import java.util.\*;

// Define a class named SJF (Shortest Job First)

public class SJF {

// Define the main method

public static void main(String args[]) {

// Create a Scanner object to read input from the user

Scanner sc = new Scanner(System.in);

// Ask the user to enter the number of processes

System.out.println("enter no of process:");

int n = sc.nextInt(); // Read the input number of processes

// Define arrays to store process details for each process

int pid[] = new int[n]; // process ids

int at[] = new int[n]; // at means arrival time

int bt[] = new int[n]; // bt means burst time

int ct[] = new int[n]; // ct means complete time

int ta[] = new int[n]; // ta means turn around time

int wt[] = new int[n]; // wt means waiting time

int f[] = new int[n]; // f means it is a flag that checks if a process is completed or not

int st = 0, tot = 0; // st represents the system time, tot tracks the number of completed processes

float avgwt = 0, avgta = 0; // Variables to calculate average waiting time and average turnaround time

// Input process details from the user

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

// Ask for the arrival time of the current process

System.out.println("enter process " + (i+1) + " arrival time:");

at[i] = sc.nextInt(); // Read the arrival time for the current process

// Ask for the burst time of the current process

System.out.println("enter process " + (i+1) + " burst time:");

bt[i] = sc.nextInt(); // Read the burst time for the current process

pid[i] = i + 1; // Assigning process IDs from 1 to n

f[i] = 0; // Initialize the flag to 0, meaning the process is not completed yet

}

// Execute the SJF scheduling algorithm

while (true) {

int c = n, min = 999;

if (tot == n) // If the total number of completed processes equals the total number of processes, terminate the loop

break;

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

/\*

\* If i'th process arrival time <= system time and its flag = 0 and burst < min

\* Then, this process will be executed first as it has the shortest burst time

\*/

if ((at[i] <= st) && (f[i] == 0) && (bt[i] < min)) {

min = bt[i]; // Update the minimum burst time

c = i; // Update the index of the process with the shortest burst time

}

}

/\*

\* If c == n, it means no process has an arrival time < system time, so we

\* increase the system time

\*/

if (c == n)

st++;

else {

ct[c] = st + bt[c]; // Calculate the completion time for the selected process

st += bt[c]; // Update the system time

ta[c] = ct[c] - at[c]; // Calculate the turnaround time for the selected process

wt[c] = ta[c] - bt[c]; // Calculate the waiting time for the selected process

f[c] = 1; // Set the flag to 1 to indicate that the process is completed

tot++; // Increment the count of completed processes

}

}

// Display the process details in tabular form

System.out.println("\npid arrival burst complete turn waiting");

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

avgwt += wt[i]; // Summing up waiting times for calculating the average later

avgta += ta[i]; // Summing up turnaround times for calculating the average later

System.out.println(pid[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" + ta[i] + "\t" + wt[i]);

}

// Print the average turnaround time and average waiting time

System.out.println("\naverage tat is " + (float) (avgta / n));

System.out.println("average wt is " + (float) (avgwt / n));

sc.close(); // Close the Scanner object as we no longer need user input

}

}