**LP-1 SPOS LA3**

**Neeti Kurulkar**

**Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).**

1. **FCFS:**

**import java.util.\*;**

**class Process {**

**String id;**

**int arrival\_time;**

**int burst\_time;**

**int completion\_time;**

**int turn\_around\_time;**

**int waiting\_time;**

**Process() {}**

**Process(String pid, int ar, int br) {**

**id = pid;**

**arrival\_time = ar;**

**burst\_time = br;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.print("Enter Number of Processes: ");**

**int n = sc.nextInt();**

**Process[] process\_queue = new Process[n];**

**Process temp = new Process();**

**String pid;**

**int ar, br;**

**float avgwt = 0, avgtat = 0;**

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

**System.out.print("Enter Arrival Time for Process " + (i + 1) + ": ");**

**ar = sc.nextInt();**

**System.out.print("Enter Burst Time for Process " + (i + 1) + ": ");**

**br = sc.nextInt();**

**process\_queue[i] = new Process(("p" + (i+1)), ar, br);**

**}**

**// Sorting based on arrival time**

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

**for (int j = 0; j < (n - i - 1); j++) {**

**if (process\_queue[j].arrival\_time > process\_queue[j + 1].arrival\_time) {**

**temp = process\_queue[j];**

**process\_queue[j] = process\_queue[j + 1];**

**process\_queue[j + 1] = temp;**

**}**

**}**

**}**

**// Calculating completion, turnaround, and waiting times**

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

**if (i == 0) {**

**process\_queue[i].completion\_time = process\_queue[i].arrival\_time + process\_queue[i].burst\_time;**

**} else {**

**if (process\_queue[i].arrival\_time > process\_queue[i - 1].completion\_time) {**

**process\_queue[i].completion\_time = process\_queue[i].arrival\_time + process\_queue[i].burst\_time;**

**} else {**

**process\_queue[i].completion\_time = process\_queue[i - 1].completion\_time + process\_queue[i].burst\_time;**

**}**

**}**

**process\_queue[i].turn\_around\_time = process\_queue[i].completion\_time - process\_queue[i].arrival\_time;**

**process\_queue[i].waiting\_time = process\_queue[i].turn\_around\_time - process\_queue[i].burst\_time;**

**avgwt += process\_queue[i].waiting\_time;**

**avgtat += process\_queue[i].turn\_around\_time;**

**}**

**// Display the process details**

**System.out.println("--------------------------------------------------------------------");**

**System.out.println("Process\t\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time");**

**System.out.println("--------------------------------------------------------------------");**

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

**System.out.println(process\_queue[i].id + "\t\t" + process\_queue[i].arrival\_time + "\t\t" + process\_queue[i].burst\_time + "\t\t" + process\_queue[i].completion\_time + "\t\t" + process\_queue[i].turn\_around\_time + "\t\t" + process\_queue[i].waiting\_time);**

**}**

**// Display average waiting and turnaround times**

**System.out.println("Average Waiting Time = " + (avgwt / n));**

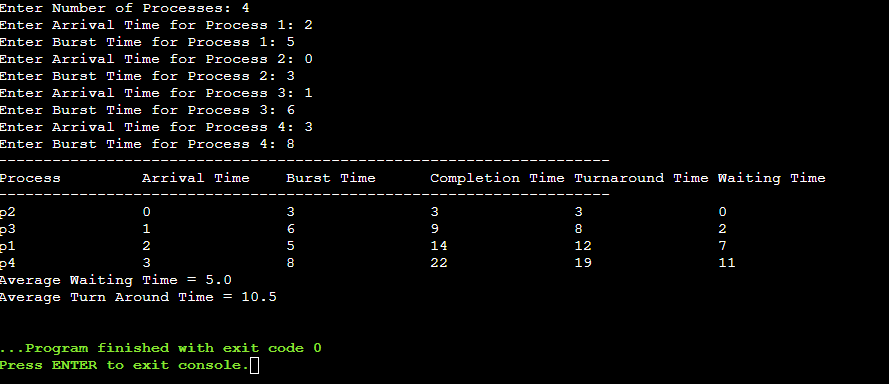
**System.out.println("Average Turn Around Time = " + (avgtat / n));**

**sc.close();**

**}**

**}**

**Output:**

****

1. **SJF (Preemptive):**

**import java.util.Scanner;**

**class Process {**

**String id;**

**int arrival\_time;**

**int burst\_time;**

**int completion\_time;**

**int turn\_around\_time;**

**int waiting\_time;**

**int remaining\_time;**

**boolean isCompleted;**

**Process(String pid, int ar, int br) {**

**id = pid;**

**arrival\_time = ar;**

**burst\_time = br;**

**remaining\_time = br;**

**isCompleted = false;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

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

**int n = sc.nextInt();**

**Process[] process\_queue = new Process[n];**

**int total\_completed = 0, current\_time = 0;**

**float avgwt = 0, avgtat = 0;**

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

**System.out.print("Enter Arrival Time for Process " + (i + 1) + ": ");**

**int ar = sc.nextInt();**

**System.out.print("Enter Burst Time for Process " + (i + 1) + ": ");**

**int br = sc.nextInt();**

**process\_queue[i] = new Process(("p" + (i+1)), ar, br);**

**}**

**while (total\_completed < n) {**

**int min\_burst\_index = n;**

**int min\_burst\_time = Integer.MAX\_VALUE;**

**// Find the process with the minimum remaining burst time that has arrived**

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

**if (process\_queue[i].arrival\_time <= current\_time &&**

**!process\_queue[i].isCompleted &&**

**process\_queue[i].remaining\_time < min\_burst\_time) {**

**min\_burst\_time = process\_queue[i].remaining\_time;**

**min\_burst\_index = i;**

**}**

**}**

**if (min\_burst\_index == n) {**

**current\_time++;**

**} else {**

**process\_queue[min\_burst\_index].remaining\_time--;**

**current\_time++;**

**if (process\_queue[min\_burst\_index].remaining\_time == 0) {**

**process\_queue[min\_burst\_index].completion\_time = current\_time;**

**process\_queue[min\_burst\_index].turn\_around\_time =**

**process\_queue[min\_burst\_index].completion\_time -**

**process\_queue[min\_burst\_index].arrival\_time;**

**process\_queue[min\_burst\_index].waiting\_time =**

**process\_queue[min\_burst\_index].turn\_around\_time -**

**process\_queue[min\_burst\_index].burst\_time;**

**process\_queue[min\_burst\_index].isCompleted = true;**

**total\_completed++;**

**avgwt += process\_queue[min\_burst\_index].waiting\_time;**

**avgtat += process\_queue[min\_burst\_index].turn\_around\_time;**

**}**

**}**

**}**

**System.out.println("--------------------------------------------------------------------------");**

**System.out.println("Process\t\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time");**

**System.out.println("--------------------------------------------------------------------------");**

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

**System.out.println(process\_queue[i].id + "\t\t" +**

**process\_queue[i].arrival\_time + "\t\t" +**

**process\_queue[i].burst\_time + "\t\t" +**

**process\_queue[i].completion\_time + "\t\t" +**

**process\_queue[i].turn\_around\_time + "\t\t" +**

**process\_queue[i].waiting\_time);**

**}**

**System.out.println("\nAverage Turn Around Time: " + (avgtat / n));**

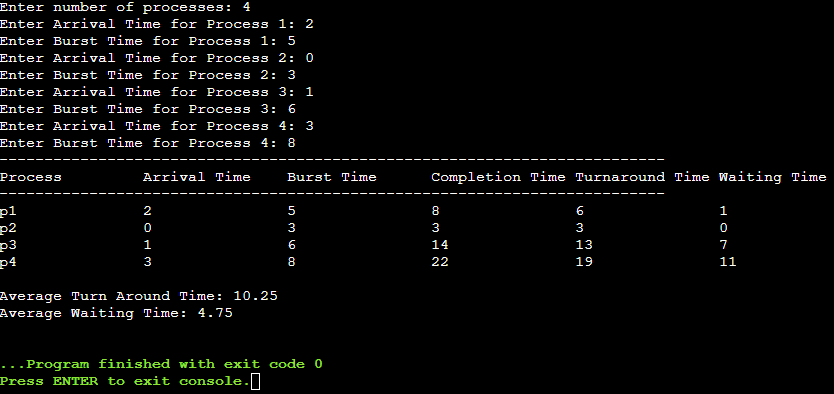
**System.out.println("Average Waiting Time: " + (avgwt / n));**

**sc.close();**

**}**

**}**

**Output:**

****

1. **Round Robin:**

**import java.util.Scanner;**

**class Process {**

**String id;**

**int arrival\_time;**

**int burst\_time;**

**int remaining\_bt;**

**int completion\_time;**

**int turnaround\_time;**

**int waiting\_time;**

**boolean is\_completed;**

**Process(String pid, int at, int bt) {**

**id = pid;**

**arrival\_time = at;**

**burst\_time = bt;**

**remaining\_bt = bt; // Remaining burst time is initially equal to burst time**

**is\_completed = false;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.print("Enter the number of processes (maximum 10): ");**

**int n = sc.nextInt();**

**Process[] process\_queue = new Process[n];**

**System.out.println("Enter the Arrival Time and Burst Time for each process:");**

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

**System.out.print("P" + (i + 1) + " (Arrival Time): ");**

**int at = sc.nextInt();**

**System.out.print("P" + (i + 1) + " (Burst Time): ");**

**int bt = sc.nextInt();**

**process\_queue[i] = new Process("P" + (i + 1), at, bt);**

**}**

**System.out.print("Enter the quantum time: ");**

**int quantum\_time = sc.nextInt();**

**// Sort processes by arrival time using Bubble Sort**

**for (int i = 0; i < n - 1; i++) {**

**for (int j = 0; j < n - i - 1; j++) {**

**if (process\_queue[j].arrival\_time > process\_queue[j + 1].arrival\_time) {**

**Process temp = process\_queue[j];**

**process\_queue[j] = process\_queue[j + 1];**

**process\_queue[j + 1] = temp;**

**}**

**}**

**}**

**// Initialize variables**

**int current\_time = 0; // Tracks the current time**

**int completed = 0; // Number of completed processes**

**int total\_tat = 0, total\_wt = 0; // Total Turnaround Time and Waiting Time**

**// Process execution using Round Robin**

**while (completed < n) {**

**boolean process\_executed = false;**

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

**Process p = process\_queue[i];**

**// Process can execute only if it's arrived and not yet completed**

**if (p.arrival\_time <= current\_time && !p.is\_completed) {**

**process\_executed = true;**

**// If remaining burst time is more than quantum time, execute for quantum time**

**if (p.remaining\_bt > quantum\_time) {**

**current\_time += quantum\_time;**

**p.remaining\_bt -= quantum\_time;**

**} else {**

**// Process completes in this round**

**current\_time += p.remaining\_bt;**

**p.remaining\_bt = 0;**

**p.completion\_time = current\_time;**

**// Calculate turnaround time and waiting time**

**p.turnaround\_time = p.completion\_time - p.arrival\_time;**

**p.waiting\_time = p.turnaround\_time - p.burst\_time;**

**total\_tat += p.turnaround\_time;**

**total\_wt += p.waiting\_time;**

**p.is\_completed = true; // Mark process as completed**

**completed++;**

**}**

**}**

**}**

**// If no process was executed, advance time to the next arriving process**

**if (!process\_executed) {**

**current\_time++;**

**}**

**}**

**// Display process information**

**System.out.println("------------------------------------------------------------------------------------");**

**System.out.println("Process\t\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time");**

**System.out.println("------------------------------------------------------------------------------------");**

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

**Process p = process\_queue[i];**

**System.out.println(p.id + "\t\t" + p.arrival\_time + "\t\t" + p.burst\_time + "\t\t" +**

**p.completion\_time + "\t\t" + p.turnaround\_time + "\t\t" + p.waiting\_time);**

**}**

**// Calculate and display average turnaround time and waiting time**

**System.out.println("\nAverage Turnaround Time = " + (float) total\_tat / n);**

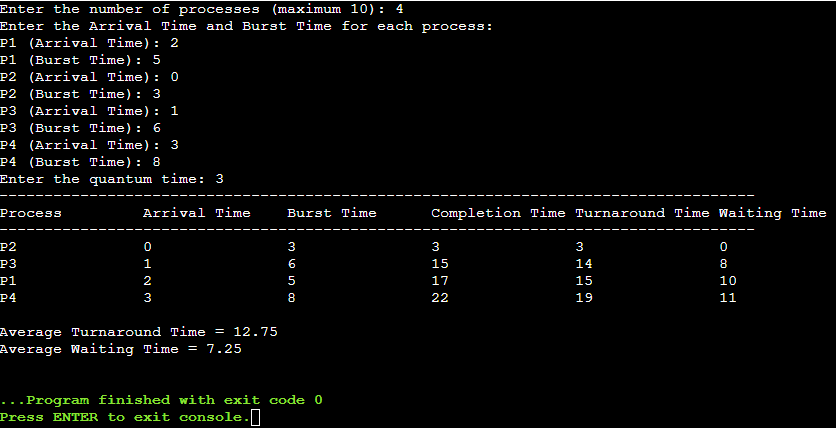
**System.out.println("Average Waiting Time = " + (float) total\_wt / n);**

**sc.close();**

**}**

**}**

**Output:**

****

1. **Priority (non-preemptive):**

**import java.util.\*;**

**class Process {**

**String id;**

**int arrival\_time;**

**int burst\_time;**

**int priority;**

**int completion\_time;**

**int turn\_around\_time;**

**int waiting\_time;**

**boolean isCompleted = false;**

**Process(String pid, int ar, int br, int pr) {**

**id = pid;**

**arrival\_time = ar;**

**burst\_time = br;**

**priority = pr;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.print("Enter Number of Processes: ");**

**int n = sc.nextInt();**

**Process[] process\_queue = new Process[n];**

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

**System.out.print("Enter Arrival Time for Process " + (i + 1) + " : ");**

**int ar = sc.nextInt();**

**System.out.print("Enter Burst Time for Process " + (i + 1) + " : ");**

**int br = sc.nextInt();**

**System.out.print("Enter Priority for Process " + (i + 1) + " : ");**

**int pr = sc.nextInt();**

**process\_queue[i] = new Process(("p" + (i+1)), ar, br, pr);**

**}**

**// Sort processes based on arrival time**

**Arrays.sort(process\_queue, Comparator.comparingInt(p -> p.arrival\_time));**

**int currentTime = 0;**

**int completedProcesses = 0;**

**float total\_tat = 0, total\_wt = 0;**

**// Continue until all processes are completed**

**while (completedProcesses < n) {**

**// Find process with highest priority from arrived processes**

**Process currentProcess = null;**

**int highestPriority = Integer.MAX\_VALUE;**

**for (Process p : process\_queue) {**

**if (!p.isCompleted && p.arrival\_time <= currentTime && p.priority < highestPriority) {**

**highestPriority = p.priority;**

**currentProcess = p;**

**}**

**}**

**if (currentProcess != null) {**

**// Process found, execute it**

**currentProcess.completion\_time = currentTime + currentProcess.burst\_time;**

**currentProcess.turn\_around\_time = currentProcess.completion\_time - currentProcess.arrival\_time;**

**currentProcess.waiting\_time = currentProcess.turn\_around\_time - currentProcess.burst\_time;**

**total\_tat += currentProcess.turn\_around\_time;**

**total\_wt += currentProcess.waiting\_time;**

**currentProcess.isCompleted = true;**

**completedProcesses++;**

**currentTime = currentProcess.completion\_time;**

**} else {**

**// If no process is ready, increment time**

**currentTime++;**

**}**

**}**

**// Print process details**

**System.out.println("------------------------------------------------------------------------------------------------------");**

**System.out.println("Process\t\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time");**

**System.out.println("------------------------------------------------------------------------------------------------------");**

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

**System.out.println(process\_queue[i].id + "\t\t" + process\_queue[i].arrival\_time + "\t\t\t" +**

**process\_queue[i].burst\_time + "\t\t\t" + process\_queue[i].completion\_time + "\t\t\t" +**

**process\_queue[i].turn\_around\_time + "\t\t\t" + process\_queue[i].waiting\_time);**

**}**

**System.out.println("\nAverage Waiting Time = " + (total\_wt / n));**

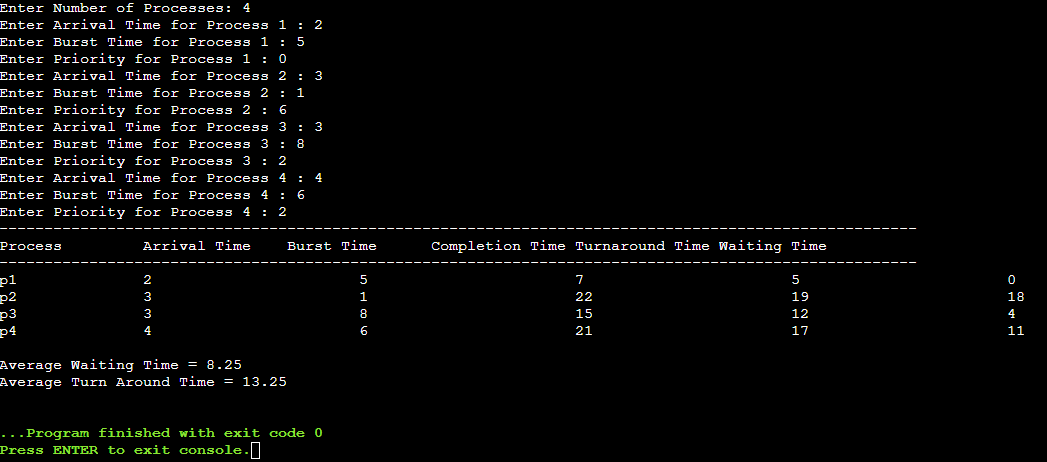
**System.out.println("Average Turn Around Time = " + (total\_tat / n));**

**sc.close();**

**}**

**}**

**Output:**

****