LAB ASSIGNMENT – 3

Scheduling Algorithms

1) FCFS Scheduling Algorithm –

Program –

```
package Scheduling.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 SchedulingFCFS {
    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 ID for Process " + (i+1) + " : ");
            pid = sc.next();
            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(pid,ar,br);
        }
        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;
                }
            }
        }
        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;
        }
        System.out.print("-------------------------------------------------------------------
-------------------------------------------------");
        System.out.print("\nProcess\t\tArrival Time\tBurst Time\tCompletion
Time\t\tTurnaround Time\t\tWaiting Time\n");
        System.out.print("-------------------------------------------------------------------
-------------------------------------------------");
        for(int  i = 0 ; i< n; i++)
        {
            System.out.print("\n  " + process_queue[i].id + "\t\t\t" +
process_queue[i].arrival_time + "\t\t\t\t" + process_queue[i].burst_time + "\t\t\t\t" +
process_queue[i].completion_time + " \t\t\t\t" + process_queue[i].turn_around_time +
"\t\t\t\t\t" + process_queue[i].waiting_time + "\n");
        }
        System.out.println("Average Waiting Time = " + (avgwt/n));
        System.out.println("Average Turn Around Time = " + (avgtat/n));
        sc.close();
    }
}
```

Output –

```
Enter Number of Processes : 5
Enter ID for Process 1 : A
Enter Arrival Time for Process 1 : 0
Enter Burst Time for Process 1 : 2
Enter ID for Process 2 : B
Enter Arrival Time for Process 2 : 1
Enter Burst Time for Process 2 : 1
Enter ID for Process 3 : C
Enter Arrival Time for Process 3 : 2
Enter Burst Time for Process 3 : 3
Enter ID for Process 4 : D
Enter Arrival Time for Process 4 : 3
Enter Burst Time for Process 4 : 5
Enter ID for Process 5 : E
Enter Arrival Time for Process 5 : 4
Enter Burst Time for Process 5 : 4

-----------------------------------------------------------------------------------------------
Process     Arrival Time    Burst Time  Completion Time     Turnaround Time     Waiting Time
-----------------------------------------------------------------------------------------------
  A         0               2               2               2                   0

  B         1               1               3               2                   1

  C         2               3               6               4                   1

  D         3               5               11              8                   3

  E         4               4               15              11                  7
Average Waiting Time = 2.4
Average Turn Around Time = 5.4
```

2) SRTF Scheduling Algorithm –

Program –

```
package Scheduling.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 PreemptiveSJF {
    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 ID for Process " + (i + 1) + " : ");
            String pid = sc.next();
            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(pid, 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\t\tTurnaround Time\t\tWaiting Time");
        System.out.println("-----------------------------------------------------------------
-------------------------------------------------------");

        for (int i = 0; i < n; i++) {
            System.out.println(process_queue[i].id + "\t\t\t\t" +
process_queue[i].arrival_time + "\t\t\t\t" + process_queue[i].burst_time + "\t\t\t\t" +
process_queue[i].completion_time + "\t\t\t\t" + process_queue[i].turn_around_time +
"\t\t\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 –

```
Enter number of processes: 5
Enter ID for Process 1 : A
Enter Arrival Time for Process 1 : 0
Enter Burst Time for Process 1 : 2
Enter ID for Process 2 : B
Enter Arrival Time for Process 2 : 1
Enter Burst Time for Process 2 : 1
Enter ID for Process 3 : C
Enter Arrival Time for Process 3 : 2
Enter Burst Time for Process 3 : 3
Enter ID for Process 4 : D
Enter Arrival Time for Process 4 : 3
Enter Burst Time for Process 4 : 5
Enter ID for Process 5 : E
Enter Arrival Time for Process 5 : 4
Enter Burst Time for Process 5 : 4
```

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	1	1	3	2	1
C	2	3	6	4	1
D	3	5	15	12	7
E	4	4	10	6	2

```
Average Turn Around Time: 5.2
Average Waiting Time: 2.2
```

3) Round Robin Scheduling Algorithm –

Program –

```
package Scheduling.RoundRobin;
import java.util.Arrays;
import java.util.Comparator;
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 RoundRobin {
    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
        Arrays.sort(process_queue, Comparator.comparingInt(p -> p.arrival_time));

        // 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\t\tTurnaround Time\t\tWaiting Time");
        System.out.println("----------------------------------------------------------------
-------------------------------------------------------");

        for (int i = 0; i < n; i++) {
            Process p = process_queue[i];
            System.out.print("\n  " + process_queue[i].id + "\t\t\t" +
process_queue[i].arrival_time + "\t\t\t\t" + process_queue[i].burst_time + "\t\t\t\t" +
process_queue[i].completion_time + " \t\t\t\t" + process_queue[i].turnaround_time +
"\t\t\t\t\t" + process_queue[i].waiting_time + "\n");
        }

        // 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 –

```
Enter the number of processes (maximum 10): 5
Enter the Arrival Time and Burst Time for each process:
P1 (Arrival Time): 0
P1 (Burst Time): 2
P2 (Arrival Time): 1
P2 (Burst Time): 1
P3 (Arrival Time): 2
P3 (Burst Time): 3
P4 (Arrival Time): 3
P4 (Burst Time): 5
P5 (Arrival Time): 4
P5 (Burst Time): 4
Enter the quantum time: 3
```

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	2	2	2	0
P2	1	1	3	2	1
P3	2	3	6	4	1
P4	3	5	14	11	6
P5	4	4	15	11	7

```
Average Turnaround Time = 6.0
Average Waiting Time = 3.0
```

4) Priority (Non-Preemptive) Scheduling Algorithm –

Program –

```
package Scheduling.Priority_NP;
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 PriorityNP {
    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 ID for Process " + (i + 1) + " : ");
            String pid = sc.next();
            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(pid, 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.print("-----------------------------------------------------------------
-------------------------------------------------");
        System.out.print("\nProcess\t\tArrival Time\tBurst Time\tCompletion
Time\t\tTurnaround Time\t\tWaiting Time\n");
        System.out.print("-----------------------------------------------------------------
-------------------------------------------------");
        for(int  i = 0 ; i< n; i++)
        {
            System.out.print("\n  " + process_queue[i].id + "\t\t\t" +
process_queue[i].arrival_time + "\t\t\t\t" + process_queue[i].burst_time + "\t\t\t\t" +
process_queue[i].completion_time + " \t\t\t\t" + process_queue[i].turn_around_time +
"\t\t\t\t\t" + process_queue[i].waiting_time + "\n");
        }
        System.out.println("Average Waiting Time = " + (total_wt / n));
        System.out.println("Average Turn Around Time = " + (total_tat / n));

        sc.close();
    }
}
```

Output –

```
Enter Number of Processes: 5
Enter ID for Process 1 : A
Enter Arrival Time for Process 1 : 0
Enter Burst Time for Process 1 : 2
Enter Priority for Process 1 : 10
Enter ID for Process 2 : B
Enter Arrival Time for Process 2 : 1
Enter Burst Time for Process 2 : 1
Enter Priority for Process 2 : 8
Enter ID for Process 3 : C
Enter Arrival Time for Process 3 : 2
Enter Burst Time for Process 3 : 3
Enter Priority for Process 3 : 4
Enter ID for Process 4 : D
Enter Arrival Time for Process 4 : 3
Enter Burst Time for Process 4 : 5
Enter Priority for Process 4 : 6
Enter ID for Process 5 : E
Enter Arrival Time for Process 5 : 4
Enter Burst Time for Process 5 : 4
Enter Priority for Process 5 : 2
```

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	1	1	15	14	13
C	2	3	5	3	0
D	3	5	14	11	6
E	4	4	9	5	1

```
Average Waiting Time = 4.0
Average Turn Around Time = 7.0
```