**OPERATING SYSTEMS – EXPERIMENT 4**
**NAME: KRISHNA R. KAMATH SAP ID: 60004200126 DIVISION/BATCH: B/B2**
**DATE: 28/03/2022**

## AIM:

CPU scheduling algorithms like FCFS, SJF, Round Robin etc.

## PROBLEM STATEMENT:

1) Perform comparative assessment of various Scheduling Policies like FCFS, SJF (preemptive and non-preemptive), Priority (preemptive and non-preemptive) and Round Robin.
2) Take the input processes, their arrival time, burst time, priority, quantum from user.

## THEORY:

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.
Terms used

1. Arrival time: The process at which the process is given to CPU.
2. Burst time: The amount of time each process takes for execution.
3. Completion time: The time at which the process has completely executed.
4. Response time: The difference between the time when the process starts execution and the arrival time.
5. Turnaround time: The difference between the time when the process completes execution and the arrival time.
6. Waiting Time: The difference between the turnaround time and burst time.

### First Come First Serve (FCFS)

The processes are executed in the order in which they have been submitted.

### Shortest Job First (SJF)

The processes are checked at each arrival time and the process which have the shortest remaining burst time at that moment gets executed first. This is non-preemptive algorithm.

### Shortest Remaining Time Next (SRTN)

This is preemptive version of SJF. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Similar to SJF, if a new process that has arrived has shorter burst time than the current running process, the previous process is pre-empted and new process is allocated CPU.

### Priority Scheduling

Each process is assigned a priority and executable process with highest priority is allowed to run.

### Round Robin

Each process is assigned a time interval called its quantum (time slice).

If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed.

**CONCLUSION:**

Thus, from this experiment we learn how the different processing algorithms work.

**CODE and OUTPUT:**

1. *First Come First Serve Scheduling –*

```
import java.util.Scanner;
class FirCom
{
	public static void main(String args[])
	{
		Scanner sc = new Scanner(System.in);
		int n;
		System.out.print("Enter number of processes to be executed: ");
		n = sc.nextInt();
		int p[] = new int[n];			//Process id array
		int at[] = new int[n];			//Arrival Time array
		int bt[] = new int[n];			//Burst Time array
		int i, j, min1, min2, min3;
		for(i = 0; i < n; i++)
		{
			p[i] = i+1;
			System.out.print("Enter arrival time: ");
			at[i] = sc.nextInt();
			System.out.print("Enter burst time: ");
			bt[i] = sc.nextInt();
		}
		for(i = 0; i < n; i++)
		{
			min1 = at[i];
			min2 = bt[i];
			min3 = p[i];
			j = i-1;
			while(j >= 0 && at[j] > min1)
			{
				at[j+1] = at[j];
				bt[j+1] = bt[j];
				p[j+1] = p[j];
				j--;
			}
			at[j+1] = min1;
			bt[j+1] = min2;
			p[j+1] = min3;
```

```
                }
                int ct[] = new int[n];                  //Completion time array
                int cur_t = 0;                                  //Current Time
                int st[] = new int[n];                  //Start Time array
                for(i = 0; i < n; i++)
                {
                        ct[i] = cur_t + bt[i];
                        st[i] = cur_t;
                        cur_t = ct[i];
                }
                int  tat[] = new int[n];          //Turnaround Time array
                int wt[] = new int[n];                  //Waiting Time array
                float sum1 = 0, sum2 = 0;               //sum1 for tat and sum2 for wt
                System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");
                for(i = 0; i < n; i++)
                {
                        tat[i] = ct[i] - at[i];
                        wt[i] = tat[i] - bt[i];
                        System.out.println("Process
"+p[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat[i]+"\t"+wt[i]);
                        sum1 += tat[i];
                        sum2 += wt[i];
                }
                System.out.println("Average Turn Around Time: "+(sum1/n));
                System.out.println("Average Waiting Time: "+(sum2/n));
                System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion
Time");
                for(i = 0; i < n; i++)
                {
                        System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);
                }
        }
}
```

*OUTPUT –*

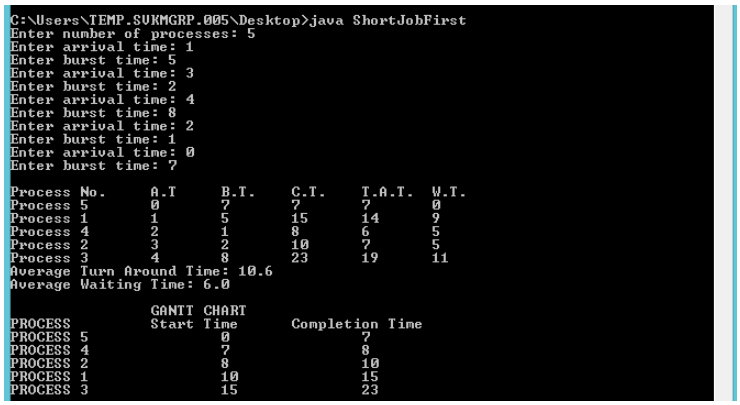
2. *Shortest Job First Scheduling (Non – Pre-emptive) –*

```
import java.util.Scanner;
class ShortJobFirst
{
	public static void main(String args[])
	{
		Scanner sc = new Scanner(System.in);
		int n;
		System.out.print("Enter number of processes: ");
		n = sc.nextInt();
		int p[] = new int[n];			//Process id array
		int at[] = new int[n];			//Arrival time array
		int bt[] = new int[n];			//Burst time array
		int flag[] = new int[n];		//To check if process is completed
		int i, j, min1, min2, min3;
		for(i = 0; i < n; i++)
		{
			p[i] = i+1;
			System.out.print("Enter arrival time: ");
			at[i] = sc.nextInt();
			System.out.print("Enter burst time: ");
			bt[i] = sc.nextInt();
			flag[i] = 0;
		}
		for(i = 0; i < n; i++)
		{
			min1 = at[i];
			min2 = bt[i];
			min3 = p[i];
			j = i-1;
			while(j >= 0 && at[j] > min1)
			{
				at[j+1] = at[j];
				bt[j+1] = bt[j];
				p[j+1] = p[j];
				j--;
			}
			at[j+1] = min1;
			bt[j+1] = min2;
			p[j+1] = min3;
		}
		int cur_t = 0;				//Current time
		int st[] = new int[n];		//Starting time of each process
		int ct[] = new int[n];		//completion time array
		int tot = 0;				//To count number of processes
completed
		int minbt = 1000;			//To store the shortest bt
```

```
		int c = 0;                                          //To track id of process to be
scheduled next
		while(tot < n)
		{
			for(i = 0; i < n; i++)
			{
				if((at[i] <= cur_t) && (flag[i] == 0) && (bt[i] <= minbt))
				{
					minbt = bt[i];
					c = i;
				}
			}
			ct[c] = cur_t + minbt;
			st[c] = cur_t;
			flag[c] = 1;
			cur_t = ct[c];
			tot++;
			minbt = 1000;                                   //reset so that bt values of
remaining processes are compared
		}
		int  tat[] = new int[n];            //Turnaround Time array
		int wt[] = new int[n];                      //Waiting Time array
		float sum1 = 0, sum2 = 0;                   //sum1 for tat and sum2 for wt
		System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");
		for(i = 0; i < n; i++)
		{
			tat[i] = ct[i] - at[i];
			wt[i] = tat[i] - bt[i];
			System.out.println("Process
"+p[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat[i]+"\t"+wt[i]);
			sum1 += tat[i];
			sum2 += wt[i];
		}
		System.out.println("Average Turn Around Time: "+(sum1/n));
		System.out.println("Average Waiting Time: "+(sum2/n));
		//To prepare Gantt Chart processes are soted according to start time
		for(i = 0; i < n; i++)
		{
			min1 = st[i];
			min2 = ct[i];
			min3 = p[i];
			j = i-1;
			while(j >= 0 && st[j] > min1)
			{
				st[j+1] = st[j];
				ct[j+1] = ct[j];
				p[j+1] = p[j];
```

```
                        j--;
                }
                st[j+1] = min1;
                ct[j+1] = min2;
                p[j+1] = min3;
        }
        System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion
Time");
        for(i = 0; i < n; i++)
        {
                System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);
        }
    }
}
```

*OUTPUT –*



```
C:\Users\TEMP.SVKMGRP.005\Desktop>java ShortJobFirst
Enter number of processes: 5
Enter arrival time: 1
Enter burst time: 5
Enter arrival time: 3
Enter burst time: 2
Enter arrival time: 4
Enter burst time: 8
Enter arrival time: 2
Enter burst time: 1
Enter arrival time: 0
Enter burst time: 7

Process No.     A.T     B.T.    C.T.    T.A.T.  W.T.
Process 5       0       7       7       7       0
Process 1       1       5       15      14      9
Process 4       2       1       8       6       5
Process 2       3       2       10      7       5
Process 3       4       8       23      19      11
Average Turn Around Time: 10.6
Average Waiting Time: 6.0

                GANTT CHART
PROCESS         Start Time      Completion Time
PROCESS 5               0               7
PROCESS 4               7               8
PROCESS 2               8               10
PROCESS 1               10              15
PROCESS 3               15              23
```

3. *Priority Scheduling (Pre-emptive) –*

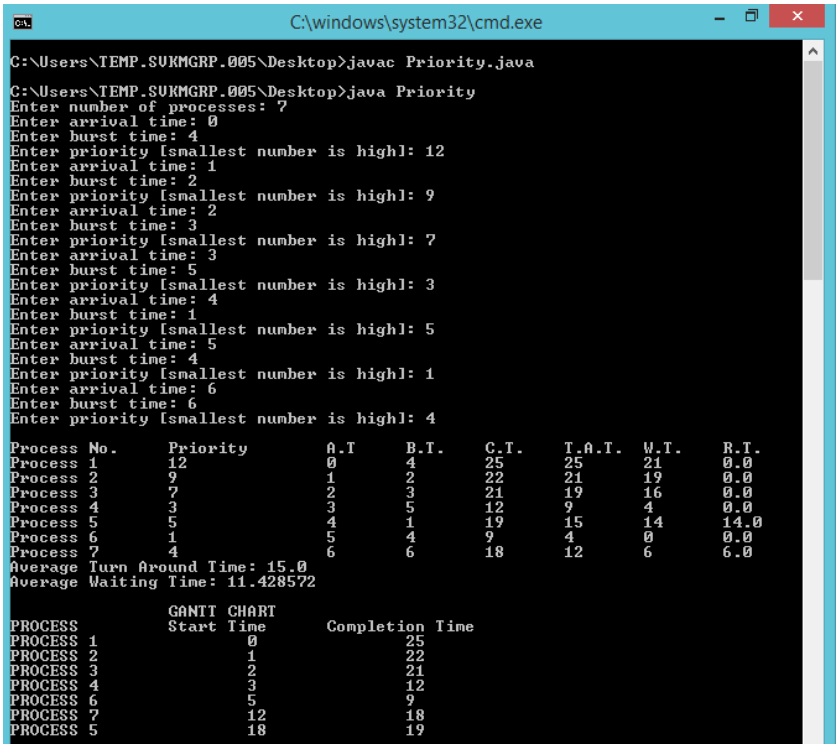
```
import java.util.Scanner;
class Priority
{
        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                int n;
                System.out.print("Enter number of processes: ");
                n = sc.nextInt();
                int p[] = new int[n];                   //Process id array
                int at[] = new int[n];                  //Arrival time array
                int bt[] = new int[n];                  //Burst time array
                int flag[] = new int[n];        //To check if process is completed
                int priority[] = new int[n];
                int i, j, min1, min2, min3, min4;
                int cur_t = 0;                                  //Current time
                int st[] = new int[n];                  //Starting time of each process
                int ct[] = new int[n];                  //completion time array
```

```
        int tot = 0;                                    //To count number of processes
completed
        int minpri = 1000;                              //To store the highest priority
        int c = 0;
        int b[] = new int[n];                           //To store a copy of burst time
array
        for(i = 0; i < n; i++)
        {
                p[i] = i+1;
                System.out.print("Enter arrival time: ");
                at[i] = sc.nextInt();
                System.out.print("Enter burst time: ");
                bt[i] = sc.nextInt();
                System.out.print("Enter priority [smallest number is high]: ");
                priority[i] = sc.nextInt();
                flag[i] = 0;
                st[i] = -1;
        }
        for(i = 0; i < n; i++)
        {
                min1 = at[i];
                min2 = bt[i];
                min3 = p[i];
                min4 = priority[i];
                j = i-1;
                while(j >= 0 && at[j] > min1)
                {
                        at[j+1] = at[j];
                        bt[j+1] = bt[j];
                        p[j+1] = p[j];
                        priority[j+1] = priority[j];
                        j--;
                }
                at[j+1] = min1;
                bt[j+1] = min2;
                p[j+1] = min3;
                priority[j+1] = min4;
        }
        for(i = 0; i < n; i++)
                b[i] = bt[i];
        while(tot < n)
        {
                for(i = 0; i < n; i++)
                {
                        if((priority[i] <= minpri) && (at[i] <= cur_t) && (flag[i] == 0))
                        {
                                minpri = priority[i];
```

```
                                        c = i;
                                }
                        }
                        if(st[c] == -1)
                        {
                                st[c] = cur_t;
                                ct[c] = cur_t;
                        }
                        b[c]--;
                        cur_t++;
                        ct[c]++;
                        for(i = 0; i < n; i++)
                        {
                                if((c != i) && (st[i] != -1) && (flag[i] == 0))
                                        ct[i]++;
                        }
                        if(b[c] == 0)
                        {
                                flag[c] = 1;
                                tot++;
                        }
                        minpri = 1000;
                }
                int  tat = 0;                           //Turnaround Time
                int wt = 0;                                     //Waiting Time array
                float sum1 = 0, sum2 = 0, rt = 0;                       //sum1 for tat and sum2
for wt and response time
                System.out.println("\nProcess
No.\tPriority\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.\tR.T.");
                for(i = 0; i < n; i++)
                {
                        tat = ct[i] - at[i];
                        wt = tat - bt[i];
                        rt = st[i] - at[i];
                        System.out.println("Process
"+p[i]+"\t"+priority[i]+"\t\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat+"\t"+wt+"\t"+rt);
                        sum1 += tat;
                        sum2 += wt;
                }
                System.out.println("Average Turn Around Time: "+(sum1/n));
                System.out.println("Average Waiting Time: "+(sum2/n));
                //To prepare Gantt Chart processes are soted according to start time
                for(i = 0; i < n; i++)
                {
                        min1 = st[i];
                        min2 = ct[i];
                        min3 = p[i];
```

```
                    j = i-1;
                    while(j >= 0 && st[j] > min1)
                    {
                            st[j+1] = st[j];
                            ct[j+1] = ct[j];
                            p[j+1] = p[j];
                            j--;
                    }
                    st[j+1] = min1;
                    ct[j+1] = min2;
                    p[j+1] = min3;
            }
            System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion
Time");
            for(i = 0; i < n; i++)
            {
                    System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);
            }
    }
}
```

*OUTPUT –*




4. *Round Robin Scheduling –*

```
import java.util.Scanner;
class Queue
{
```

```
	Node front, rear;
	int queueSize;
	class Node
	{
		int data;
		Node next;
	}
	Queue()
	{
		front = null;
		rear = null;
		queueSize = 0;
	}
	boolean isEmpty()
	{
		return (queueSize == 0);
	}
	int dequeue()
	{
		int data = front.data;
		front = front.next;
		if (isEmpty())
		{
			rear = null;
		}
		queueSize--;
		return data;
	}
	void enqueue(int data)
	{
		Node oldRear = rear;
		rear = new Node();
		rear.data = data;
		rear.next = null;
		if (isEmpty())
		{
			front = rear;
		}
		else
		{
			oldRear.next = rear;
		}
		queueSize++;
	}
}

class RoundRobin
```

```
{
	public static void main(String args[])
	{
		Scanner sc = new Scanner(System.in);
		int n, tq;
		System.out.print("Enter number of processes: ");
		n = sc.nextInt();
		System.out.print("Enter time quantum: ");
		tq = sc.nextInt();
		Queue q = new Queue();
		int p[] = new int[n];			//Process id array
		int at[] = new int[n];			//Arrival time array
		int bt[] = new int[n];			//Burst time array
		int flag[] = new int[n];		//To check if process is completed
		int f[] = new int[n];			//To check if process has arrived
		int i, j, min1, min2, min3;
		int cur_t = 0;					//Current time
		int st[] = new int[n];			//Starting time of each process
		int ct[] = new int[n];			//completion time array
		int tot = 0;					//To count number of processes
completed
		int c = 0;					//To track id of process to be
scheduled next
		int b[] = new int[n];			//Copy of Burst time array
		for(i = 0; i < n; i++)
		{
			p[i] = i+1;
			System.out.print("Enter arrival time: ");
			at[i] = sc.nextInt();
			System.out.print("Enter burst time: ");
			bt[i] = sc.nextInt();
			flag[i] = 0;
			f[i] = 0;
			st[i] = -1;
		}
		for(i = 0; i < n; i++)
		{
			min1 = at[i];
			min2 = bt[i];
			min3 = p[i];
			j = i-1;
			while(j >= 0 && at[j] > min1)
			{
				at[j+1] = at[j];
				bt[j+1] = bt[j];
				p[j+1] = p[j];
				j--;
```

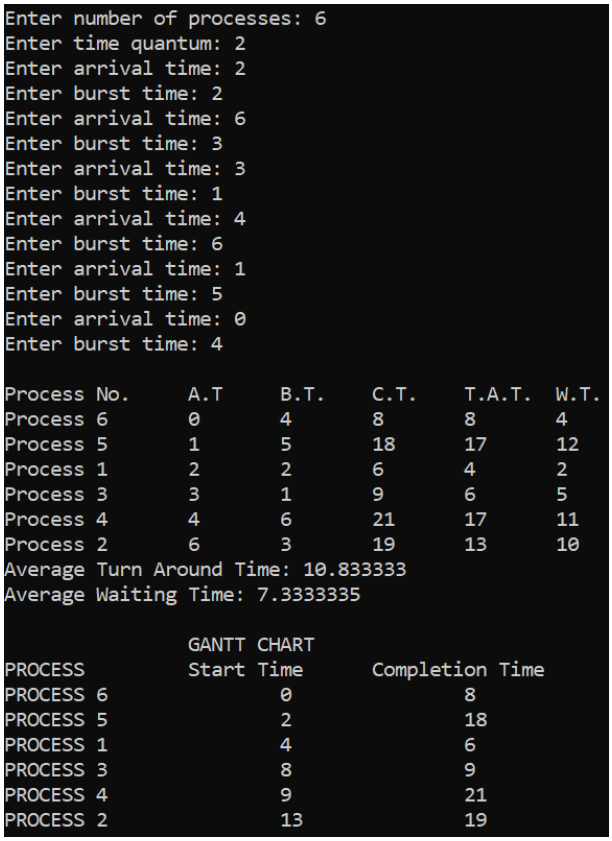
```
		}
		at[j+1] = min1;
		bt[j+1] = min2;
		p[j+1] = min3;
	}
	for(i = 0; i < n; i++)
		b[i] = bt[i];
	while(tot < n)
	{
		for(i = 0; i < n; i++)
		{
			if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
			{
				q.enqueue(i);
				f[i] = 1;
			}
		}
		c = q.dequeue();
		if(st[c] == -1)
			st[c] = cur_t;
		ct[c] = cur_t;
		if(b[c] > tq)
		{
			ct[c] += tq;
			b[c] -= tq;
			cur_t += tq;
			for(i = 0; i < n; i++)
			{
				if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
				{
					q.enqueue(i);
					f[i] = 1;
				}
			}
			if(b[c] > 0)
				q.enqueue(c);
		}
		else if(b[c] <= tq)
		{
			ct[c] += b[c];
			cur_t += b[c];
			b[c] = 0;
			for(i = 0; i < n; i++)
			{
				if((f[i] == 0) && (flag[i] == 0) && (at[i] <= cur_t))
				{
					q.enqueue(i);
```

```
                                        f[i] = 1;
                                    }
                                }
                            }
                            if(b[c] == 0)
                            {
                                flag[c] = 1;
                                tot++;
                            }
                        }
                        int  tat = 0;                                   //Turnaround Time
                        int wt = 0;                                             //Waiting Time
array
                        float sum1 = 0, sum2 = 0;                       //sum1 for tat and sum2 for wt
                        System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");
                        for(i = 0; i < n; i++)
                        {
                            tat = ct[i] - at[i];
                            wt = tat - bt[i];
                            System.out.println("Process
"+p[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+tat+"\t"+wt);
                            sum1 += tat;
                            sum2 += wt;
                        }
                        System.out.println("Average Turn Around Time: "+(sum1/n));
                        System.out.println("Average Waiting Time: "+(sum2/n));

                        //To prepare Gantt Chart processes are soted according to start time
                        for(i = 0; i < n; i++)
                        {
                            min1 = st[i];
                            min2 = ct[i];
                            min3 = p[i];
                            j = i-1;
                            while(j >= 0 && st[j] > min1)
                            {
                                st[j+1] = st[j];
                                ct[j+1] = ct[j];
                                p[j+1] = p[j];
                                j--;
                            }
                            st[j+1] = min1;
                            ct[j+1] = min2;
                            p[j+1] = min3;
                        }
                        System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion
Time");
```

```
                for(i = 0; i < n; i++)
                {
                        System.out.println("PROCESS "+p[i]+"\t\t"+st[i]+"\t\t"+ct[i]);
                }
        }
}
```

*OUTPUT –*



```
Enter number of processes: 6
Enter time quantum: 2
Enter arrival time: 2
Enter burst time: 2
Enter arrival time: 6
Enter burst time: 3
Enter arrival time: 3
Enter burst time: 1
Enter arrival time: 4
Enter burst time: 6
Enter arrival time: 1
Enter burst time: 5
Enter arrival time: 0
Enter burst time: 4

Process No.     A.T     B.T.    C.T.    T.A.T.  W.T.
Process 6       0       4       8       8       4
Process 5       1       5       18      17      12
Process 1       2       2       6       4       2
Process 3       3       1       9       6       5
Process 4       4       6       21      17      11
Process 2       6       3       19      13      10
Average Turn Around Time: 10.833333
Average Waiting Time: 7.3333335

                GANTT CHART
PROCESS         Start Time      Completion Time
PROCESS 6               0               8
PROCESS 5               2               18
PROCESS 1               4               6
PROCESS 3               8               9
PROCESS 4               9               21
PROCESS 2               13              19
```