**Assignment 4.2**

**Name** : Komal Potdar

**Roll No**.: 92

**PRN No**.: 12320165

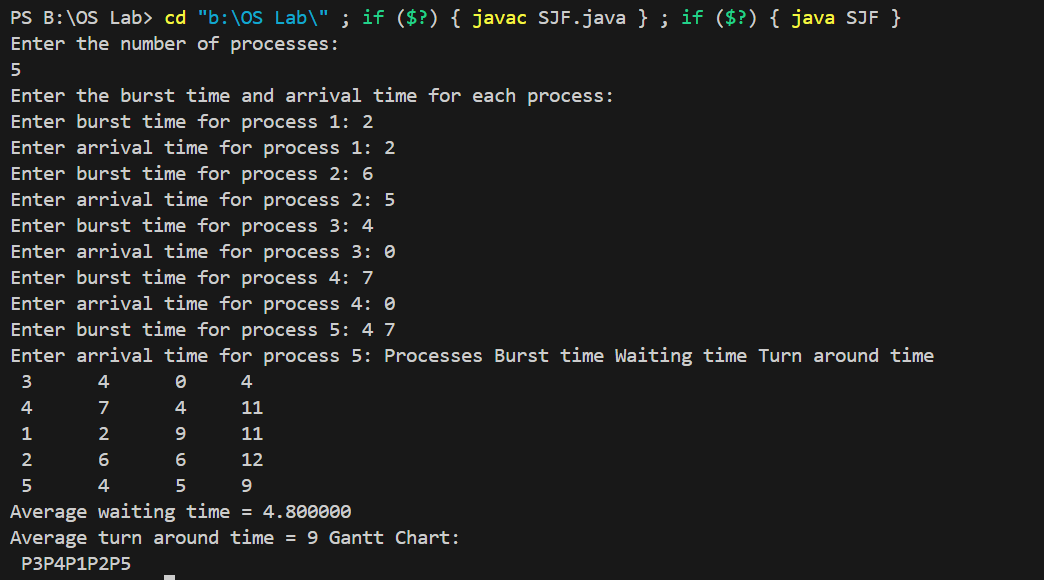
**Div**: CS B SY

**Batch**: 3

**SJF**

|  |
| --- |
| import java.util.Scanner;  class Main {  // Function to find the waiting time for all  // processes  static void findWaitingTime(int processes[], int n,  int bt[], int wt[], int at[]) {  // waiting time for first process is 0  wt[0] = 0;  // calculating waiting time  for (int i = 1; i < n; i++) {  // Waiting time for a process is the sum of the burst times of all  // processes which arrived earlier and have not been executed yet  wt[i] = bt[i - 1] + wt[i - 1] - at[i];    // If the waiting time calculated is negative, it means that the process  // arrived later than the previous process completed, so set it to 0  if (wt[i] < 0) {      wt[i] = 0;  }  }  }  // Function to calculate turn around time  static void findTurnAroundTime(int processes[], int n,  int bt[], int wt[], int tat[]) {  // calculating turnaround time by adding  // bt[i] + wt[i]  for (int i = 0; i < n; i++) {  tat[i] = bt[i] + wt[i];  }  }  // Function to create Gantt chart  static void createGanttChart(int processes[], int n,  int bt[], int wt[], int at[]) {  int currentTime = 0;  System.out.println("Gantt Chart:");  System.out.print(" ");  for (int i = 0; i < n; i++) {  // Print dashes for idle time  while (currentTime < at[i]) {  System.out.print("--");  currentTime++;  }  // Print process ID  System.out.print("P" + processes[i]);  // Update current time  currentTime += bt[i];  }  System.out.println();  }  // Function to calculate average time  static void findavgTime(int processes[], int n, int bt[], int at[]) {  int wt[] = new int[n], tat[] = new int[n];  int total\_wt = 0, total\_tat = 0;  // Sort processes based on arrival time  for (int i = 0; i < n - 1; i++) {  for (int j = 0; j < n - i - 1; j++) {  if (at[j] > at[j + 1]) {  int temp = at[j];  at[j] = at[j + 1];  at[j + 1] = temp;  temp = bt[j];  bt[j] = bt[j + 1];  bt[j + 1] = temp;  temp = processes[j];  processes[j] = processes[j + 1];  processes[j + 1] = temp;  }  }  }  // Function to find waiting time of all processes  findWaitingTime(processes, n, bt, wt, at);  // Function to find turn around time for all processes  findTurnAroundTime(processes, n, bt, wt, tat);  // Display processes along with all details  System.out.printf("Processes Burst time Waiting"  +" time Turn around time\n");  // Calculate total waiting time and total turn  // around time  for (int i = 0; i < n; i++) {  total\_wt = total\_wt + wt[i];  total\_tat = total\_tat + tat[i];  System.out.printf(" %d ", processes[i]);  System.out.printf("     %d ", bt[i]);  System.out.printf("     %d", wt[i]);  System.out.printf("     %d\n", tat[i]);  }  float s = (float)total\_wt /(float) n;  int t = total\_tat / n;  System.out.printf("Average waiting time = %f", s);  System.out.printf("\n");  System.out.printf("Average turn around time = %d ", t);  // Create Gantt chart  createGanttChart(processes, n, bt, wt, at);  }  // Driver code  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);    System.out.println("Enter the number of processes:");  int n = scanner.nextInt();  int processes[] = new int[n];  int burst\_time[] = new int[n];  int arrival\_time[] = new int[n];  System.out.println("Enter the burst time and arrival time for each process:");  for (int i = 0; i < n; i++) {  System.out.printf("Enter burst time for process %d: ", i + 1);  burst\_time[i] = scanner.nextInt();  System.out.printf("Enter arrival time for process %d: ", i + 1);  arrival\_time[i] = scanner.nextInt();  processes[i] = i + 1;  }  findavgTime(processes, n, burst\_time, arrival\_time);    scanner.close();  }  } |

**Output:**



**SRTN**

|  |
| --- |
| import java.util.Scanner;  class Main {      // Function to find the waiting time for all      // processes using Shortest Job First (SJF)      static void findWaitingTime(int processes[], int n,              int bt[], int wt[]) {          int rt[] = new int[n];          // Copy the burst time into rt[]          for (int i = 0; i < n; i++) {              rt[i] = bt[i];          }          int complete = 0, t = 0, minm = Integer.MAX\_VALUE;          int shortest = 0, finish\_time;          boolean check = false;          // Process until all processes are completed          while (complete != n) {              // Find the process with the minimum remaining time              for (int j = 0; j < n; j++) {                  if ((rt[j] <= t) && (rt[j] < minm) && rt[j] > 0) {                      minm = rt[j];                      shortest = j;                      check = true;                  }              }              if (!check) {                  t++;                  continue;              }              // Reduce remaining time by 1 for the chosen process              rt[shortest]--;              minm = rt[shortest];              if (minm == 0)                  minm = Integer.MAX\_VALUE;              // If a process is completed              if (rt[shortest] == 0) {                  complete++;                  check = false;                  // Calculate finish time for this process                  finish\_time = t + 1;                  // Calculate waiting time                  wt[shortest] = finish\_time - bt[shortest];                  if (wt[shortest] < 0)                      wt[shortest] = 0;              }              // Increment time              t++;          }      }      // Function to calculate turn around time      static void findTurnAroundTime(int processes[], int n,              int bt[], int wt[], int tat[]) {          // calculating turnaround time by adding          // bt[i] + wt[i]          for (int i = 0; i < n; i++) {              tat[i] = bt[i] + wt[i];          }      }      // Function to create Gantt chart      static void createGanttChart(int processes[], int n,              int bt[], int wt[], int at[]) {          int currentTime = 0;          System.out.println("Gantt Chart:");          System.out.print(" ");          int[] rt = new int[n];          for (int i = 0; i < n; i++) {              rt[i] = bt[i];          }          int complete = 0;          boolean check = false;          int index = 0;          while (complete != n) {              for (int j = 0; j < n; j++) {                  if ((rt[j] > 0) && (at[j] <= currentTime)) {                      if (!check) {                          System.out.print("P" + processes[j]);                          index = j;                          check = true;                      } else {                          if (processes[j] != processes[index]) {                              System.out.print(" P" + processes[j]);                              index = j;                          }                      }                      rt[j]--;                      currentTime++;                      if (rt[j] == 0) {                          complete++;                          check = false;                      }                  }              }              if (!check) {                  System.out.print("--");                  currentTime++;              }          }          System.out.println();      }      // Function to calculate average time      static void findavgTime(int processes[], int n, int bt[], int at[]) {          int wt[] = new int[n], tat[] = new int[n];          int total\_wt = 0, total\_tat = 0;          // Sort processes based on arrival time          for (int i = 0; i < n - 1; i++) {              for (int j = 0; j < n - i - 1; j++) {                  if (at[j] > at[j + 1]) {                      int temp = at[j];                      at[j] = at[j + 1];                      at[j + 1] = temp;                      temp = bt[j];                      bt[j] = bt[j + 1];                      bt[j + 1] = temp;                      temp = processes[j];                      processes[j] = processes[j + 1];                      processes[j + 1] = temp;                  }              }          }          // Function to find waiting time of all processes          findWaitingTime(processes, n, bt, wt);          // Function to find turn around time for all processes          findTurnAroundTime(processes, n, bt, wt, tat);          // Display processes along with all details          System.out.printf("Processes Burst time Waiting"                  + " time Turn around time\n");          // Calculate total waiting time and total turn          // around time          for (int i = 0; i < n; i++) {              total\_wt = total\_wt + wt[i];              total\_tat = total\_tat + tat[i];              System.out.printf(" %d ", processes[i]);              System.out.printf("     %d ", bt[i]);              System.out.printf("     %d", wt[i]);              System.out.printf("     %d\n", tat[i]);          }          float s = (float) total\_wt / (float) n;          int t = total\_tat / n;          System.out.printf("Average waiting time = %f", s);          System.out.printf("\n");          System.out.printf("Average turn around time = %d ", t);          // Create Gantt chart          createGanttChart(processes, n, bt, wt, at);      }      // Driver code      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          System.out.println("Enter the number of processes:");          int n = scanner.nextInt();          int processes[] = new int[n];          int burst\_time[] = new int[n];          int arrival\_time[] = new int[n];          System.out.println("Enter the burst time and arrival time for each process:");          for (int i = 0; i < n; i++) {              System.out.printf("Enter burst time for process %d: ", i + 1);              burst\_time[i] = scanner.nextInt();              System.out.printf("Enter arrival time for process %d: ", i + 1);              arrival\_time[i] = scanner.nextInt();              processes[i] = i + 1;          }          findavgTime(processes, n, burst\_time, arrival\_time);          scanner.close();      }  } |

**Output:**

