Shortest Job First Non_Preemptive Scheduling- 60004190057

CODE:

```
import java.util.*;
import java.util.Scanner;
import java.io.*;
import java.time.*;

public class sjfNonPreemptive {

   static int[] at= new int[50] ;
   static int[] bt= new int[50] ;
   static int[] tat= new int[50] ;
   static int[] wt= new int[50] ;
   static int[] process_order = new int[50] ;
   static boolean[] process_status= new boolean[50] ;  // false => executed

   static int[] processID = new int[50] ;
   static int num , total_tat=0 , total_wt=0 ;
   static double avg_tat=0.00 , avg_wt=0.00;


    public static void sortArrayByAT(int processID[] ,int at[] , int bt[]){

        for (int i = 1; i <num;  ++i) {
            int at_key = at[i];
            int bt_key = bt[i];
            int process_key = processID[i];

            int hole = i - 1;
            while (hole >= 0 && at[hole] > at_key) {
                at[hole + 1] = at[hole];
                bt[hole + 1] = bt[hole];
                processID[hole + 1] = processID[hole];
                hole = hole - 1;
            }
            at[hole + 1] = at_key;
            bt[hole + 1] = bt_key;
            processID[hole + 1] =process_key ;

        }

    }

    public static void sortArrayByProcessID(int processID[] ,int at[] , int
bt[] , int tat[] , int wt[]){
```

```
        for (int i = 1; i <num;  ++i) {
            int at_key = at[i];
            int bt_key = bt[i];
            int tat_key = tat[i];
            int wt_key = wt[i];
            int process_key = processID[i];

            int hole = i - 1;
            while (hole >= 0 && processID[process_order[hole]] >
process_key) {
                at[hole + 1] = at[hole];
                bt[hole + 1] = bt[hole];
                wt[hole + 1] = wt[hole];
                tat[hole + 1] = tat[hole];
                processID[hole + 1] = processID[hole];
                hole = hole - 1;
            }
            at[hole + 1] = at_key;
            bt[hole + 1] = bt_key;
            wt[hole + 1] = wt_key;
            tat[hole + 1] = tat_key;
            processID[hole + 1] =process_key ;

        }

    }

    public static void printInputTable(int processID[] ,int at[] , int
bt[]){

        System.out.println("\n \n*** The  Input Table  ***\n ") ;
        System.out.println("\nPID \t AT\t BT") ;

        for(int j=0 ; j<num ; j++){
            System.out.println(processID[j]+"\t  "+at[j]+"\t  "+ bt[j] ) ;
        }
        System.out.println() ;
    }

    public static void printOutputTable(int processID[] ,int at[] , int bt[]
, int tat[], int wt[], int  process_order[]){

        System.out.println("\n \n*** The Final Output Table *** \n ") ;
        System.out.println("\nPID \t AT\t BT\t TAT\t  WT") ;

        for(int j=0 ; j<num ; j++){
            System.out.println(processID[process_order[j]]+"\t
```

```
"+at[process_order[j]]+"\t  "+ bt[process_order[j]]+"\t  "+
tat[process_order[j]]+"\t   "+ wt[process_order[j]] ) ;
        }
        System.out.println() ;

        System.out.println("Average TurnAround Time : "+ avg_tat) ;
        System.out.println("Average Waiting Time : " +avg_wt) ;

    }

    public static void gantChart(int processID[] ,int switch_time[], int
process_order[]){
        int j=0 ;
        System.out.println("** GANTT CHART **\n") ;
        if(at[0] != 0){
          System.out.print(switch_time[j++]+"\t[NA]\t") ;
        }

        for(int i=0 ; i<num ; i++){
        // System.out.print(switch_time[j++]+"\t"+ processID[i] +" | ") ;
        System.out.print(switch_time[j++]+"\t["+ processID[process_order[i]]
+"]\t") ;
        }
        System.out.print(switch_time[j]) ;

    }
    public static void CalcTAT_and_WT(int switch_time[] ,int at[] , int
bt[], int  process_order[]){

        int j=1 ;

        if(at[0] != 0){
            j++ ;
        }
        for(int i=0 ; i<num ; i++){
                tat[process_order[i]] = switch_time[j++] -
at[process_order[i]] ;
                wt[process_order[i]] =tat[process_order[i]] -
bt[process_order[i]] ;
                total_tat += tat[process_order[i]];
                total_wt +=  wt[process_order[i]];
        }

        avg_tat = (double)total_tat/num ;
        avg_wt = (double)total_wt/num ;
    }

    public static void SJFNP(int  processID[],int at[] , int bt[]){
```

```
        int total_time=0 , j=0;
        int switch_time[] = new int[50] ;
        switch_time[j++] = 0;

        if(at[0] != 0){
            total_time = total_time + at[0] ;
            switch_time[j++] =  total_time ;
        }

        int min_burst ;
        int index =0 ;

        for(int i=0 ; i<num ; i++){
            min_burst =999 ;
            for(int k=0 ; k<num ; k++){
              if((process_status[k] == true) && (at[k] <= total_time) &&
(bt[k]<min_burst)){
                min_burst= bt[k];
                index = k;
             }

           }

            total_time =total_time + min_burst;
            switch_time[j++] =  total_time ;
            process_status[index] = false ;
            process_order[i] =index ;
            // System.out.print(processID[i]+"\t" + index +"\n") ;

        }
        System.out.println() ;

        gantChart(processID  ,switch_time , process_order) ;
        CalcTAT_and_WT(switch_time ,at , bt , process_order) ;
      // sortArrayByProcessID(processID, at, bt, tat, wt);
        printOutputTable(processID, at, bt ,tat , wt , process_order) ;
    }



    public static void main(String args[]){
        Scanner sc = new Scanner(System.in) ;

        System.out.println("Enter the number of processes : ") ;
        num =sc.nextInt() ;

        System.out.println("Enter the Arrival Time & Burst Time of the
Processes :") ;
```

```
        for(int i=0 ; i<num ; i++){
           // System.out.println("Arrival Time & Burst Time of "+ (i+1)+" :
") ;
            processID[i] =i+1 ;
            at[i] =sc.nextInt() ;
            bt[i] =sc.nextInt() ;
            process_status[i]= true;

         }


        //Sorting of the Processes wrt AT
        sortArrayByAT(processID,at,bt) ;

        //Print Array
        printInputTable(processID,at,bt) ;

        //SJF NP
        SJFNP(processID,at,bt) ;

    }
}
```

OUTPUT:

```
Enter the number of processes :
5
Enter the Arrival Time & Burst Time of the Processes :
1 7
2 5
3 1
4 2
5 8


*** The  Input Table  ***


PID      AT      BT
1         1       7
2         2       5
3         3       1
4         4       2
5         5       8


** GANTT CHART **

0       [NA]    1       [1]     8       [3]     9       [4]     11      [2]     16      [5]     24

*** The Final Output Table ***


PID      AT      BT      TAT      WT
1         1       7       7        0
3         3       1       6        5
4         4       2       7        5
2         2       5       14       9
5         5       8       19       11

Average TurnAround Time : 10.6
Average Waiting Time : 6.0
```

Shortest Job First Preemptive Scheduling- 60004190057

CODE:

```java
import java.util.*;

class P {
    int id;
    int burstTime;
    int arrivalTime;

    public P(int id, int arrivalTime, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
        this.arrivalTime = arrivalTime;
    }
```

```
}

class P_SRT {

    static void findWaitingTime(P process_arr[], int waitingTime[], int
finish_time[], ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int remainingTime[] = new int[length];

        for (int i = 0; i < length; i++)
            remainingTime[i] = process_arr[i].burstTime;

        int complete = 0;
        int time = 0;
        int minimum = Integer.MAX_VALUE;
        int shortest = 0;
        boolean check = false;

        while (complete != length) {

            for (int j = 0; j < length; j++) {
                if ((process_arr[j].arrivalTime <= time) &&
(remainingTime[j] < minimum) && remainingTime[j] > 0) {
                    minimum = remainingTime[j];
                    shortest = j;
                    check = true;
                }
            }

            if (check == false) {
                Grant.add(0);
                time++;
                continue;
            }

            Grant.add(process_arr[shortest].id);
            remainingTime[shortest]--;

            minimum = remainingTime[shortest];
            if (minimum == 0)
                minimum = Integer.MAX_VALUE;

            if (remainingTime[shortest] == 0) {

                complete++;
                check = false;
                finish_time[shortest] = time + 1;
```

```
                // waiting time
                waitingTime[shortest] = finish_time[shortest] -
process_arr[shortest].burstTime
                        - process_arr[shortest].arrivalTime;
            }
            time++;
        }
    }

    static void findTurnAroundTime(P process_arr[], int waitingTime[], int
turnAroundTime[]) {
        int length = process_arr.length;
        for (int i = 0; i < length; i++)
            turnAroundTime[i] = process_arr[i].burstTime + waitingTime[i];
    }

    static void findavgTime(P process_arr[], int waitingTime[], int
turnAroundTime[], int finishTime[],
            ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int total_waitTime = 0;
        int total_tATime = 0;
        System.out.println("Processes " + " Arrival Time " + " Burst time "
+ "Finish Time" + "  Waiting time "
                + "  Turn around time");

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < length; i++) {
            total_waitTime = total_waitTime + waitingTime[i];
            total_tATime = total_tATime + turnAroundTime[i];
            System.out.println(" " + process_arr[i].id + "\t\t" + " " +
process_arr[i].arrivalTime + "\t\t"
                    + +process_arr[i].burstTime + "\t" + finishTime[i] +
"\t\t" + waitingTime[i] + "\t\t"
                    + turnAroundTime[i]);
        }

        System.out.println("Average waiting time = " + (double)
total_waitTime / (double) length);
        System.out.println("Average turn around time = " + (double)
total_tATime / (double) length);
        System.out.println("Grant Chart: " + Grant);

    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of processes");
        int n = sc.nextInt();
        sc.nextLine();
        ArrayList<Integer> Grant = new ArrayList<Integer>();
        int arrivalTime = 0, bursttime = 0;
        int waitingTime[] = new int[n];
        int turnAroundTime[] = new int[n];
        int finishTime[] = new int[n];
        P process_arr[] = new P[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter arrival time and burst time ");
            arrivalTime = sc.nextInt();
            bursttime = sc.nextInt();
            process_arr[i] = new P(i + 1, arrivalTime, bursttime);
        }
        sc.close();

        findWaitingTime(process_arr, waitingTime, finishTime, Grant);
        findTurnAroundTime(process_arr, waitingTime, turnAroundTime);
        findavgTime(process_arr, waitingTime, turnAroundTime, finishTime,
Grant);
    }
}
```

OUTPUT:

```
Enter number of processes
5
Enter arrival time and burst time
1 7
Enter arrival time and burst time
2 5
Enter arrival time and burst time
3 1
Enter arrival time and burst time
4 2
Enter arrival time and burst time
5 8
Processes  Arrival Time  Burst time Finish Time  Waiting time   Turn around time
 1             1            7      16            8             15
 2             2            5      10            3             8
 3             3            1      4             0             1
 4             4            2      6             0             2
 5             5            8      24            11            19
Average waiting time = 4.4
Average turn around time = 9.0
Grant Chart: [0, 1, 2, 3, 4, 4, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 5, 5]
```

Preemptive Priority Scheduling- 60004190057

CODE:

```
import java.util.*;

class Prow {

    int id;
    int burstTime;
    int arrivalTime;
    int priority;

    public Prow(int id, int arrivalTime, int burstTime, int priority) {
        this.id = id;
        this.burstTime = burstTime;
        this.arrivalTime = arrivalTime;
        this.priority = priority;
    }
}

public class Preemptive_Priority {
    public static void main(String[] args) {
        int n;
        System.out.println("Enter Number of Processes:");
        Scanner ob = new Scanner(System.in);
        n = ob.nextInt();
        ArrayList<Integer> Grant = new ArrayList<Integer>();
        int arrivalTime = 0, bursttime = 0, priority;
        int waitingTime[] = new int[n];
        int turnAroundTime[] = new int[n];
        int finishTime[] = new int[n];
        Prow process_arr[] = new Prow[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter Arrival Time, Burst Time and Priority
for the Process:");
            arrivalTime = ob.nextInt();
            bursttime = ob.nextInt();
            priority = ob.nextInt();
            process_arr[i] = new Prow(i + 1, arrivalTime, bursttime,
priority);
        }
        findWaitingTime(process_arr, waitingTime, finishTime, Grant);
        findTurnAroundTime(process_arr, waitingTime, turnAroundTime);
        findavgTime(process_arr, waitingTime, turnAroundTime, finishTime,
Grant);

        ob.close();
```

```
    }


    static void findWaitingTime(Prow process_arr[], int waitingTime[], int
finish_time[], ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int min = 0;
        int priorityList[] = new int[length];
        int remainingTime[] = new int[length];

        for (int i = 0; i < length; i++)
            priorityList[i] = process_arr[i].priority;

        for (int i = 0; i < length; i++)
            remainingTime[i] = process_arr[i].burstTime;

        int complete = 0;
        int time = 0;
        int minimum = Integer.MAX_VALUE;
        int shortest = 0;
        boolean check = false;

        while (complete != length) {

            for (int j = 0; j < length; j++) {

                if ((process_arr[j].arrivalTime <= time) && (priorityList[j]
< minimum && remainingTime[j] != 0) && (priorityList[j] > 0)) {
                    minimum = priorityList[j];
                    shortest = j;
                    check = true;
                }
            }

            if (check == false) {
                Grant.add(0);
                time++;
                continue;
            }

            Grant.add(process_arr[shortest].id);
            remainingTime[shortest]--;

            //System.out.println("Remaining Time : " +
remainingTime[shortest]);
```

```
            minimum = priorityList[shortest];

            min = remainingTime[shortest];
            if (min <= 0)
                minimum = Integer.MAX_VALUE;

            if (remainingTime[shortest] <= 0) {

                complete++;
                check = false;
                finish_time[shortest] = time + 1;

                // waiting time
                waitingTime[shortest] = finish_time[shortest] -
process_arr[shortest].burstTime
                        - process_arr[shortest].arrivalTime;
            }
            time++;
        }
    }

    static void findTurnAroundTime(Prow process_arr[], int waitingTime[],
int turnAroundTime[]) {
        int length = process_arr.length;
        for (int i = 0; i < length; i++)
            turnAroundTime[i] = process_arr[i].burstTime + waitingTime[i];
    }

    static void findavgTime(Prow process_arr[], int waitingTime[], int
turnAroundTime[], int finishTime[],
            ArrayList<Integer> Grant) {

        int length = process_arr.length;
        int total_waitTime = 0;
        int total_tATime = 0;
        System.out.println("Processes " + " Arrival Time " + " Burst time "
+  "Priority" + " Finish Time" + "  Waiting time "
                + "  Turn around time");

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < length; i++) {
            total_waitTime = total_waitTime + waitingTime[i];
            total_tATime = total_tATime + turnAroundTime[i];
            System.out.println(" " + process_arr[i].id + "\t\t" + " " +
process_arr[i].arrivalTime + "\t\t"
                    + +process_arr[i].burstTime + "\t" +
process_arr[i].priority + "\t" + finishTime[i] + "\t\t" + waitingTime[i] +
```

```
"\t\t"
                    + turnAroundTime[i]);
        }

        System.out.println("Average waiting time = " + (double)
total_waitTime / (double) length);
        System.out.println("Average turn around time = " + (double)
total_tATime / (double) length);
        System.out.println("Grant Chart: " + Grant);

    }
}
```

OUTPUT:

```
Enter Number of Processes:
7
Enter Arrival Time, Burst Time and Priority for the Process:
0 4 2
Enter Arrival Time, Burst Time and Priority for the Process:
1 2 4
Enter Arrival Time, Burst Time and Priority for the Process:
2 3 6
Enter Arrival Time, Burst Time and Priority for the Process:
3 5 10
Enter Arrival Time, Burst Time and Priority for the Process:
4 1 8
Enter Arrival Time, Burst Time and Priority for the Process:
5 4 12
Enter Arrival Time, Burst Time and Priority for the Process:
6 6 9
Processes  Arrival Time  Burst time Priority Finish Time  Waiting time   Turn around time
 1              0            4       2       4              0              4
 2              1            2       4       6              3              5
 3              2            3       6       9              4              7
 4              3            5       10      21             13             18
 5              4            1       8       10             5              6
 6              5            4       12      25             16             20
 7              6            6       9       16             4              10
Average waiting time = 6.428571428571429
Average turn around time = 10.0
Grant Chart: [1, 1, 1, 1, 2, 2, 3, 3, 3, 5, 7, 7, 7, 7, 7, 7, 4, 4, 4, 4, 4, 6, 6, 6, 6]
```

Round Robin Scheduling- 60004190057

CODE:

```
import java.util.*;
import java.util.Scanner;

public class RoundRobin {
    static ArrayList<chartItem> ganttChart = new ArrayList<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of Process: ");
        int n =sc.nextInt();
        Process3[] process = new Process3[n];
        System.out.println("ID AT BT");
        for(int i=0;i<n;i++)
        {
            int id = sc.nextInt();
            int AT = sc.nextInt();
            int BT = sc.nextInt();
            process[i] = new Process3(id,AT,BT);
        }
        System.out.println("Enter Quantum Time : ");
        int QT =sc.nextInt();
        sortProcess(process);
        completeProcess(process,QT);
        float wait =0.0f;
        float Ttime =0.0f;
        System.out.println("Id\tAT\tBT\tCT\tWT\tTT ");
        for(int i=0;i<process.length;i++)
        {

System.out.println(process[i].id+"\t"+process[i].AT+"\t"+process[i].BT+"\t"+
process[i].CT+"\t"+process[i].WT+"\t"+process[i].TT);
            wait+=process[i].WT;
            Ttime+=process[i].TT;
        }
        System.out.print("\nProcess: ");
        for(int i=0;i<ganttChart.size();i++)
        {
            System.out.print("| "+ganttChart.get(i).id+"\t");

        }
        System.out.println("|\n");
        System.out.print("Time:-\t");
```

```
        int time=0;
        System.out.print(" "+time+"\t");
        for(int i=0;i<ganttChart.size();i++)
        {
            time+=ganttChart.get(i).time;
            System.out.print(" "+time+"\t");

        }
        System.out.println("\n");
        System.out.println("AVG Waiting time :
"+(wait/process.length)+"\nAVG Turn around time : "+(Ttime/process.length));

    }

    static void completeProcess(Process3[] process,int QT) {
        ArrayList<Process3> readyQueue = new ArrayList<>();
        int time = 0 ;
        int pt =0;
        while(pt<process.length)
        {
            if(readyQueue.isEmpty())
            {
                readyQueue.add(new Process3(process[pt]));
                time = process[pt].AT>=time ?process[pt].AT:time;
            }
            else {
            if(process[pt].AT>time)
            {
                break;
            }
            else
            {
                readyQueue.add(new Process3(process[pt]));
            }
            }
            pt++;
        }

        while(!readyQueue.isEmpty()) {
            boolean completed = false;
            Process3 p = readyQueue.remove(0);

            if(p.BT>QT) {
                p.BT-=QT;
                time+=QT;
                chartItem c=new chartItem(p.id,QT);
                ganttChart.add(c);
                completed=false;
```

```
            }
            else {
                time +=p.BT;
                int id = getId(process,p);
                process[id].CT=time;
                process[id].TT=time-process[id].AT;
                process[id].WT=process[id].TT-process[id].BT;
                chartItem c=new chartItem(p.id,p.BT);
                ganttChart.add(c);
                completed=false;
                completed=true;

            }

            while(pt<process.length)
            {
                if(readyQueue.isEmpty())
                {
                    readyQueue.add(new Process3(process[pt]));
                    time = process[pt].AT>=time ?process[pt].AT:time;
                }
                else {
                if(process[pt].AT>time)
                {
                    break;
                }
                else
                {
                    readyQueue.add(new Process3(process[pt]));
                }
                }
                pt++;
            }

            if(!completed) {
                readyQueue.add(p);
            }

        }
    }

    static void sortProcess(Process3[] process)
    {
        for (int i = 0; i < process.length; i++) {
            for (int j = 0; j < process.length - i - 1; j++) {
            if (process[j].AT > process[j+1].AT) {
                    Process3 Temp = process[j];
                    process[j]=process[j+1];
```

```
                    process[j+1]=Temp;
                }
            }
        }
    }

    static int getId(Process3[] ps,Process3 p)
    {
        int id =0;
        for(id=0;id<ps.length;id++)
        {
            if(ps[id].id==p.id)
                break;
        }
        return id;
    }

}

class Process3{
    int id,AT,BT,WT=0,TT=0,CT=0;
    Process3(int id,int AT,int BT)
    {
        this.id=id;
        this.AT=AT;
        this.BT=BT;
    }
    Process3(Process3 p){
        this.id=p.id;
        this.AT=p.AT;
        this.BT=p.BT;
    }
}

class chartItem{
    int id,time;
    chartItem(int id,int time){
        this.id=id;
        this.time = time;
    }
}
```

OUTPUT:

```
Enter number of Process:
6
ID AT BT
1 0 4
2 1 5
3 2 2
4 3 1
5 4 6
6 6 3
Enter Quantum Time :
2
Id      AT      BT      CT      WT      TT
1       0       4       8       4       8
2       1       5       18      12      17
3       2       2       6       2       4
4       3       1       9       5       6
5       4       6       21      11      17
6       6       3       19      10      13

Process: | 1    | 2    | 3    | 1    | 4    | 5    | 2    | 6    | 5    | 2
 6      | 5    |

Time:-   0      2      4      6      8      9      11     13     15     17
18      19     21

AVG Waiting time : 7.3333335
AVG Turn around time : 10.833333
```