |  7  |  6  |  2  |  2

  5  |  4  |  2  |  13  |  9  |  7  |  7

  7  |  6  |  9  |  27  |  21  |  12  |  12

  4  |  5  |  4  |  11  |  6  |  2  |  2

  10  |  7  |  10  |  37  |  30  |  20  |  20

Average Waiting Time : 7.714286

Average Turnaround Time : 13.0

Average Response Time : 7.714286

SJF -

 Process No. || Arr. Time || Burst Time || Compl. Time || Turnaround Time || Waiting Time || Response Time

  1  |  2  |  6  |  15  |  13  |  7  |  1

  2  |  5  |  2  |  7  |  2  |  0  |  0

  3  |  1  |  8  |  23  |  22  |  14  |  14

  4  |  0  |  3  |  3  |  3  |  0  |  0

  5  |  4  |  4  |  10  |  6  |  2  |  0

 \*/