



**GATE** INSTITUTE OF TECHNOLOGY  
& MANAGEMENT SCIENCES

(Approved by AICTE, Affiliated to Sri Venkateswara University, TIRUPATI)

**ICET CODE : GTMT**



**Master of Computer Applications**

**SEMESTER - I**

Hall Ticket No: \_\_\_\_\_

Name of the Student: P. Janakiram

Subject: Operating System

Year: I Semester: I Batch 2025 - 2026



# **GATE** EDUCATIONAL INSTITUTIONS

DEPARTMENT OF COMPUTERS

YEAR I SEMESTER I

Reg No. \_\_\_\_\_

## **CERTIFICATE**

*Certified that this is the bonafide record of practical work done in the laboratory by the candidate P. Janakiram during the academic year 2025-2027 subject Operating System.*

No. of Practical's Conducted 15

No. of Practical's Attended 15

LECTURER

HEAD OF THE DEPARTMENT

Submitted for the Practical Examination held on \_\_\_\_\_

Examiners

1.

2.



# INDEX

| S.No. | Date     | Name of the Experiment                                                          | Page No. | Marks Awarded | Remarks |
|-------|----------|---------------------------------------------------------------------------------|----------|---------------|---------|
| 01.   | 24/10/25 | Write a Java Program to implement the concept of First in first out Scheduling. |          |               |         |
| 02.   | 27/10/25 | Write a Java program to implement the concept of Shortest job first Scheduling  |          |               |         |
| 03.   | 30/10/25 | Write a Java program to implement the concept of Round robin job Scheduling.    |          |               |         |
| 04.   | 3/11/25  | Write a Java program to implement the concept of Priority Scheduling            |          |               |         |
| 05.   | 6/11/25  | Write a Java program to implement the concept of Bankers algorithm.             |          |               |         |
| 06.   | 10/11/25 | Write a Java program to implement the concept of dining philosophers Problem.   |          |               |         |
| 07.   | 12/11/25 | Write a Java program to implement the concept of Producer, Consumer Problem.    |          |               |         |



# INDEX

| S.No. | Date     | Name of the Experiment                                                     | Page No. | Marks Awarded | Remarks |
|-------|----------|----------------------------------------------------------------------------|----------|---------------|---------|
| 08    | 2/12/25  | Write a Java program to implement best fit algorithm.                      |          |               |         |
| 09    | 4/12/25  | Write a Java program to implement memory fragmentation.                    |          |               |         |
| 10    | 6/12/25  | Write a Java program to implement the concept of FIFO page replacement.    |          |               |         |
| 11    | 8/12/25  | Write a Java program to implement the concept of LRU page replacement.     |          |               |         |
| 12    | 10/12/25 | Write a Java program to implement the concept of optimum page replacement. |          |               |         |
| 13    | 12/12/25 | Write a Java program to implement the concept of SCAN disk scheduling.     |          |               |         |
| 14    | 16/12/25 | Write a Java program to implement the concept of CLOOK disk scheduling.    |          |               |         |
| 15    | 19/12/25 | Write a Java program to implement the concept of FCFS disk scheduling.     |          |               |         |



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>01 |

**AIM:** To write a java program to implement the concept of first in first out job scheduling.

**Description:** The program implements the First Come first Serve (FCFS) CPU scheduling algorithm using Java. In FCFS scheduling, the process that arrives first is executed first. The program accepts the number of processes & their burst time as input. It then calculates.

- Waiting Time (WT)
- Turnaround Time (TAT)
- Average Waiting Time.
- Average Turnaround Time.

Finally, it displays the process details in tabular form & print the Gantt chart for execution order.

## Algorithm:

Step 1: Start.

Step 2: Input Data.

- Step 2.1: Read the number of processes  $p$ .
- Step 2.2: Read the burst time of each process into an array  $bt[]$ .

Step 3: Initialize Arrays and Variables.

- Step 3.1: Initialize waiting time array  $wt[]$  & turnaround time array  $tat[]$ .
- Step 3.2: Set  $wt[0] = 0$  (first process does not wait).
- Step 3.3: Initialize total waiting time  $t1 = 0$  & total turnaround time  $t2 = 0$ .

Step 4: Calculate Waiting Time (WT).

- Step 4.1: for each process  $i$  from 1 to  $p-1$ :  
 $wt[i] = wt[i-1] + bt[i-1]$ .
- Step 4.2: Add  $tat[i]$  to total turnaround time  $t2$ .

Step 5: Calculate Turnaround Time (TAT).

- Step 5.1: for each process  $i$  from 0 to  $p-1$ :  
 $tat[i] = bt[i] + wt[i]$ .
- Step 5.2: Add  $tat[i]$  to total turnaround time  $t2$ .

Step 6: Calculate Average Times.

- Step 6.1: Average waiting time =  $t1 / p$ .
- Step 6.2: Average turnaround time =  $t2 / p$ .

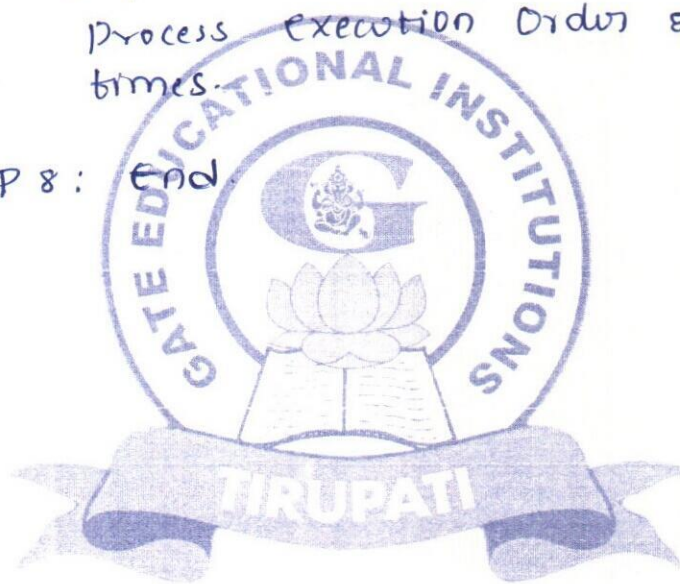


|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>01 |

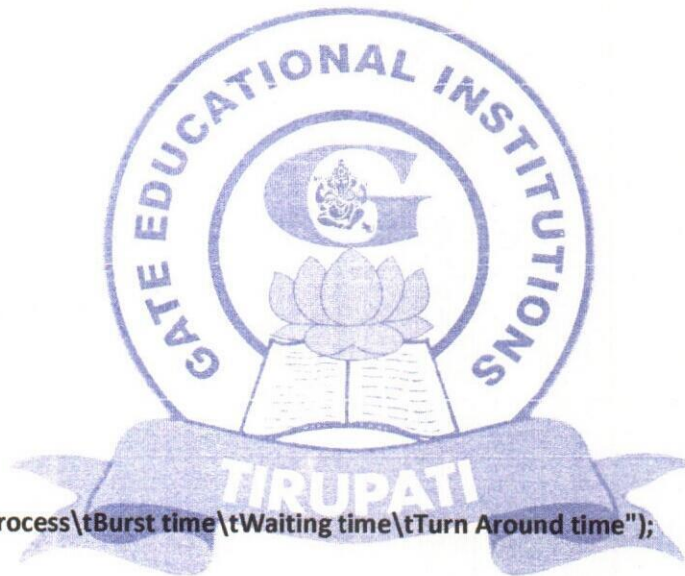
Step 7: Display result.

- Step 7.1 : Display a table with Process number, burst time, waiting time, & turnaround time.
- Step 7.2 : Display average waiting time & average turnaround time.
- Step 7.3 : Display Gantt chart showing Process execution Order & Completion times.

Step 8: End



|                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                 |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Date :                           | <div style="text-align: center;">2</div> <b>GATE Educational Institutions</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Page No. :      |
| import java.util.*;<br>class FCF |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Practical No. : |
|                                  | <div style="text-align: center;">//1. FCFS SCHEDULING</div> <pre> { public static void main(String args[]) { Scanner in=new Scanner(System.in); int p; float t1=0,t2=0; System.out.println("Enter number of processes"); p=in.nextInt(); int bt[]=new int[p];  System.out.println("Enter burst time"); for(int i=0;i&lt;p;i++) { System.out.println("Burst time for P"+(i+1)+"="); bt[i]=in.nextInt(); } int wt[]=new int[p]; wt[0]=0; for(int i=1;i&lt;p;i++) { wt[i]=bt[i-1]+wt[i-1]; t1+=wt[i]; } int tat[]=new int[p]; for(int i=0;i&lt;p;i++) { tat[i]=bt[i]+wt[i]; t2+=tat[i]; } System.out.println("Process\tBurst time\tWaiting time\tTurn Around time"); for(int i=0;i&lt;p;i++) { System.out.println("P"+(i+1)+"\t\t"+bt[i]+" \t\t"+wt[i]+" \t\t"+tat[i]); } float awt,atat; System.out.println("Average Waiting time="+t1/p); System.out.println("Average Turn Around time="+t2/p); System.out.println("_____"); System.out.println("GANT CHART"); System.out.println("_____"); for(int i=1;i&lt;=p;i++) {System.out.print("P"+i); System.out.print("\t-&gt;");} System.out.println("\n_____"); System.out.print("0"); for(int i=0;i&lt;p;i++) { System.out.print("\t"+(wt[i]+bt[i]));} </pre> |                 |





Date :

3

# GATE Educational Institutions

Page No. :

}  
}

Practical No. :

/\*OUTPUT

D:\Java>java FCF

Enter number of processes:3

Enter burst time

Burst time for P1=24

Burst time for P2=3

Burst time for P3=3

| Process | Burst time | Waiting time | Turn Around time |
|---------|------------|--------------|------------------|
|---------|------------|--------------|------------------|

|    |    |   |    |
|----|----|---|----|
| P1 | 24 | 0 | 24 |
|----|----|---|----|

|    |   |    |    |
|----|---|----|----|
| P2 | 3 | 24 | 27 |
|----|---|----|----|

|    |   |    |    |
|----|---|----|----|
| P3 | 3 | 27 | 30 |
|----|---|----|----|

Average Waiting time=17.0

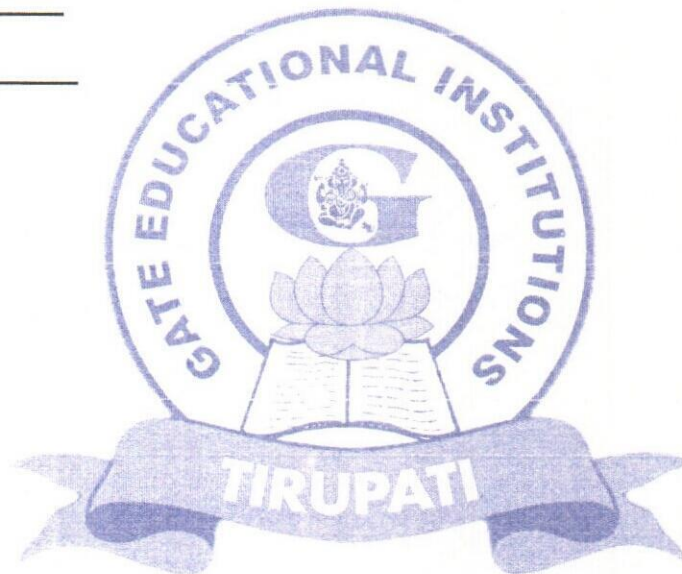
Average Turn Around time=27.0

GANTT CHART

P1   ->P2   ->P3   ->

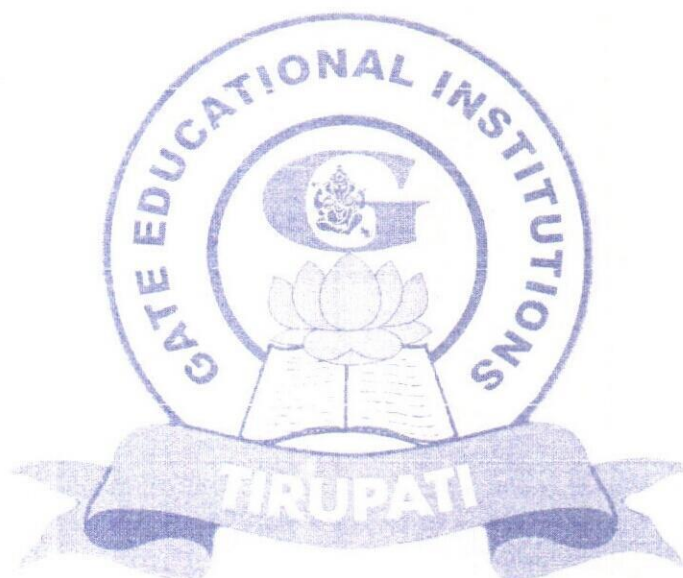
0   24   27   30

\*/



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>01. |

**Conclusion :** The Program Successfully demonstrates  
FCFS scheduling by computing waiting  
times, turnaround times, averages, &  
visualizing execution using a Gantt  
chart.





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>02. |

**Aim:** To write a java Program to implement the concept of shortest job first scheduling.

**Description:** To program simulates the SJF scheduling algorithm, a non-preemptive scheduling technique in operating system where the process with the smallest burst time is executed first.

The user inputs the number of processes & their burst times. The program then sorts the processes based on burst time, calculates waiting time for each process as the sum of the burst time of all previous processes, and computes the turnaround time as the sum of waiting time & burst time.

Finally, the program calculates the average waiting time & displays a table. A table showing process number, burst time, waiting time & turnaround time. This helps to evaluate the efficiency of the SJF scheduling in reducing waiting time compared to other scheduling methods like FCFS.

|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>02. |

Algorithm :

Step 1: Start.

Step 2: Input Data.

- Step 2.1: Read the number of Processes  $n$ .
- Step 2.2: Read burst time (BT) for each Processes & store in array  $BT[]$ .

Step 3: Initialize Arrays.

- Step 3.1: Initialize waiting time array  $WT[]$  and turnaround time array  $TAT[]$  to 0.
- Step 3.2: Initialize total waiting time variable  $AWT = 0$ .

Step 4: Sort Processes by Burst Time.

- Step 4.1: Compare each pair of burst times  $BT[j]$  and  $BT[j+1]$ .
- Step 4.2: If  $BT[j] > BT[j+1]$ , swap the burst times & also swap the corresponding waiting times.
- Step 4.3: Repeat until all Processes are sorted in ascending order of burst time.

Step 5: Calculate Waiting Time (WT) & Turnaround Time (TAT).

- Step 5.1: for the first process,  $WT[0] = 0$  &  $TAT[0] = BT[0]$ .
- Step 5.2: for each Subsequent Process  $i$  [from 1 to  $n-1$ ]:



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>02. |

- $WT[i] = TAT[i-1]$ .
- $TAT[i] = BT[i] + WT[i]$ .

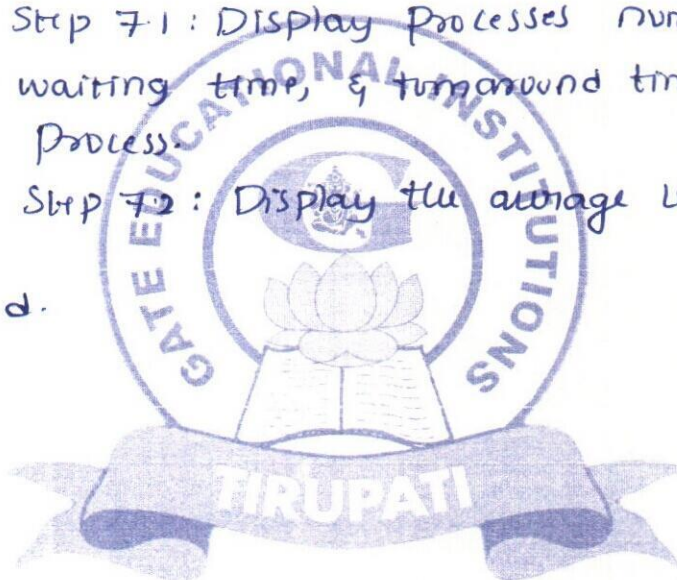
step 6: Calculate Average Waiting Time (AWT).

- step 6.1: Sum all waiting times :  $AWT = \sum WT[i] / n$ .

step 7: Display Results.

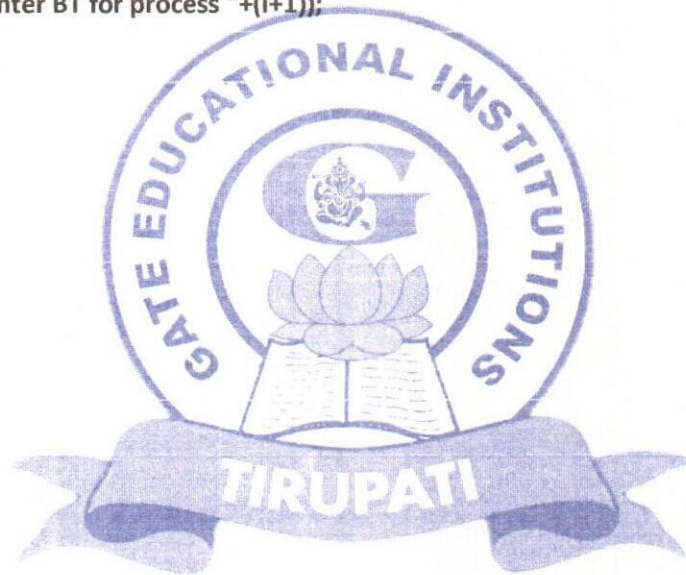
- Step 7.1: Display Processes number, burst time, waiting time, & turnaround time for each process.
- Step 7.2: Display the average waiting time.

Step 8: end.



|        |                                                                               |                 |
|--------|-------------------------------------------------------------------------------|-----------------|
| Date : | <div style="text-align: center;">4</div> <b>GATE Educational Institutions</b> | Page No. :      |
|        | //2. SHORTEST JOB FIRST SCHEDULING(SJF)                                       | Practical No. : |

```
import java.util.*;
class SJF
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int n,BT[],WT[],TAT[];
        System.out.println("Enter no of process");
        n=sc.nextInt();
        BT=new int[n+1];
        WT=new int[n+1];
        TAT=new int[n+1];
        float AWT=0;
        System.out.println("Enter Burst time for each process");
        for(int i=0;i<n;i++)
        {
            System.out.println("Enter BT for process "+(i+1));
            BT[i]=sc.nextInt();
        }
        for(int i=0;i<n;i++)
        {WT[i]=0;
            TAT[i]=0;
        }
        int temp;
        for(int i=0;i<n;i++)
        {for(int j=0;j<n-1;j++)
            { if(BT[j]>BT[j+1])
                {temp=BT[j];
                    BT[j]=BT[j+1];
                    BT[j+1]=temp;
                    temp=WT[j];
                    WT[j]=WT[j+1];
                    WT[j+1]=temp;
                }
            }
        }
        for(int i=0;i<n;i++)
        {TAT[i]=BT[i]+WT[i];
            WT[i+1]=TAT[i];
        }
        TAT[n]=WT[n]+BT[n];
        System.out.println(" PROCESS BT WT TAT ");
        for(int i=0;i<n;i++)
        System.out.println(" "+i+" "+BT[i]+" "+WT[i]+" "+TAT[i]);
        for(int j=0;j<n;j++)
            AWT+=WT[j];
        AWT=AWT/n;
        System.out.println("*****");
        System.out.println("Avg waiting
time="+AWT+"\n*****");
    }
}
```





|                      |                                                      |                 |
|----------------------|------------------------------------------------------|-----------------|
| Date :<br><br>}<br>} | <sup>5</sup><br><b>GATE Educational Institutions</b> | Page No. :      |
|                      |                                                      | Practical No. : |

/\*OUTPUT

Enter no of process

4

Enter Burst time for each process

Enter BT for process 1

5

Enter BT for process 2

3

Enter BT for process 3

3

Enter BT for process 4

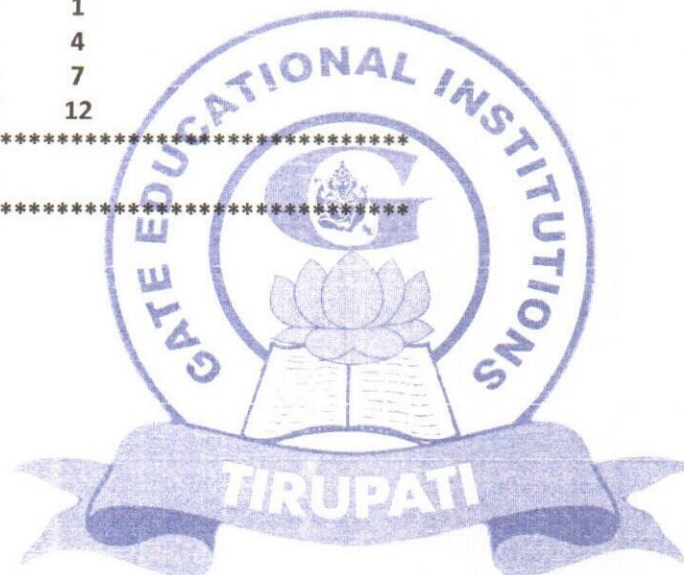
1

| PROCESS | BT | WT | TAT |
|---------|----|----|-----|
| 0       | 1  | 0  | 1   |
| 1       | 3  | 1  | 4   |
| 2       | 3  | 4  | 7   |
| 3       | 5  | 7  | 12  |

\*\*\*\*\*

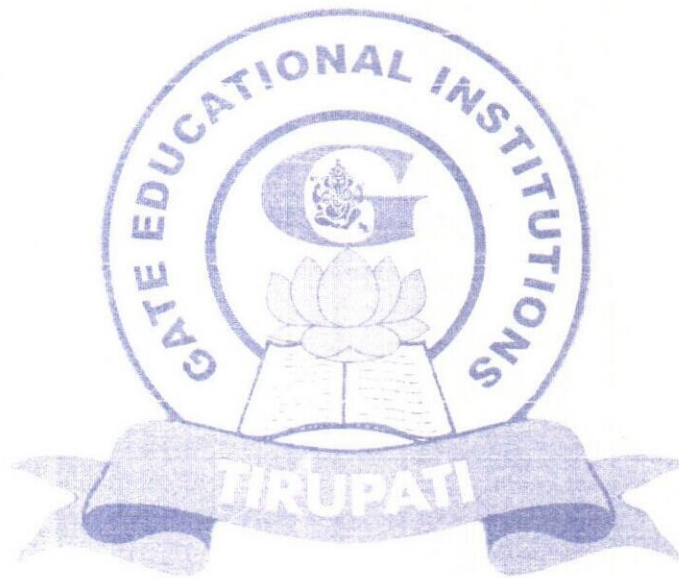
Avg waiting time=3.0

\*\*\*\*\*



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>02 |

conclusion: The SJF scheduling program Successfully calculate the waiting time turn around time and average waiting time by executing Processes in increasing order of burst time. This demonstrates how the sh





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>03. |

**AIM:** To write a Java program to implement the concept of round robin job scheduling.

**Description:** This program implements the Round Robin CPU scheduling algorithm using Java. Round Robin Scheduling is a Preemptive Scheduling technique where each process is assigned a fixed time quantum. The CPU executes each process for a given time slice, & if the process is not completed within that time, it is moved to the back of the ready queue. The program takes the number of processes, their burst times, & the time quantum as input. It calculates the waiting time, turnaround time, and their averages, & displays them in tabular form.

**Algorithm:**

Step-1: Start.

Step-2: Input Data.

- Step 2.1: Input the number of Processes  $n$ .
- Step 2.2: Input the burst time (BT) for each Process into the array  $bt[]$ .
- Step 2.3: Input the time quantum  $q$ .

Step-3: Initialize Arrays.

- Step 3.1: copy burst time into a temporary array  $a[]$  to preserve original BT values.
- Step 3.2: Initializing waiting time array  $wt[]$  with 0.
- Step 3.3: Initialize turnaround time array  $tat[]$ .

Step-4: Execute Round Robin Scheduling.

- Step 4.1: Repeat the following until all Processes are completed.
  - Step 4.1.1: for each process  $i$  from 0 to  $n-1$ .
    - Step 4.1.1.1: IF  $bt[i] > q$ , reduce  $bt[i]$  by  $q$  and add  $q$  to the waiting time of all other unfinished Processes.
    - Step 4.1.1.2: Else, add the remaining  $bt[i]$  to the waiting time of all other unfinished Processes & set  $bt[i] = 0$ .
- Step 4.2: Calculate the sum of  $bt[]$  to check if all Processes are completed. Repeat until the sum becomes 0.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>03. |

Step 5 : Calculate Turnaround Time (TAT)

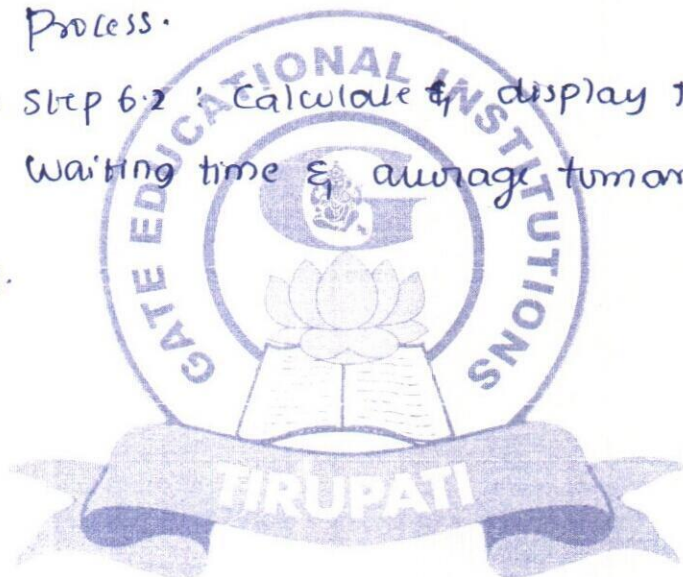
- Step 5.1: for each process  $i$ , calculate  

$$tat[i] = wt[i] + a[i]$$

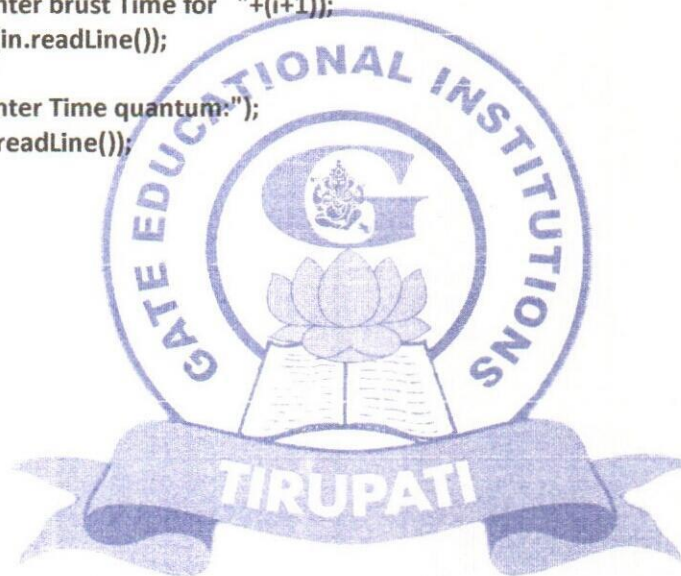
Step 6: Output Result.

- step 6.1: Display Process number, burst time, waiting time & turnaround time for each process.
- step 6.2: Calculate & display the average waiting time & average turnaround time.

Step 7: end.



|                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                   |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| Date :                           | <div style="text-align: center;">6</div> <b>GATE Educational Institutions</b><br>//3.ROUND ROBIN SCHEDULING                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Page No. :<br><br>Practical No. : |
| import java.io.*;<br>class round | <pre> { public static void main(String args[])throws IOException { DataInputStream in=new DataInputStream(System.in); int i,j,k,q,sum=0; System.out.println("Enter number of process:"); int n=Integer.parseInt(in.readLine()); int bt[]=new int[n]; int wt[]=new int[n]; int tat[]=new int[n]; int a[]=new int[n]; System.out.println("Enter brust Time:"); for(i=0;i&lt;n;i++) { System.out.println("Enter brust Time for "+(i+1)); bt[i]=Integer.parseInt(in.readLine()); } System.out.println("Enter Time quantum:"); q=Integer.parseInt(in.readLine()); for(i=0;i&lt;n;i++) a[i]=bt[i]; for(i=0;i&lt;n;i++) wt[i]=0; do { for(i=0;i&lt;n;i++) { if(bt[i]&gt;q) { bt[i]-=q; for(j=0;j&lt;n;j++) { if((j!=i)&amp;&amp;(bt[j]!=0)) wt[j]+=q; }} else { for(j=0;j&lt;n;j++) { if((j!=i)&amp;&amp;(bt[j]!=0)) wt[j]+=bt[i]; } bt[i]=0; }} sum=0; for(k=0;k&lt;n;k++) sum=sum+bt[k]; } </pre> |                                   |





Date :

Page No. :

```
while(sum!=0);
for(i=0;i<n;i++)
tat[i]=wt[i]+a[i];
```

Practical No. :

```
System.out.println("process\tBT\tWT\tTAT");
for(i=0;i<n;i++)
{
System.out.println("process" +(i+1) +"\t"+a[i]+"\t"+wt[i]+"\t"+tat[i]);
}
float avg_wt=0;
float avg_tat=0;
for(j=0;j<n;j++)
{
avg_wt+=wt[j];
}
for(j=0;j<n;j++)
{
avg_tat+=tat[j];
}
System.out.println("average wating time "+(avg_wt/n)+"\nAverage turn around
time" +(avg_tat/n));
}}
```

**OUTPUT**

/\*Round robin Scheduling algorithm output:

Enter number of process:4

Enter brust Time:

Enter brust Time for 1:4

Enter brust Time for 2:5

Enter brust Time for 3:6

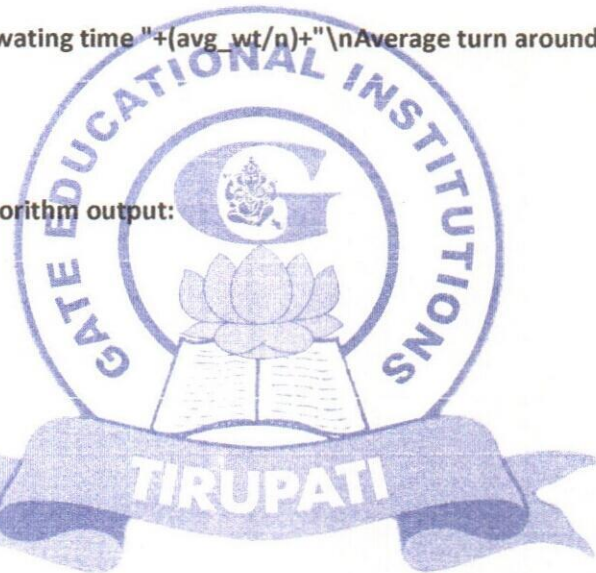
Enter brust Time for 4:7

Enter Time quantum:4

| process  | BT | WT | TAT |
|----------|----|----|-----|
| process1 | 4  | 0  | 4   |
| process2 | 5  | 12 | 17  |
| process3 | 6  | 13 | 19  |
| process4 | 7  | 15 | 22  |

average wating time 10.0

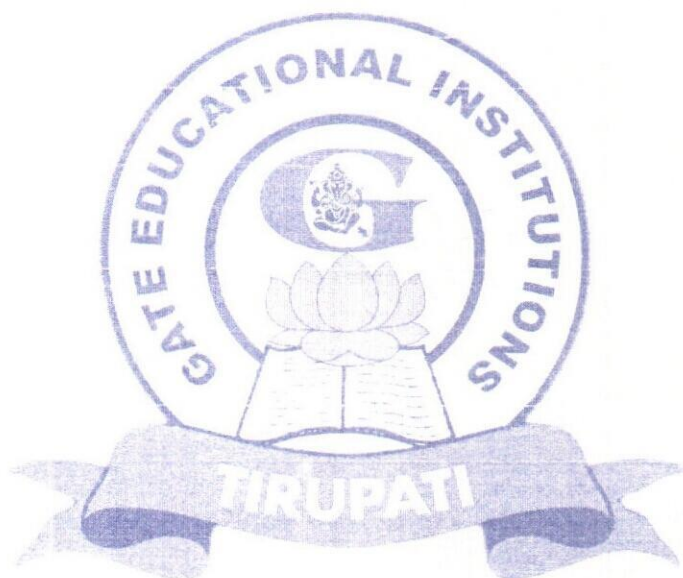
Average turn around time15.5



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>03. |

### Conclusion :

The round Robin Scheduling Program Successfully demonstrates time- Sharing cpu scheduling by fixed time quantum to each Processes. It ensure fairness among Processes & accurately Compares waiting time, turnaround time. & their averages.





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>04. |

**Aim:** To write a Java Program to implement the concept of Priority Scheduling.

**Description:** This Program implements the priority Scheduling algorithm used in an Operating System. Each process is assigned a priority, & the Process with the highest priority (lower Priority number) is executed first using a non-preemptive approach. The Program takes the number of processes as input along with their burst times & priority values. It then sorts the processes based on priority, calculates the waiting time and turnaround time for each process, & finally displays the results in a tabular format.

## Algorithm:

Step-1: Start.

Step-2: Input Data.

- Step 2.1: Input the number of processes  $n$ .

- Step 2.2: for each process, input:

- Burst Time (BT) into  $bt[]$ .

- Priority into  $pr[]$ .

- Step 2.3: Assign process ID's to  $pid[]$   
(eg, 1 to  $n$ ).

Step-3: Sort processes by priority.

- Step 3.1: Compare the priority of all processes

- Step 3.2: Swap processes so that the process with the highest priority (lowest number) comes first

- Step 3.3: Ensure that burst times & process ID's are swapping along with priorities.

Step-4: Calculate waiting Time (WT)

- Step 4.1: Set  $wt[0] = 0$  for the first process.

- Step 4.2: for each subsequent process  $i$   
(from 1 to  $n-1$ ):

- $wt[i] = wt[i-1] + bt[i-1]$ .

Step-5: calculate Turnaround Time (TAT)

- Step 5.1: for each process  $i$ , calculate  
 $tat[i] = wt[i] + bt[i]$ .

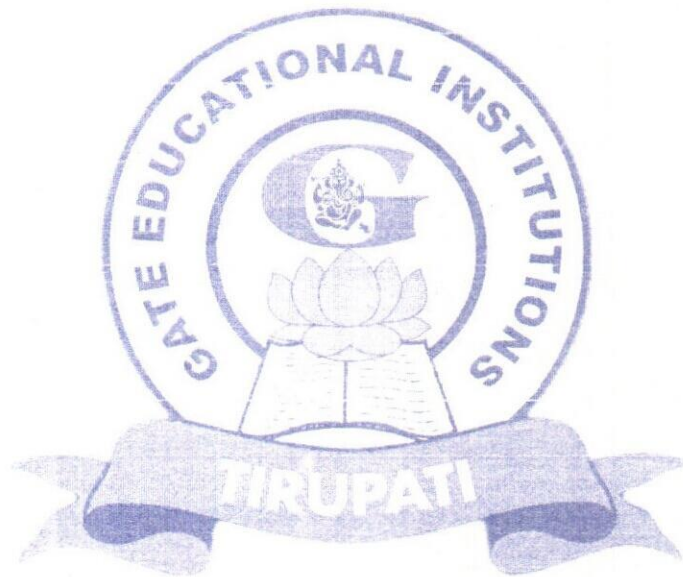


|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>04. |

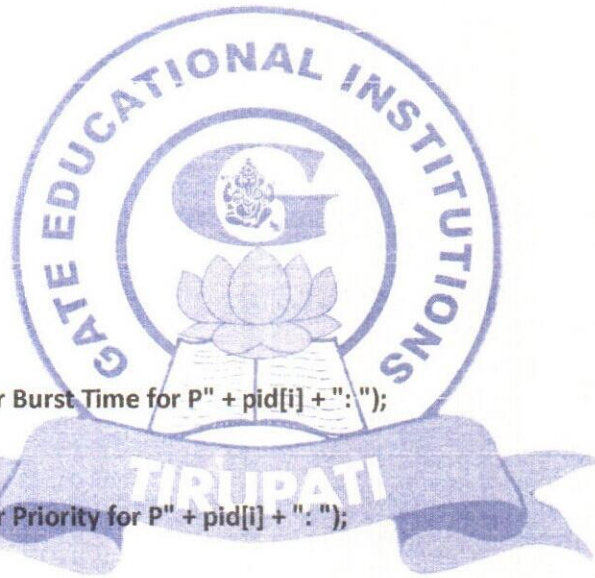
Step 6: Output Results.

- Step 6.1: Display the Process ID, Burst time, Priority, waiting Time, & Turnaround Time for each process.

Step 7: End.



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                           |                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|-----------------|
| Date :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 8<br><b>GATE Educational Institutions</b> | Page No. :      |
| // 4. PRIORITY SCHEDULING ALGORITHM:<br><br>import java.util.Scanner;                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                           | Practical No. : |
| <pre> class PrioritySchedulingOS {     public static void main(String[] args) {         Scanner sc = new Scanner(System.in);          System.out.print("Enter number of processes: ");         int n = sc.nextInt();          int pid[] = new int[n];         int bt[] = new int[n];         int pr[] = new int[n];         int wt[] = new int[n];         int tat[] = new int[n];         // Input process details         for (int i = 0; i &lt; n; i++) {             pid[i] = i + 1;             System.out.print("Enter Burst Time for P" + pid[i] + ": ");             bt[i] = sc.nextInt();             System.out.print("Enter Priority for P" + pid[i] + ": ");             pr[i] = sc.nextInt();         }          // Sort by priority (OS scheduler behavior)         for (int i = 0; i &lt; n - 1; i++) {             for (int j = i + 1; j &lt; n; j++) {                 if (pr[i] &gt; pr[j]) {                     int temp = pr[i];                     pr[i] = pr[j];                     pr[j] = temp;                 }             }         }     } } </pre> |                                           |                 |





|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                        |                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Date :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <div style="display: flex; justify-content: center; align-items: center;"> <span style="margin-right: 10px;">9</span> <h2 style="margin: 0;">GATE Educational Institutions</h2> </div> | Page No. :      |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <pre>temp = bt[i]; bt[i] = bt[j];</pre>                                                                                                                                                | Practical No. : |
| <pre>         bt[j] = temp;          temp = pid[i];         pid[i] = pid[j];         pid[j] = temp;     } }  // Waiting Time calculation wt[0] = 0; for (int i = 1; i &lt; n; i++) {     wt[i] = wt[i - 1] + bt[i - 1]; }  // Turnaround Time calculation for (int i = 0; i &lt; n; i++) {     tat[i] = wt[i] + bt[i]; }  System.out.println("\nPID\tBT\tPriority\tWT\tTAT"); for (int i = 0; i &lt; n; i++) {     System.out.println(pid[i] + "\t" + bt[i] + "\t" + pr[i] +         "\t\t" + wt[i] + "\t" + tat[i]); }  sc.close(); }     </pre> <div data-bbox="478 824 1161 1393" data-label="Image" style="text-align: center;"> </div> |                                                                                                                                                                                        |                 |

|                |                                            |                 |
|----------------|--------------------------------------------|-----------------|
| Date :         | 10<br><b>GATE Educational Institutions</b> | Page No. :      |
| <u>OUTPUT:</u> |                                            | Practical No. : |

Enter number of processes: 3

Enter Burst Time for P1: 10

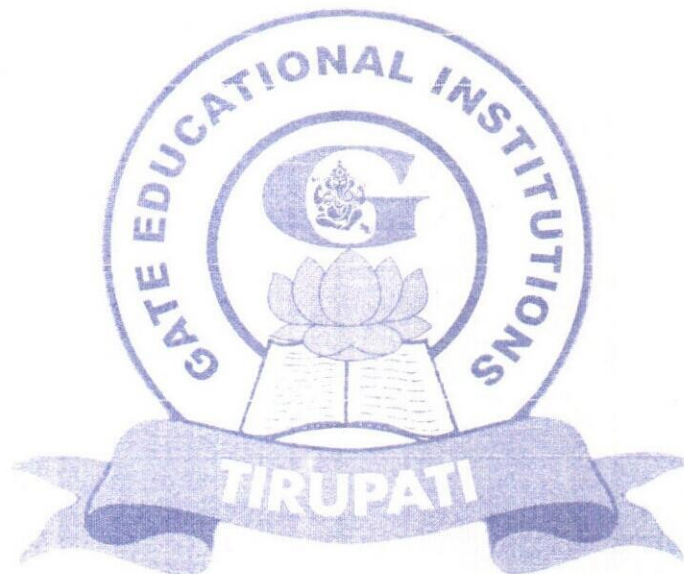
Enter Priority for P1: 3

Enter Burst Time for P2: 1

Enter Priority for P2: 1

Enter Burst Time for P3: 2

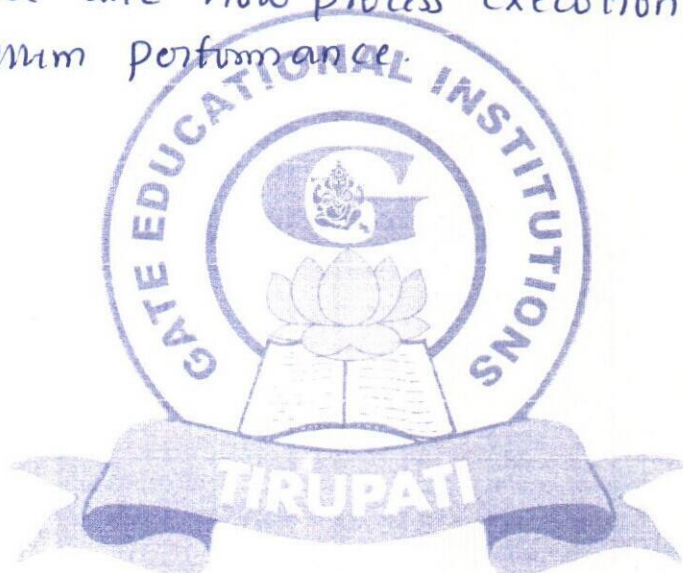
Enter Priority for P3: 2





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>04. |

**Conclusion:** The program Successfully demonstrates Priority Scheduling in an Operating System Using Java. It efficiently orders processes based on priority and accurately computes waiting & turnaround times. This implementation helps in understanding how cpu scheduling decision are made and how process execution order affects system performance.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>05. |

**Aim:** To write a java program to implement the concept of banker's algorithm.

**Description:** This program implements the Banker's Algorithm to prevent deadlocks in an operating system. It checks whether resources can be allowed safely to processes using available, claim, & allocation data. The program calculates the need of each process & determines whether all processes can complete safely. Finally, it displays whether the system is in a safe state or not.

**Algorithm:**

- Step 1 : Start.
- Step 2 : Input Data
  - Step 2.1 : Input the number of processes  $P$  & the number of resources  $r$ .
  - Step 2.2 : Input the available resources vector  $av[]$ .
  - Step 2.3 : Input the claim matrix  $claim[p][r]$ .
  - Step 2.4 : Input the allocation matrix  $alloc[p][r]$ .
- Step 3 : Calculate Initial Need.
  - Step 3.1 : for each resource, calculate  $need[i][j]$ 

$$= claim[i][j] - alloc[i][j]$$



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>05 |

- Step 3.2: calculate the remaining available resources after current allocations.

Step 4: check Resource Requests.

- step 4.1: for each process  $i$ :
  - step 4.1.1: if need  $[i]$   $\leq$  available resources, the process can complete.
  - step 4.1.2: Add the resources allocated to this process back to the available resources after completion.
- step 4.2: Repeat until all processes are checked or no more processes can be completed in this iteration.

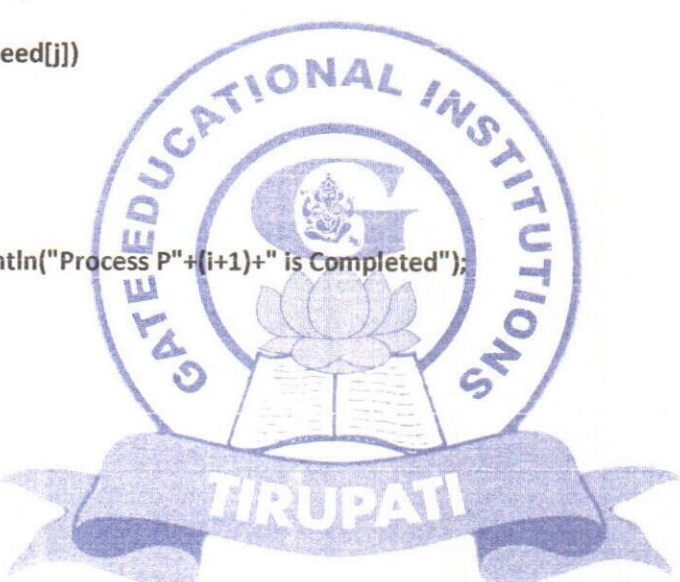
Step 5: Determine System State.

- step 5.1 : If all processes are completed, Print "system is in safe state".
- step 5.2: If one or more processes cannot complete, Print "System is Not in safe State".

Step 6: End.



|                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                 |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Date :                               | <div style="text-align: center;">11</div> <b>GATE Educational Institutions</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Page No. :      |
| import java.util.*;<br>class Bankers |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Practical No. : |
|                                      | <div style="text-align: center;">//5.BANKERS ALGORITHM</div> <pre> {     int claim[][] , alloc[][] , av[] , need[] , p , r , alsum[] , sum , temp[][];     int a=0 , b=0 , c;     boolean flag;     Scanner src = new Scanner(System.in);     Bankers(int p , int r)     {         this.p=p;         this.r=r;         claim=new int[p][r];         alloc=new int[p][r];         av=new int [r];         need=new int [r];         alsum=new int[r];         temp=new int[p][r];         System.out.println("ENTER THE "+r+" VALUES IN AVAILABLE VECTOR");         for(int i=0;i&lt;r;i++)         {             av[i]=src.nextInt();         }         System.out.println("ENTER THE CLAIM MATRIX OF "+p+" X "+r);         for(int i=0;i&lt;p;i++)             for(int j=0;j&lt;r;j++)                 claim[i][j]=src.nextInt();         System.out.println("ENTER THE ALLOCATION MATRIX OF "+p+" X "+r);         for(int i=0;i&lt;p;i++)             for(int j=0;j&lt;r;j++)                 alloc[i][j]=src.nextInt();     }     public void calNeed()     {         for(int i=0;i&lt;r;i++)         {             for(int j=0;j&lt;p;j++)             {                 sum=sum+alloc[j][i];             }             alsum[i]=sum;             sum=0;         }         for(int i=0;i&lt;r;i++)         {             need[i]=av[i]-alsum[i];             System.out.print("Initial Need Vector ");             System.out.print(need[i]);             System.out.println();             System.out.println();         }     } } </pre> |                 |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                     |                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-----------------|
| Date :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <div style="text-align: center;">12</div> <b>GATE Educational Institutions</b>      | Page No. :      |
| <pre> public void checkNeed() {     for(int i=0;i&lt;p;i++)         for(int j=0;j&lt;r;j++)         {             temp[i][j]=claim[i][j]-alloc[i][j];         } } </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                     | Practical No. : |
| <pre> public void compareNeed() {     while(b&lt;p)     {         flag=false;         for(int i=0;i&lt;p;i++)         {             for(int j=0;j&lt;r;j++)             {                 if(temp[i][0]==89)                     break;                 if(temp[i][j]&lt;=need[j])                     a++;                 else break;             }             if(a==r)             {                 System.out.println("Process P"+(i+1)+" is Completed");                 addtoNeed(i);                 temp[i][0]=89;                 a=0;                 c++;                 flag=true;             }             if(flag==true)                 break;         }         b++;     } } </pre> |  |                 |
| <pre> public void addtoNeed(int k) {     int n=k;     for(int j=0;j&lt;r;j++)     {         need[j]=need[j]+alloc[n][j];     }     if(c==p)         System.out.println("SYSTEM IS IN SAFE STATE");     else         System.out.println("SYSTEM IS NOT IN SAFE STATE"); } } </pre>                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                     |                 |

Date :

13

# GATE Educational Institutions

Page No. :

class BankerExp1

Practical No. :

```

{
    public static void main(String args[])
    {
        Scanner src = new Scanner(System.in);
        System.out.println("ENTER THE NO. OF PROCESS");
        int p= src.nextInt();
        System.out.println("ENTER THE NO. OF RESOURCES");
        int r = src.nextInt();
        Bankers obj = new Bankers(p,r);
        obj.calNeed();
        obj.checkNeed();
        obj.compareNeed();
    }
}

```

**OUTPUT**

ENTER THE NO. OF PROCESS:4

ENTER THE NO. OF RESOURCES:3

ENTER THE 3 VALUES IN AVAILABLE VECTOR:9 3 6

ENTER THE CLAIM MATRIX OF 4 X 3

3 2 2

6 1 3

3 1 4

4 2 2

ENTER THE ALLOCATION MATRIX OF 4 X 3

1 0 0

5 1 1

2 1 1

0 0 2

Initial Need Vector 1

Initial Need Vector 1

Initial Need Vector 2

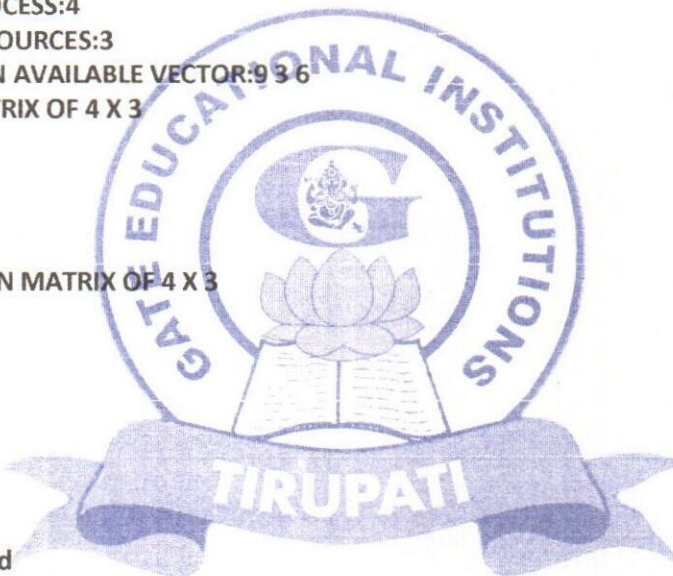
Process P2 is Completed

Process P1 is Completed

Process P3 is Completed

Process P4 is Completed

SYSTEM IS IN SAFE STATE





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>05. |

### Conclusion :

The program Successfully demonstrates the working of Banker's Algorithm for deadlock avoidance. It ensures that resources are allocated safely by checking system conditions before execution. By identifying whether the system is in a safe or unsafe state, this program helps in understanding how operating system prevent deadlocks and manage resources efficiently.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>06. |

**AIM:** To implement the Dining Philosophers Problem using Java & demonstrate how Synchronization helps avoid deadlock while multiple processes compete for shared resources.

**Description:** This program simulates the Dining philosophers problem, where five philosophers alternate between thinking & eating. To eat, a philosopher needs two shared forks. Each philosopher can be thinking, hungry, or eating. A hungry philosopher eats only if both neighbours are not eating. Otherwise, they wait. After eating, they return to thinking & notify neighbours to check if they can eat.

If you want, I can make it even simpler.

**Algorithm:**

Step-1: start the program.

- 1.1. Create an object of the Philo class.
- 1.2. Initialize an array state [5] to keep track of each philosopher's state (Thinking = 0, Eating = 1, Hungry = 2).



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>06 |

Step-2: Philosophers Become hungry.

- 2.1 Loop through all philosophers ( $i=0$  to  $4$ ).
- 2.2 Call Pickup ( $i$ ) for each philosopher:
  - 2.2.1 Set  $State[i] = \text{Hungry}$  (2).
  - 2.2.2 Print "philosopher  $i$  is hungry".
  - 2.2.3 Call test( $i$ ) to check if the philosopher can eat immediately.

Step-3: Check if a philosopher can eat (test( $i$ ))

- 3.1 If  $State[i] = \text{Hungry}$  AND both neighbours are not eating:
  - 3.1.1 Left Neighbour:  $(i+4) \% 5$
  - 3.1.2 Right Neighbour:  $(i+1) \% 5$ .
- 3.2 If neighbours are not eating, Set  $State[i] = \text{Eating}$  (1)
- 3.3 Print "philosopher  $i$  is eating".

Step-4: Waiting for forks.

- 4.1 If a philosopher Cannot eat after testing, Print "philosopher  $i$  is waiting".
- 4.2 Philosopher remains in hungry state until neighbors finish eating.

Step-5: Philosopher finishes eating (putdown( $i$ ))

- 5.1 change  $State[i] = \text{Thinking}$  (0)



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>06 |

5.2. Print "philosophers i is thinking".

5.3. call test () for both neighbours:

5.3.1  $(i+1) \% 5 \rightarrow$  Right neighbour

5.3.2  $(i+4) \% 5 \rightarrow$  Left neighbour

5.4. This check if any waiting philosophers can now start eating.

Step 6: Repeat for All philosophers.

6.1 loop through all philosophers again ( $i=0$  to  $4$ ) & call put-down ( $i$ )

6.2 This ensures every philosopher eventually eats safely.

Step-7: End program.

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 14                                   | Practical No. : |

//6.DINING PHILOSOPHER PROBLEM

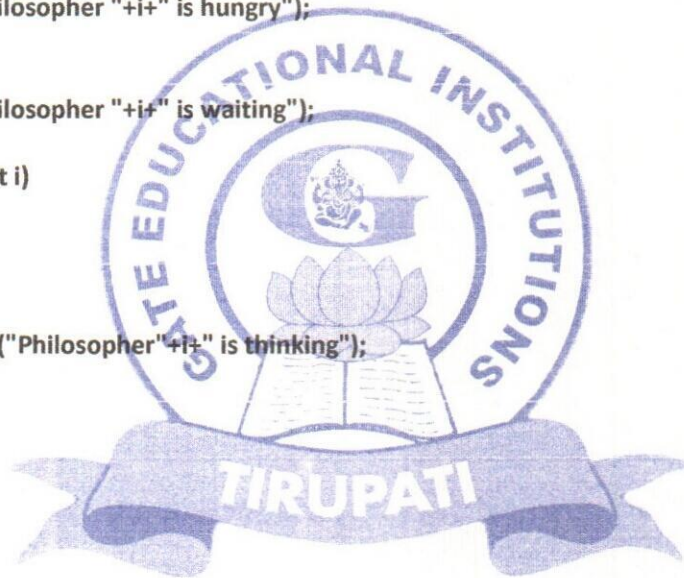
```

import java.io.*;
class Philo
{
int state[]=new int[5];
int thinking,eating,hungry;
void Op()
{
thinking=0;
eating=1;
hungry=2;
}
public void pickup(int i)
{
state[i]=2;
System.out.println("Philosopher "+i+" is hungry");
test(i);
if(state[i]==2)
System.out.println("Philosopher "+i+" is waiting");
}
public void putdown(int i)
{
if(state[i]==1)
{
state[i]=0;
System.out.println("Philosopher "+i+" is thinking");
test((i+1)%5);
test((i+5)%5);
}
}

public void test(int i)
{
if(state[i]==2 && state[(i+1)%5]!=1 && state[(i+4)%5]!=1)
{
state[i]=1;
System.out.println("philosopher "+i+" is eating");
}
}
}

class Philosopher
{
public static void main(String args[])throws IOException
{
Philo d=new Philo();
for(int i=0;i<5;i++)
d.pickup(i);
for(int i=0;i<5;i++)

```

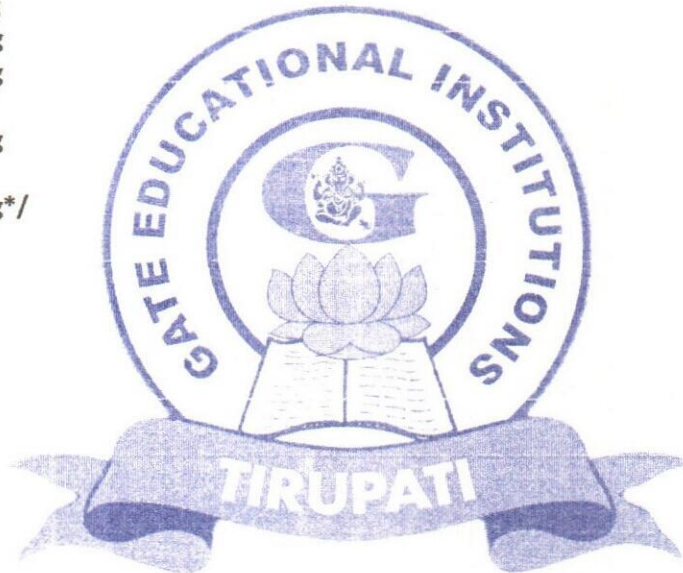


|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 15                                   | Practical No. : |

```
d.putdown(i);
}
}
```

/\* Output for dining philosophers problem

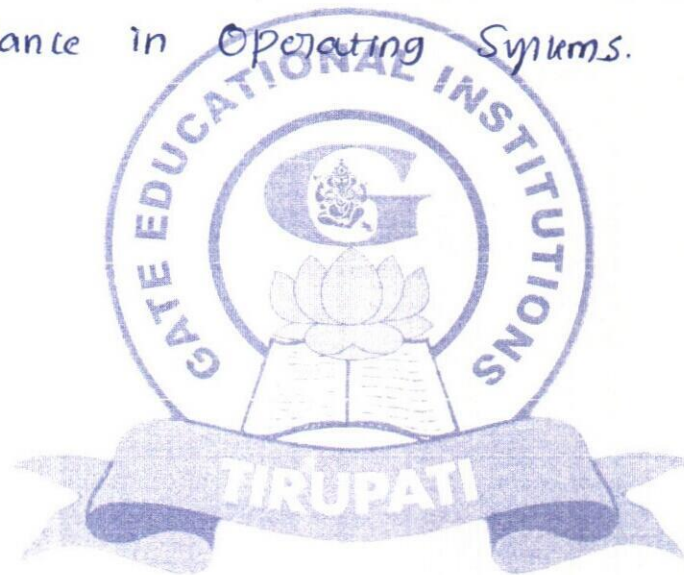
```
Philosopher 0 is hungry
philosopher 0 is eating
Philosopher 1 is hungry
Philosopher 1 is waiting
Philosopher 2 is hungry
philosopher 2 is eating
Philosopher 3 is hungry
Philosopher 3 is waiting
Philosopher 4 is hungry
Philosopher 4 is waiting
Philosopher0 is thinking
Philosopher2 is thinking
philosopher 3 is eating
Philosopher3 is thinking
philosopher 4 is eating
Philosopher4 is thinking*/
```





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>06. |

Conclusion: The Program Successfully demonstrates the Dining Philosophers Problem using Java. It shows how proper Synchronization can prevent deadlock when multiple processes share limited resources. This Simulation helps in understanding resource allocation, Process Coordination, & deadlock avoidance in Operating Systems.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>07. |

**AIM:** To implement a circular buffer based procedure-Consumer Simulator in Java that allows inserting and removing items from a fixed-size buffer.

**Description:** This Java program simulates a simple Producer-Consumer System using a circular buffer of size 10. The user can choose to produce (insert) items into the buffer or consume (remove) items from it. The buffer uses in and out pointers that wrap around modulo arithmetic.

- When producing, if the buffer is full, insertion is prevented.
- When consuming, if the buffer is empty, removal is prevented.
- The program continues until the user chooses to exit.

**Algorithm:**

- Step-1: Start
- Step-2: Initialize an integer array buffer of size 10.
- Step-3: Set  $bufferSize = 10$ ,  $in = 0$ ,  $out = 0$ .
- Step-4: Repeat until user enters 3 (exit).

|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>07 |

Step 5: Display menu:

1. Produce
2. Consume
3. Exit.

Step 6: Read User choice.

Step 7: If choice == 1 (produce).

7.1 Check if  $(in+1) \geq bufsize$  == out

7.2 If True → Buffer is full (display message).

7.3: Else → Prompt for value.

7.4: Insert the value at buffer [in].

7.5: Update  $in = (in+1) \% bufsize$ .

Step-8: If choice == 2 (consume).

8.1 Check if  $in \geq out$

8.2 If True → Buffer is Empty (display message)

8.3 Else → Read value from buffer [out]

8.4 Display Consumed Value

8.5 Update  $out = (out+1) \% bufsize$

Step-9: If choice == 3 → exit loop.

Step-10: end



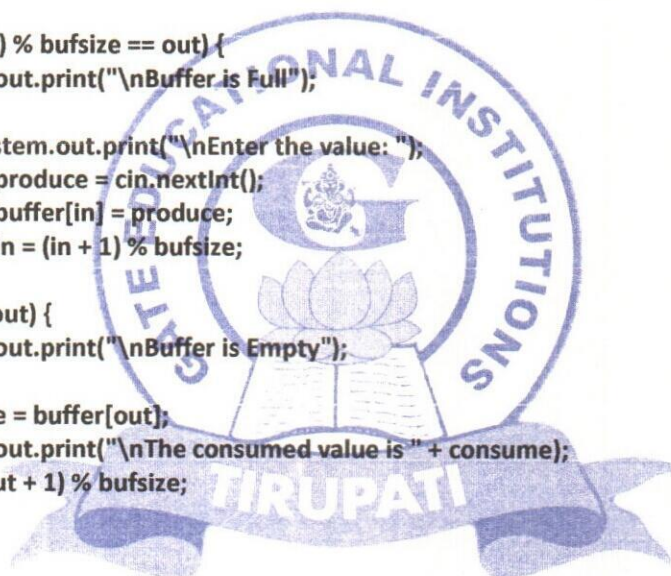
|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 16                                   | Practical No. : |

//7.producer and consumer problem

```

import java.util.Scanner;
public class Prod1{
public static void main(String args[]) {
    int[] buffer = new int[10];
    int bufsize, in, out, produce, consume, choice = 0;
    in = 0;
    out = 0;
    bufsize = 10;
    Scanner cin = new Scanner(System.in);
    while(choice != 3) {
        System.out.print("\n1. Produce \n 2. Consume \n3. Exit");
        System.out.print("\nEnter your choice: ");
        choice = cin.nextInt();
        switch(choice) {
            case 1:
                if((in + 1) % bufsize == out) {
                    System.out.print("\nBuffer is Full");
                }
                else {System.out.print("\nEnter the value: ");
                    produce = cin.nextInt();
                    buffer[in] = produce;
                    in = (in + 1) % bufsize;
                }break;
            case 2: if(in == out) {
                System.out.print("\nBuffer is Empty");
            } else {
                consume = buffer[out];
                System.out.print("\nThe consumed value is" + consume);
                out = (out + 1) % bufsize;
            }break;
        }
    }
}
}

```



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 17                                   | Practical No. : |

/\*OUTPUT

D:\Java>java Prod1

1. Produce

2. Consume

3. Exit

Enter your choice: 1

Enter the value: 15

1. Produce

2. Consume

3. Exit

Enter your choice: 1

Enter the value: 20

1. Produce

2. Consume

3. Exit

Enter your choice: 1

Enter the value: 30

1. Produce

2. Consume

3. Exit

Enter your choice: 2

The consumed value is 15

1. Produce

2. Consume

3. Exit

Enter your choice: 2

The consumed value is 20

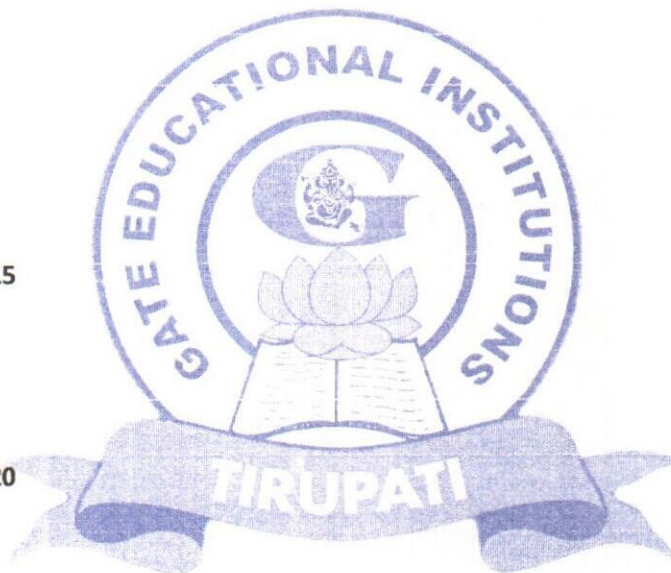
1. Produce

2. Consume

3. Exit

Enter your choice:3

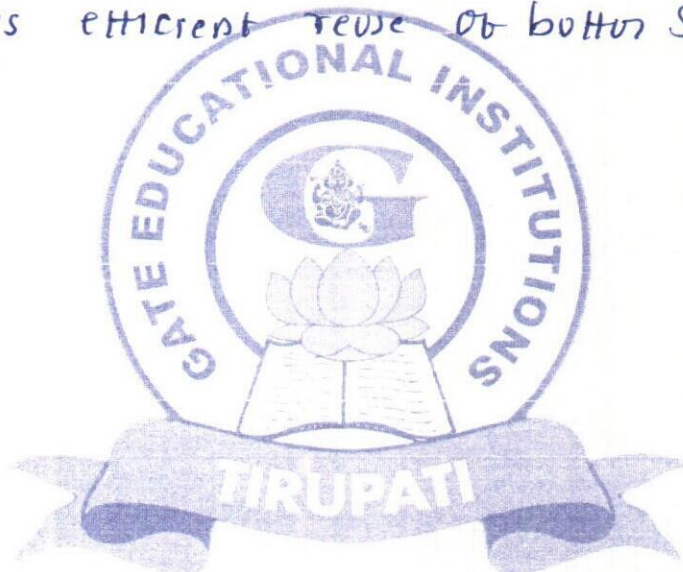
\*/



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>07 |

### Conclusion:

The Program Successfully Implements a Circular queue (buffets) for producer & Consumer operations. It handles buffer overflow & underflow conditions & repeatedly accepts user input until the exit option is selected. The correct use of modular arithmetic ensures efficient reuse of buffer space.





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>08. |

**Aim:** To implement the Best fit memory Allocation algorithm in Java for assigning files to memory blocks such that the smallest sufficient block is Selected for each file.

**Description:**

This Java program Simulates the Best fit Strategy of memory management. The user enters the number of memory blocks & files along with their sizes. For each file, the program searches through the list of available blocks & assigns the file to the block that leaves the smallest left over space. Once a block is allocated, it is marked as unavailable for further allocation. Finally, the program displays each file's assigned block & the resulting fragmentation.

**Algorithm:**

Step-1: Start.

Step-2: Initialize array for blocks (b), files (f), block flag (bf), file-to-block map (ff), & fragmentation (frag).

Step-3: Read number of blocks (nb) & number of files (nf).

Step-4: Input size of each block into array  $b[1..nb]$

Step 5: Input size of each file into array  $f[1..nf]$ .

Step 6: For each file  $i$  from 1 to  $nf$ .

6.1 : Set lowest = a large value (eg., 9999)

6.2 : for each block  $j$  from 1 to  $nb$ .

6.2.1 : if block  $j$  is not allocated ( $bf[j] \neq 1$ )

6.2.1.1 : calculate  $temp = b[j] - f[i]$

6.2.1.2 : If  $temp \geq 0$  &  $temp < lowest$ .

6.2.1.2.1 : Set  $lowest = temp$ .

6.2.1.2.2 : Assign  $ff[i] = j$  (block index for file  $i$ )

6.3. After Scanning all blocks.

6.3.1 : Store fragmentation  $frag[i] = lowest$ .

6.3.2 : mark chosen block as allocated:  $bf[ff[i]] = 1$ .

Step-7: Display table of file No, file size, Block No, Block size, and fragmentation.

Step-8: end.



Date :

Page No. :

Practical No. :

## //8. Memory Allocation-Best Fit

```

import java.util.Scanner;
public class Best1 {
    public final static int MAX = 25;
    private static int[] bf = new int[MAX], ff = new int[MAX];
    public static void main(String args[]) {
        int[] frag = new int[MAX], b = new int[MAX], f = new int[MAX];
        int i, nb, nf, temp, lowest = 9999;
        Scanner cin = new Scanner(System.in);
        System.out.print("\nEnter the number of blocks:");
        nb = cin.nextInt();
        System.out.print("Enter the number of files:");
        nf = cin.nextInt();
        System.out.println("\nEnter the size of the blocks:-");
        for(i = 1; i <= nb; i++) {
            System.out.println("Block " + i + ":");
            b[i] = cin.nextInt();
            System.out.println("Enter the size of the files :-");
            for(i = 1; i <= nf; i++) {
                System.out.print("File " + i + ":");
                f[i] = cin.nextInt();
            }
            for(i = 1; i <= nf; i++) {
                for(int j = 1; j <= nb; j++) {
                    if(bf[j] != 1) {
                        temp = b[j] - f[i];
                        if(temp >= 0) {
                            if(lowest > temp) {
                                ff[i] = j;
                                lowest = temp;
                            }
                        }
                    }
                }
            }
            frag[i] = lowest;
            bf[ff[i]] = 1;
            lowest = 9999;
        }
        System.out.print("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
        for(i = 1; i <= nf && ff[i] != 0; i++) {
            System.out.print("\n" + i + "\t\t" + f[i] + "\t\t" + ff[i] + "\t\t" + b[ff[i]] + "\t\t" + frag[i]);
        }
    }
}

```



Date :

# GATE Educational Institutions

Page No. :

19

Practical No. :

/\*OUTPUT

D:\Java>javac Best1.java

D:\Java>java Best1

Enter the number of blocks:3

Enter the number of files:3

Enter the size of the blocks:-

Block 1:

100

Block 2:

200

Block 3:

50

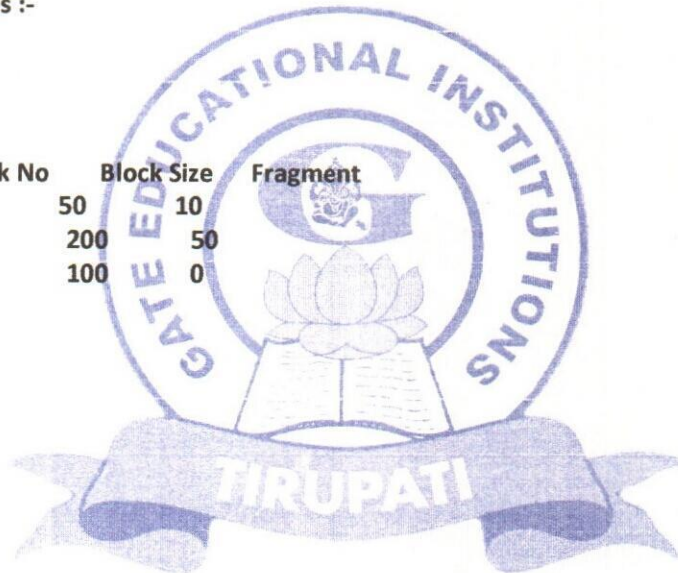
Enter the size of the files :-

File 1:40

File 2:150

File 3:100

| File No | File Size | Block No | Block Size | Fragment |
|---------|-----------|----------|------------|----------|
| 1       | 40        | 3        | 50         | 10       |
| 2       | 150       | 2        | 200        | 50       |
| 3       | 100       | 1        | 100        | 0        |



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>08. |

conclusion:

The program correctly implements the Best fit memory allocation technique by assigning each file to the most appropriately sized memory block available. The result shows efficient utilization of memory with minimum fragmentation for each file. If no suitable block remains for a file, it is not allocated. This helps in understanding dynamic memory allocation strategies used in operating system.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>09. |

**AIM:** To write a Java program to implement the concept of memory fragmentation.

**Description:** This Java Program Simulates memory allocation using fixed partitioning. The user enters total memory size, block size & number of processes with their memory requirements. The total memory is divided into fixed size partitions (blocks). Each process is allocated a block if its requirement memory is less than or equal to the block size. The Program displays whether each process is allocated, computes the internal fragmentation for each allocated block & finally shows the total internal & external fragmentations.

**Algorithm:**

Step-1: Start

Step-2: Input total memory size (ms) & block size (bs)

Step-3: calculate number of blocks:  $nob = ms / bs$

Step-4: calculate external fragments:  $ef = ms - (nob * bs)$

Step-5: Input number of processes (n).



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>09. |

Step-6: for  $i=0$  to  $n-1$

Input memory required for process  $i$  into array  $mp[i]$ .

Step-7: Display number of blocks available.

Step-8: Initialize total internal fragmentation ( $tif$ ) = 0  
& block pointer  $p=0$ .

Step-9: for  $i=0$  to  $n-1$  &  $p < nob$ .

9.1: print processes number & memory

requirement

9.2: If  $mp[i] > bs$

9.2.1 mark as "Not allocated".

9.3: Else

9.3.1 Allocate block  $\rightarrow$  print "Yes" &

$(bs - mp[i])$

9.3.2 Add  $(bs - mp[i])$  to total

internal fragmentation ( $tif$ )

9.3.3  $p = p + 1$

Step-10: If there are unallocated processes due to no free blocks.

10.1 Display that remaining processes cannot be accommodated.

Step-11: Print total internal fragmentation ( $tif$ ) & total external fragmentation ( $et$ )

Step-12: End.

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 20                                   | Practical No. : |

```

//9.Memory Fragmentation
import java.util.Scanner;
public class Mft {
    public static void main(String[] args) {
        int ms, bs, nob, ef, n, tif = 0;
        int[] mp = new int[10];
        int i, p = 0;
        Scanner cin = new Scanner(System.in);
        System.out.print("Enter the total memory available (in Bytes) -- ");
        ms = cin.nextInt();
        System.out.print("Enter the block size (in Bytes) -- ");
        bs = cin.nextInt();
        nob = ms / bs;
        ef = ms - nob * bs;
        System.out.print("\nEnter the number of processes -- ");
        n = cin.nextInt();
        for(i = 0; i < n; i++) {
            System.out.print("Enter memory required for process " + (i + 1) + " (in Bytes)-- ");
            mp[i] = cin.nextInt();
        }
        System.out.print("\nNo. of Blocks available in memory -- " + nob);
        System.out.print("\n\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL\nFRAGMENTATION");
        for(i = 0; i < n && p < nob; i++) {
            System.out.print("\n " + (i + 1) + "\t\t" + mp[i]);
            if(mp[i] > bs) {
                System.out.print("\t\tNO\t\t--");
            } else {
                System.out.print("\t\tYES\t\t" + (bs - mp[i]));
            }
            tif = tif + bs - mp[i];
            p++;
        }
        if(i < n)
        {System.out.print("\nMemory is Full, Remaining Processes cannot be accomodated");
        }
        System.out.print("\n\nTotal Internal Fragmentation is " + tif);
        System.out.print("\nTotal External Fragmentation is " + ef);
    }
}

```



Date :

# GATE Educational Institutions

Page No. :

21

Practical No. :

/\*OUTPUT

D:\Java>java Mft

Enter the total memory available (in Bytes) -- 1000

Enter the block size (in Bytes) -- 300

Enter the number of processes -- 5

Enter memory required for process 1 (in Bytes)-- 260

Enter memory required for process 2 (in Bytes)-- 400

Enter memory required for process 3 (in Bytes)-- 300

Enter memory required for process 4 (in Bytes)-- 150

Enter memory required for process 5 (in Bytes)-- 230

No. of Blocks available in memory -- 3

| PROCESS | MEMORY REQUIRED | ALLOCATED | INTERNAL FRAGMENTATION |
|---------|-----------------|-----------|------------------------|
|---------|-----------------|-----------|------------------------|

|   |     |     |    |
|---|-----|-----|----|
| 1 | 260 | YES | 40 |
|---|-----|-----|----|

|   |     |    |     |
|---|-----|----|-----|
| 2 | 400 | NO | --- |
|---|-----|----|-----|

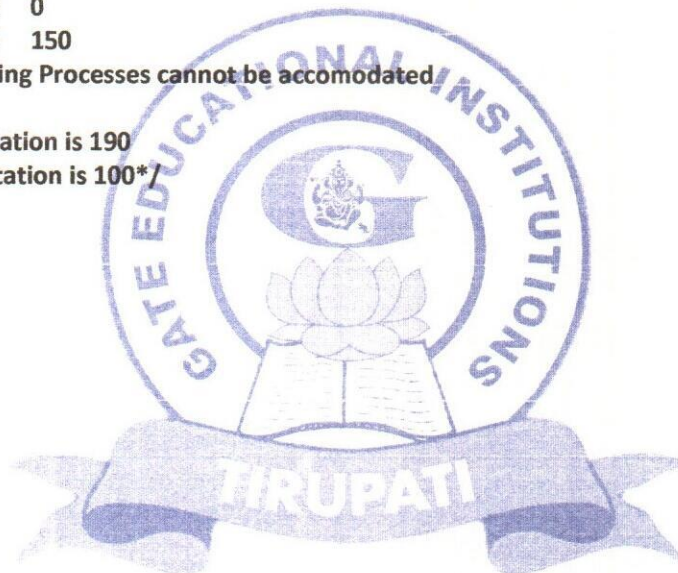
|   |     |     |   |
|---|-----|-----|---|
| 3 | 300 | YES | 0 |
|---|-----|-----|---|

|   |     |     |     |
|---|-----|-----|-----|
| 4 | 150 | YES | 150 |
|---|-----|-----|-----|

Memory is Full, Remaining Processes cannot be accomodated

Total Internal Fragmentation is 190

Total External Fragmentation is 100\*/

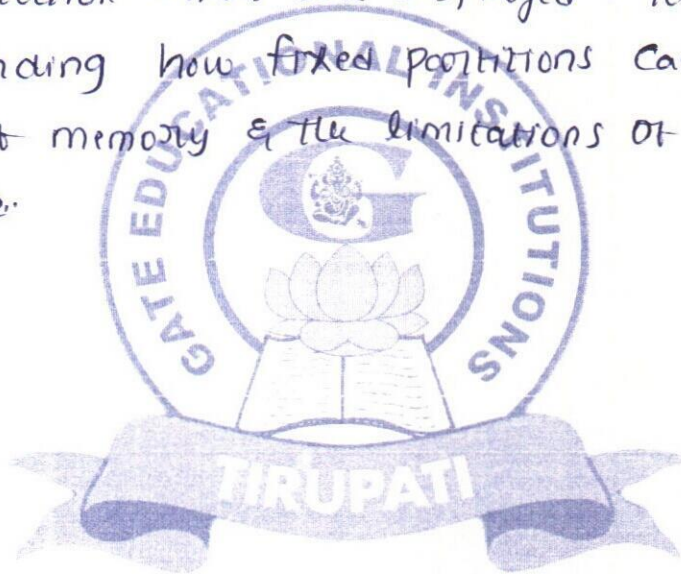




|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>09. |

### Conclusion:

The program Successfully demonstrates fixed Partition memory Allocation. It Partitions memory into fixed blocks & allocates them to Processes if possible. Internal fragmentation is calculated for each allocated Processes, & total internal & external fragmentation values are displayed. This helps in understanding how fixed partitions can lead to waste of memory & the limitations of this allocation technique.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>10. |

**Aim:** To implement the first-in first-out (fifo) Page Replacement algorithm in Java & Compute the Number of page hits, faults & the hit ratio for a given Page reference String & number of frames.

**Description:**

This program Simulates the fifo (first-in first out) Page Replacement algorithm, a basic memory management technique used in operating systems, where pages are loaded in fixed number of frames & if a referenced page is already present it is counted as a hit, otherwise it results in a page fault; when all frames are full and a new page must be loaded, the oldest page in memory (the one that arrived first) is replaced with the new one, & the program tracks & displays the frame contents over time & finally prints the total number of hits, the hit ratio, & the total faults.

**Algorithm:**

- Step-1: Start
- Step-2: Read number of frames (frames)
- Step-3: Read length of reference string (ref\_len)
- Step-4: Read reference string into array reference [0.. ref\_len-1].
- Step-5: Initialize buffer [0.. frames-1] = -1 (empty).



Step-6: Initialize  $pointer = 0$ ,  $hit = 0$ ,  $fault = 0$ .

Step-7: for  $i$  from 0 to  $ref\_len - 1$ :

7.1 Set  $Search = -1$

7.2 for  $j$  from 0 to  $frames - 1$

7.2.1 if  $buttons[j] == reference[i]$ :

7.2.2  $Search = j$

7.2.3  $hit++$  // Page already in memory.

7.3 if  $Search == -1$  (Page fault)

7.3.1  $button[pointer] = reference[i]$  // Replace a  
Pointer.

7.3.2  $fault++$

7.3.3  $pointer = pointer + 1$

7.3.4 If  $pointer == frames$ :

7.3.5  $pointer = 0$ .

7.4 Store current button state in  $mem\_layout[i][j]$ .

Step-8: Display frame contents over the reference  
Sequence.

Step-9: Print total hits, hit ratio ( $hit / ref\_len$ ), & total  
faults.

Step-10: End.



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 22                                   | Practical No. : |

//10.FIFO PAGE REPLACEMENT

```
import java.io.*;
public class FIFO12 {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int frames, pointer = 0, hit = 0, fault = 0, ref_len;
        int buffer[];
        int reference[];
        int mem_layout[][];

        System.out.println("Please enter the number of Frames: ");
        frames = Integer.parseInt(br.readLine());

        System.out.println("Please enter the length of the Reference string: ");
        ref_len = Integer.parseInt(br.readLine());

        reference = new int[ref_len];
        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
            buffer[j] = -1;

        System.out.println("Please enter the reference string: ");
        for(int i = 0; i < ref_len; i++)
        {
            reference[i] = Integer.parseInt(br.readLine());
        }
        System.out.println();
        for(int i = 0; i < ref_len; i++)
        {
            int search = -1;
            for(int j = 0; j < frames; j++)
            {
                if(buffer[j] == reference[i])
                {
                    search = j;
                    hit++;
                    break;
                }
            }
            if(search == -1)
            {
                buffer[pointer] = reference[i];
                fault++;
                pointer++;
                if(pointer == frames)
                    pointer = 0;
            }
        }
    }
}
```

Date :

**GATE Educational Institutions**

Page No. :

23

Practical No. :

```

    }
    for(int j = 0; j < frames; j++)
        mem_layout[i][j] = buffer[j];
    }

    for(int i = 0; i < frames; i++)
    {
        for(int j = 0; j < ref_len; j++)
            System.out.printf("%3d ", mem_layout[j][i]);
        System.out.println();
    }

    System.out.println("The number of Hits: " + hit);
    System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
    System.out.println("The number of Faults: " + fault);
}
}

```

/\*OUTPUT:-

Please enter the number of Frames: 3

Please enter the length of the Reference string: 20

Please enter the reference string:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 7 7 7

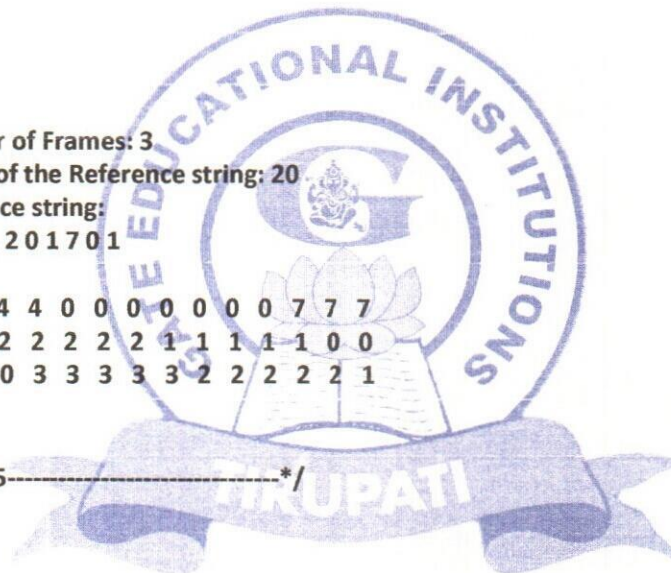
-1 0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 0 0

-1 -1 1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2 2 1

The number of Hits: 5

Hit Ratio: 0.25

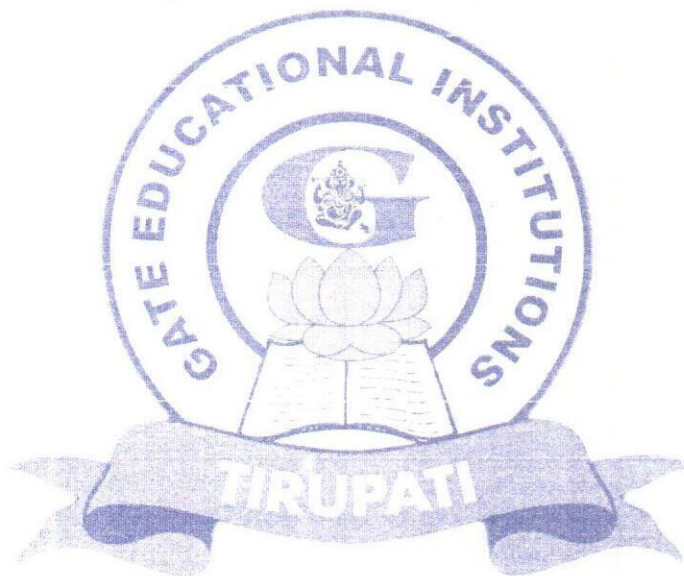
The number of Faults: 15-----\*/



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>10. |

conclusion:

The program implements the fifo Page replacement algorithm, showing how pages are loaded into fixed memory frames & counting hits when a page is already in memory & faults when it must be loaded. It calculates a hit ratio to measure performance. fifo is simple and easy to implement with low





|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>11. |

**Aim:** To implement the Least Recently Used (LRU) Page Replacement algorithm in Java for managing Page frames and counting the number of Page faults based on recent Page Usage.

**Description:** This Java Program Simulates the LRU Page replacement Strategy, a common memory management technique used in operating systems to decide which page to replace when a new page must be located but all frames are full. LRU works on the principle that the page that has not been used for the longest time is the one least likely to be used soon, so the page is replaced first when needed. The program accepts the number of frames & the sequence of pages, load pages into frames, & replace using the LRU logic. It displays the frame content after each insertion & finally prints the total number of Page faults.

|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>11. |

Algorithm :

Step-1: Start

Step-2: Read number of frames (size) & number of Pages (pages).

Step-3: Initialize arrays : frame [size], Present [size], Page [pages]

Step-4: Read the page reference string into Page [].

Step-5: Set all frame entries to -1 (empty)

Step-6: Initialize page fault count pf = 0.

Step-7: for each Page index i from 0 to Page - 1.

7.1. If frame contains Page[i] (check returns >= 0) → no fault.

7.2: Else → Page fault.

7.2.1. If frames not filled yet ( $i < \text{size}$ ), load Page directly.

7.2.2. Else find the least recently used page using replace (i).

7.2.3. Replace the frame with Page[i].

7.2.4. Increment pf.

7.3: Display current frame contents.

Step-8: After all references, Print total Page faults (pf).

Step-9: End.



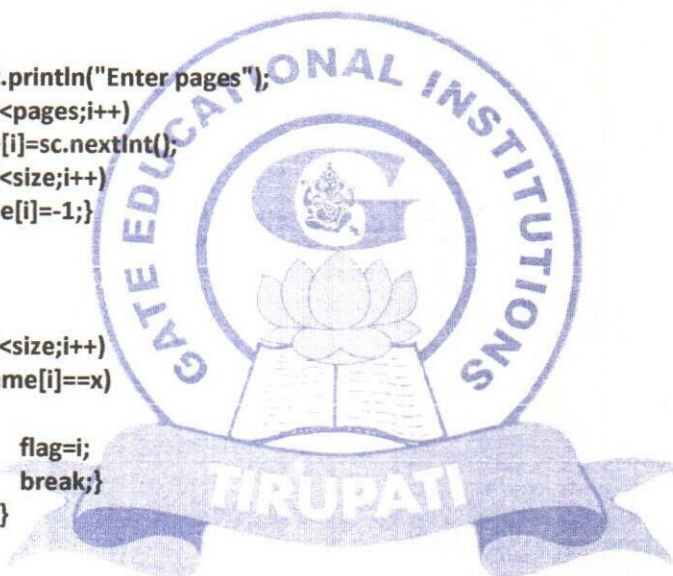
|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 24                                   | Practical No. : |

**//11.LRU PAGE REPLACEMENT**

```

import java.util.*;
class LRU
{
    Scanner sc=new Scanner(System.in);
    int[] frame,page,present;
    int size,pages,pf=0,flag=0,flag1=0;
    LRU(int size,int pages)
    {
        this.size=size;
        this.pages=pages;
        frame=new int[size];
        present=new int[size];
        page=new int[pages];}
    void get()
    {
        System.out.println("Enter pages");
        for(int i=0;i<pages;i++)
            page[i]=sc.nextInt();
        for(int i=0;i<size;i++)
            frame[i]=-1;}
    int check(int x)
    {
        flag=-1;
        for(int i=0;i<size;i++)
            if(frame[i]==x)
            {
                flag=i;
                break;}
        return flag;}
    int replace(int x)
    {
        int i;
        for(i=0;i<size;i++)
            present[i]=0;
        flag1=0;
        for(i=x-1;i>=0;i--)
        {
            if(check(page[i])>=0)
            {
                flag1++;
                present[check(page[i])]=1;}
            if(flag1==(size-1)) break;
        }
        for(i=0;i<size;i++)
            if(present[i]==0)
            {

```





|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 25                                   | Practical No. : |

```

        flag1=i;
        break;}

    return i;
}
void lru()
{
    for(int i=0;i<pages;i++)
    {
        if(i<size)
        {
            frame[i]=page[i];
            pf++;
            for(int j=0;j<size;j++)
                System.out.print(frame[j]+" ");
            System.out.println();
        }
        else
        {
            if(check(page[i])==-1)
            {
                int r=replace(i);
                frame[r]=page[i];
                pf++;
                for(int j=0;j<size;j++)
                    System.out.print(frame[j]+" ");
                System.out.println();
            }
            else
            {
                for(int j=0;j<size;j++)
                    System.out.print(frame[j]+" ");
                System.out.println();
            }
        }
        System.out.println("PAGE FAULT: "+pf);
    }
}

class KLRU
{
    public static void main(String arg[])
    {
        Scanner s=new Scanner(System.in);
        System.out.print("Enter frame size:");
        int n=s.nextInt();
        System.out.print("Enter no of pages:");
        int p=s.nextInt();
        LRU obj=new LRU(n,p);
        obj.get();
        obj.lru();
    }
}

```

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 26                                   | Practical No. : |

#### OUTPUT

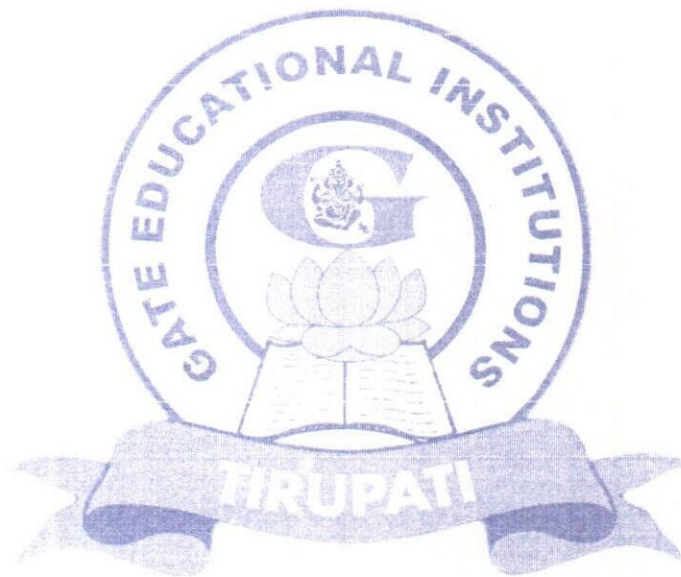
```

D:\Java>java KLRU
Enter frame size:3
Enter no of pages:20
Enter pages
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 -1 -1
7 0 -1
7 0 1
2 0 1
2 0 1
2 0 3
2 0 3
4 0 3
4 0 2
4 3 2
0 3 2
0 3 2
0 3 2
1 3 2
1 3 2
1 0 2
1 0 2
1 0 7
1 0 7
1 0 7

```

PAGE FAULT: 12

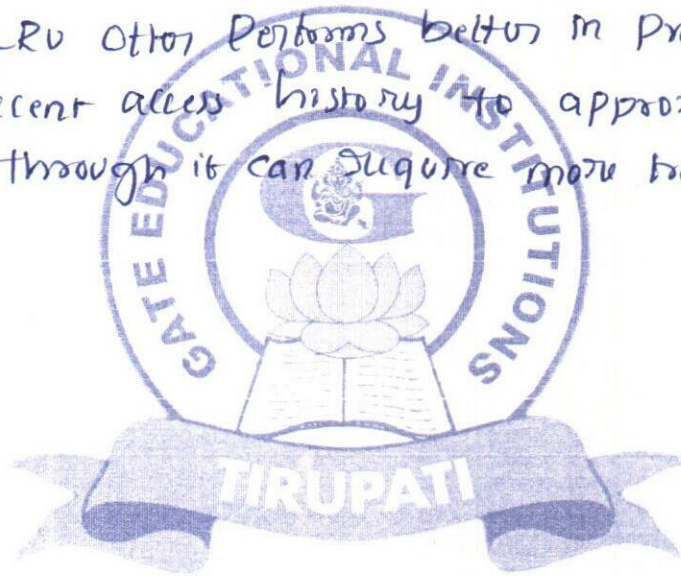
D:\Java>



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>11. |

### Conclusion:

The Program Successfully implements the LRU Page Replacement algorithm by loading Pages into fixed frames & replacing the least recently used Page when necessary. It accurately counts & displays Page faults compared with Simpler Strategies like FIFO. LRU often performs better in practice because it uses recent access history to approximate future use, though it can require more tracking overhead.





|        |                                      |                              |
|--------|--------------------------------------|------------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :                   |
|        |                                      | Practical No. :<br><b>12</b> |

Aim:

To write a java program to implement the concept of optimal page replacement.

Description:

This Java program simulates the optimal Page Replacement Algorithm for efficient memory management. It takes the number of memory frames, the length of the reference string, and the reference string as input, and processes each page sequentially. If a page is already in memory, it is counted as a hit, if not, a page fault occurs. When memory is full, the program replaces the page that will not be used for the longest time in the future, following the optimal strategy. The program displays the memory layout at each step along with the total hits, hit ratio, and page faults, demonstrating how the algorithm minimizes page faults and manages memory effectively.

Algorithm:

Step 1: Start

Step 2: Initialize variables and arrays

- Step 2.1: Declare variables: frames, pointer = 0, hit = 0, fault = 0, ref\_len, isFull = false.

- Step 2.2: Declare arrays:

2.2.1 buffer [frames] → Stores the reference string entered by the user.

2.2.2 reference [ref\_len] → Stores the reference string entered by the user.



2d.3 mem - layout [ref-len][frames] →  
Stores memory layout at each step.

Step 3: Input

Step 3.1: Input the number of frames (frames)

Step 3.2: Input the length of the reference string (ref-len)

Step 3.3: Input the reference string (Pages requested) into reference[].

Step 4: Process each page in the reference string

Step 4.1: For each page reference[i] in the reference string:

Step 4.1.1: Set search = -1

Step 4.1.2: Check if the page is already in memory (buffer[]).

Step 4.1.2.1: If yes, set search = j (index of page in memory) and increment hit

Step 4.1.3: If search == -1 (Page fault):

Step 4.1.3.1: If memory is not full (isFull = false):

- Place the page in buffer [pointer].
- Increment pointer.
- If pointer == frames, set isFull = true and reset pointer = 0.

Step 4.1.3.2: If memory is full (isFull = true), use Optimal.

Replacement:

Step 4.1.3.2.1: For each page in buffer[], find its next occurrence in reference [i+1...ref-len]

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

Step 4.1.3.2.2: If a page is not found in the future, assign a large value (e.g., 200) to indicate it won't be used.

Step 4.1.3.2.3: Select the page in memory with the farthest future use and replace it with the current page.

Step 4.1.3.3: Increment fault.

Step 4.1.4: Update `mem_layout[i][j]` with the current contents of `buffer[j]`.

Step 5: Display memory layout

Step 5.1: Print `mem_layout[j][i]` row-wise to show pages were loaded and replaced in memory.

Step 6: Calculate and display results

Step 6.1: Display total number of hits (hit)

Step 6.2: Display hit ratio ( $\text{hit} \div \text{ref\_len}$ ).

Step 6.3: Display total number of page faults (fault).

Step 7: End



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 27                                   | Practical No. : |

**//12. OPTIMAL PAGE REPLACEMENT**

```

import java.io.*;
import java.util.*;
public class Opt12{
    public static void main(String[] args) throws IOException
    {
        Scanner br = new Scanner(System.in);
        int frames, pointer = 0, hit = 0, fault = 0, ref_len;
        boolean isFull = false;
        int buffer[];
        int reference[];
        int mem_layout[][];

        System.out.println("Please enter the number of Frames: ");
        frames = br.nextInt();

        System.out.println("Please enter the length of the Reference string: ");
        ref_len = br.nextInt();

        reference = new int[ref_len];
        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
            buffer[j] = -1;

        System.out.println("Please enter the reference string: ");
        for(int i = 0; i < ref_len; i++)
        {
            reference[i] = br.nextInt();
        }
        System.out.println();
        for(int i = 0; i < ref_len; i++)
        {
            int search = -1;
            for(int j = 0; j < frames; j++)
            {
                if(buffer[j] == reference[i])
                {
                    search = j;
                    hit++;
                    break;
                }
            }
            if(search == -1)
            {
                if(isFull)
            
```

Date :

**GATE Educational Institutions**

Page No. :

28

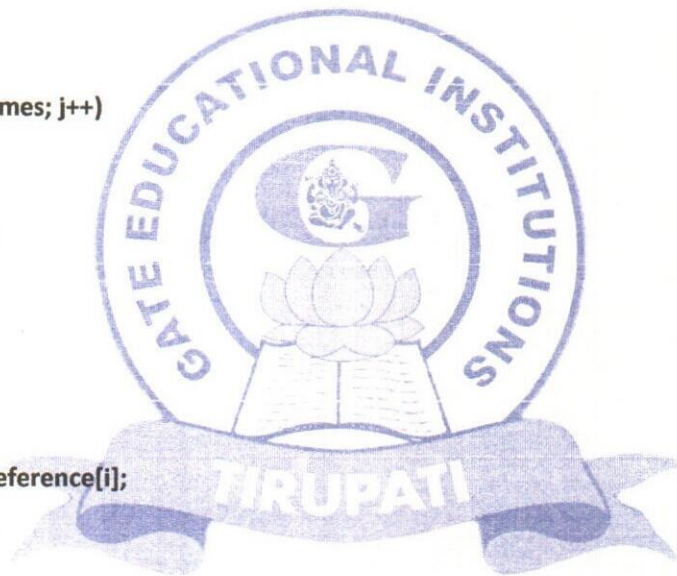
Practical No. :

```

int index[] = new int[frames];
boolean index_flag[] = new boolean[frames];
for(int j = i + 1; j < ref_len; j++)
{
    for(int k = 0; k < frames; k++)
    {
        if((reference[j] == buffer[k]) && (index_flag[k] == false))
        {
            index[k] = j;
            index_flag[k] = true;
            break;
        }
    }
}
int max = index[0];
pointer = 0;
if(max == 0)
    max = 200;
for(int j = 0; j < frames; j++)
{
    if(index[j] == 0)
        index[j] = 200;
    if(index[j] > max)
    {
        max = index[j];
        pointer = j;
    }
}
buffer[pointer] = reference[i];
fault++;
if(!isFull)
{
    pointer++;
    if(pointer == frames)
    {
        pointer = 0;
        isFull = true;
    }
}
for(int j = 0; j < frames; j++)
    mem_layout[i][j] = buffer[j];
}

for(int i = 0; i < frames; i++)
{
    for(int j = 0; j < ref_len; j++)

```



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 29                                   | Practical No. : |

```

System.out.printf("%3d ",mem_layout[j][i]);
System.out.println();
}

System.out.println("The number of Hits: " + hit);
System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
System.out.println("The number of Faults: " + fault);
}
}
/*OUTPUT:

```

D:\Java>java Opt12

Please enter the number of Frames:

3

Please enter the length of the Reference string:

20

Please enter the reference string:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7 7 7 2 2 2 2 2 2 2 2 2 2 2 2 2 7 7 7

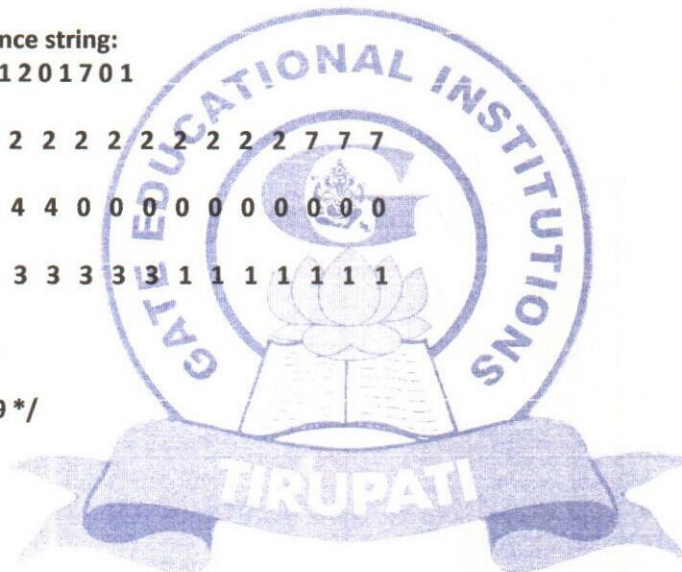
-1 0 0 0 0 0 0 4 4 4 0 0 0 0 0 0 0 0 0 0

-1 -1 1 1 1 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1

The number of Hits: 11

Hit Ratio: 0.55

The number of Faults: 9 \*/





|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

### Conclusion:

The program demonstrates the optimal Page Replacement Algorithm for efficient memory management. It shows how pages are loaded and replaced in memory to minimize page faults. From the output, we can see the number of hits and faults, and the memory is managed optimally to reduce unnecessary replacements.



|        |                                      |                       |
|--------|--------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :            |
|        |                                      | Practical No. :<br>13 |

**Aim :-** To implement the scan disk scheduling algorithm in Java and calculate the total head movement for a given set of disk I/O requests.

**Description :-** The scan disk scheduling algorithm also known as the elevator algorithm, the disk used in operating system to schedule disk I/O requests efficiently. In this algorithm, the disk arm starts from its current position and moves in a particular direction (towards one end of the disk). It services all requests in that direction until it reaches the end of the disk, then reverses direction and continues servicing requests.

This Java program takes the number of disk I/O requests, the initial head position, and list of requests as input. It then sorts the requests in ascending order and calculates the total head movement based on the scan algorithm. The program simulates the movement of the disk head, ensuring that all requests are serviced while minimizing unnecessary head movement. Finally, it outputs the total head movement.

**Algorithm :-**

step - 1 :- start

step - 2 :- Input data

\* step - 2.1 :- Input the number of processes / requests (a).



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

\* step-2.2 : Input the initial position of the disk head ( $h$ ).

\* step-2.3 : Input the disk I/O requests into an array  $ar[]$ .

\* step-2.4 : set  $ar[0] = 0$  to include the start of the disk.

step-3 :- Sort Requests

\* step-3.1 : Sort all elements in  $ar[]$  in ascending order.

step-4 :- calculate Total head movement.

\* step-4.1 : Initialize  $hm = 0$  (total head movement).

\* step-4.2 : Traverse through the sorted request array:

o step-4.2.1 : for each pair of consecutive requests, calculate the movement ;  

$$x = ar[c+1] - ar[c]$$

o step-4.2.2 : Add  $x$  to  $hm$ .

o step-4.2.3 : If the current request equals the initial head position

$(ar[c] == h)$ :

step-4.2.3.1 : subtract the last added movement ( $hm = hm - x$ )

step-4.2.3.2 : Add the movement from head to next request ( $hm = hm + ar[c+1]$ )



|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

\* step 4.3 : Repeat until all requests are serviced in SCAN order.

step - 5 :- Output results.

\* step - 5.1 : Display the total head movement (hm)

step - 6 :- End.



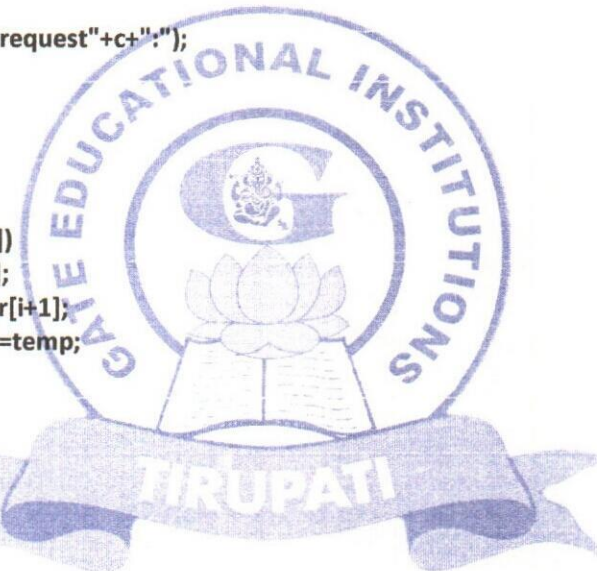
|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 30                                   | Practical No. : |

**//13.SCAN Disk Scheduling Algorithm**

```

import java.util.*;
public class Scan1
{
    public static void main(String args[])throws IOException
    {
        Scanner br=new Scanner(System.in);
        int a,x,c,h,i,hm=0,temp;
        int ar[]=new int[30];
        System.out.println("\nSCAN Scheduling");
        System.out.println("\nEnter no of process:");
        a=br.nextInt();
        System.out.println("\ninitial head movemnt:");
        h=br.nextInt();
        System.out.println("\n");
        ar[0]=0;
        for(c=1;c<=a;c++)
        { System.out.println("request"+c+":");
          ar[c]=br.nextInt();
        }
        ar[c]=h;
        for(c=0;c<a+1;c++)
        { for(i=0;i<a-c+1;i++)
          {if(ar[i]>ar[i+1])
            { temp=ar[i];
              ar[i]=ar[i+1];
              ar[i+1]=temp;
            }
          }
        }
        for(c=0;c<a+1;c++)
        { x=ar[c+1]-ar[c];
          hm=hm+x;
          if(ar[c]==h)
          { hm=hm-x;
            hm=hm+ar[c+1];
          }
          else;
        }
        System.out.println("\ntotal movements:"+hm);
    }
}

```



**/\*OUTPUT:**

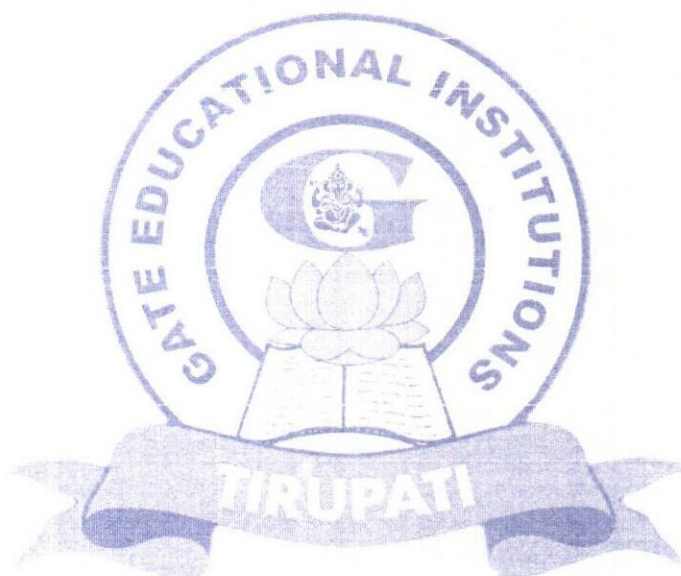
```

D:\Java>java Scan12
Enter no of process: 8
initial head movement: 53
request1:98
request2:183
request3:37

```

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 31                                   | Practical No. : |

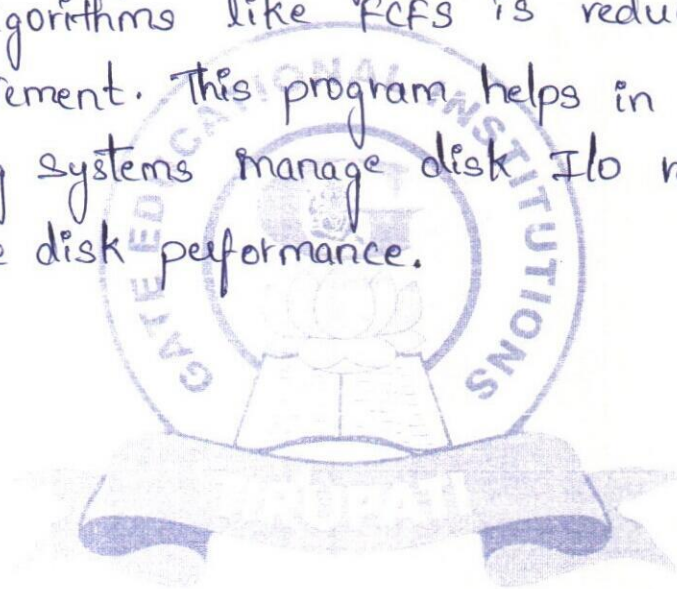
request4:122  
request5:14  
request6:124  
request7:65  
request8:67  
total movements:236  
\*/





|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

conclusion :- The program Successfully implements the scan disk algorithm scheduling. It demonstrates how the disk head moves in one direction, services all requests, and then reverse direction to continue processing, similar to an elevator. The total head movement is calculated efficiently, showing the advantage of scan over simpler algorithms like FCFS is reducing unnecessary disk arm movement. This program helps in understanding how operating systems manage disk I/O requests and optimize disk performance.



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>124 |

Aim:

To write a java program to implement the concept of Clook disk Scheduling

DESCRIPTION:

This program implements the C-look disk scheduling algorithm, which efficiently manages disk head movement to service requests. It takes the number of requests, initial head position, and request locations as input, sorts the requests, and moves the head jumps to the smallest request and continues in the same direction, servicing all remaining requests. The program calculates and displays the total head movement, showing how C-look reduces unnecessary movement and improves disk efficiency.

ALGORITHM:

Step 1: Start

Step 2: Input Data

Step 2.1: Input the number of requests / processes.

Step 2.2: Input the initial head position  $h$ .

Step 2.3: Input the disk I/O requests into an array  $ar[]$

Step 2.4: (Optional) Include the initial head position in  $ar[]$  for sorting purposes.

Step 3: Sort Requests

Step 3.1: Sort all elements in  $ar[]$  in ascending order.

Step 3.2: Identify the position/index of the initial head  $h$  in the sorted array.

Step 4: calculate Total Head movement

Step 4.1: Initialize  $hm = 0$

Step 4.2: move head from the initial position toward the higher end of the required array;

Step 4.2.1: For each consecutive request  $ar[c]$  to  $ar[c+1]$ , calculate the movement:  $x = |ar[c+1] - ar[c]|$ .



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>14. |

Step 4.2.2: Add  $x$  to  $hm$ .

Step 4.3: when the last request in this direction is reached, move the head circularly to the first request in the array.

Step 4.3.1: Calculate movement from last request to

First request :  $x = |ar[lat] - ar[0]|$ .

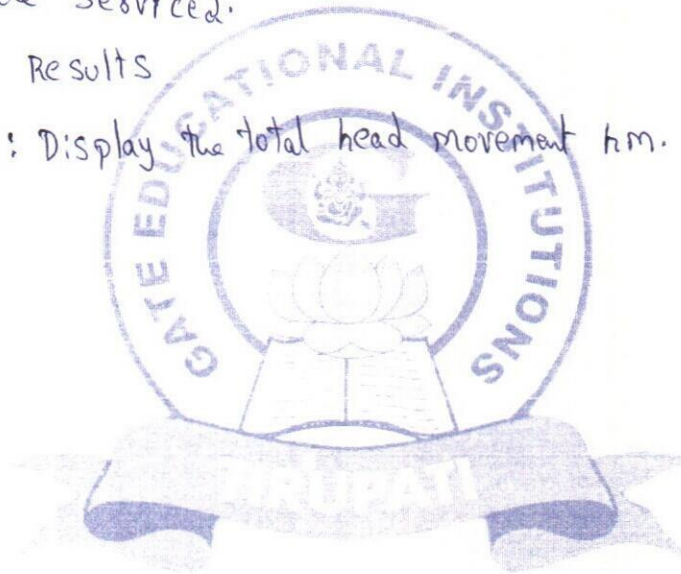
Step 4.3.2: Add  $x$  to  $hm$ .

Step 4.4 : Continue moving the head to service the remaining requests in order, adding each movement to  $hm$  until all requests are serviced.

Step 5: output Results

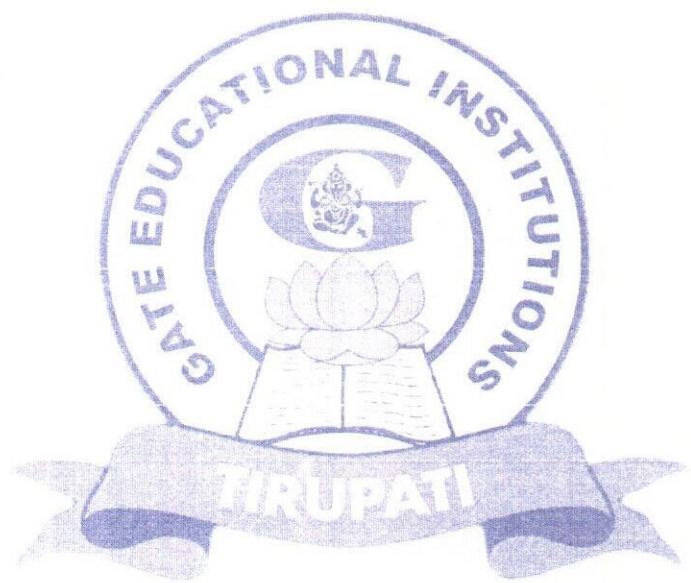
Step 5.1: Display the total head movement  $hm$ .

Step 6: End.





|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                      |                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-----------------|
| Date :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <b>GATE Educational Institutions</b> | Page No. :      |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 33                                   | Practical No. : |
| <pre> { head=ar[c]-ar[c+1]; if(head&lt;0) {      head=head*(-1); } if(c&lt;k-1) {System.out.println(" "); } hm=hm+head; System.out.println(" "+ar[c]+"-"+ar[c+1]+" "); }  System.out.println("="+hm); System.out.println("\n total head moves:"+hm); } } /*OUTPUT:  D:\Java&gt;java Clook  Clook disk Scheduling  How many process:8  initial head position:53 enter request request:1:98 request:2:183 request:3:37 request:4:122 request:5:14 request:6:124 request:7:65 request:8:67 Computation  53-65 + 65-67 + 67-98 + 98-122 + 122-124 + 124-183 + 183-14  +  14-37  =322  total head moves:322 */ </pre> |                                      |                 |



|        |                                      |                        |
|--------|--------------------------------------|------------------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :             |
|        |                                      | Practical No. :<br>14. |

### Conclusion :

The C-look disk scheduling algorithm efficiently minimize total head movement by servicing requests in one direction & then jumping circularly to the first request. This implementation accurately calculates the total head movements, demonstrating improved disk I/O performance & reduced seek time.





|        |                                                     |                       |
|--------|-----------------------------------------------------|-----------------------|
| Date : | <b>GATE Educational Institutions</b>                | Page No. :            |
|        | first come, first-served disk scheduling algorithm. | Practical No. :<br>15 |

**Aim:-** To calculate the total head movement for servicing disk I/O requests using the First-come, First-served (FCFS) disk scheduling algorithm in Java

**Description:-** The FCFS disk scheduling algorithm services disk I/O requests in the order they arrive. It is the simplest scheduling algorithm but may not always be the most efficient in terms of total head movement

In this program:

- \* The user inputs the number of disk cylinders and their request sequence
- \* The initial head position is provided
- \* The program calculates the distance the head moves to service each request sequentially.
- \* The total head movement is then displayed

This helps understand how the head travels across the disk and the efficiency of FCFS scheduling.

**Algorithm:-**

Step 1: Start

Step 2: Input Data

\* Step 2.1: Input the number of cylinders n.

\* Step 2.2: Input the cylinder numbers into an array c[]

\* Step 2.3: Input the initial head position hs



step 3: calculate Head Movement

\* step 3.1: Initialize total = 0 (total head movement)

\* step 3.2: Initialize flag = h<sub>s</sub> (current head position)

\* step 3.3: For each cylinder c<sub>i</sub> in the array:

\* step 3.3.1: calculate head movement hm<sub>i</sub> = |c<sub>i</sub> - flag|.

\* step 3.3.2: Add hm<sub>i</sub> to total

\* step 3.3.3: update flag = c<sub>i</sub> (move head to current request)

step 4: output Results

\* step 4.1: Display the total head movement total

step 5: end

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        | 34                                   | Practical No. : |

//15.FCFS Scheduling

```
import java.util.*;
class DiskF
{
    public static void main(String args[])
    {
        int n,i,hs,flag,total=0;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no of cylinders");
        n=sc.nextInt();
        int c[]=new int[n];
        int hm[]=new int[n];
        System.out.println("Enter the cylinder no.");
        for(i=0;i<n;i++)
        c[i]=sc.nextInt();
        System.out.println("enter head start");
        hs=sc.nextInt();
        flag=hs;
        for(i=0;i<n;i++)
        {
            if(c[i] > flag)
            hm[i]=c[i]-flag;
            else
            hm[i]=flag-c[i];
            flag=c[i];
        }
        for(i=0;i<n;i++)
        total=total+hm[i];
        System.out.println("The total no of cylinders traversed during disk movement is:"+ total);
    }
}
```

/\*OUTPUT:

```
D:\Java>java DiskF
Enter no of cylinders
8
Enter the cylinder no.
98 183 37 122 14 124 65 67
enter head start
53
The total no of cylinders traversed during disk movement is:640
*/
```

|        |                                      |                 |
|--------|--------------------------------------|-----------------|
| Date : | <b>GATE Educational Institutions</b> | Page No. :      |
|        |                                      | Practical No. : |

conclusion:-

The FCFS disk scheduling algorithm is simple and easy to implement servicing requests in the order they arrive. This program accurately calculates the total head movement for the given request sequence. Although FCFS is straight forward, it may lead to higher seek times compared to other algorithms like SSTF, SCAN, DISC-LOOK, especially when requests are widely scattered.

