NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

# COMPILATION OF LABORATORY ACTIVITIES

| | | | |
|---|---|---|---|
| **Course Code/Subject** | CCOPSYSL | **Term/AY** | **1st / 25-26** |
| **Section** | COM232 | **Date** | October 7, 2025 |
| **Group # \| Leader Name** | TALOSIG, JAY ARRE P. | | |
| **CCOPSYSL GitHub Repo** | https://github.com/flexycode/CCOPSYSL | | |

## LISTS OF ACTIVITIES

| No. | Title |
|---|---|
| **1** | First Come, First Served Algorithm |
| **2** | Shortest Job First (SRTF) Algorithm |
| **3** | Round Robin Algorithm |
| **4** | Priority Algorithm |
| **5** | Highest Response Ratio Next Algorithm |

This form must be properly filled up before submission.

Prepared by:

Talosig, Jay Arre P.

Signature over group leader name

| ACTIVITY NO | # 1 |
|---|---|
| TITLE | First Come, First Served Algorithm |

# ACTIVITY 1

## FCFS

SAMPLE OUTPUT FOR CHECKING PURPOSE ONLY

**Activity 1: First Come, First Serve Algorithm**

| Process | AT | BT | ST | CT | TAT | WT |
|---|---|---|---|---|---|---|
| P1 | 0 | 4 | 0 | 4 | 4 | 0 |
| P2 | 1 | 3 | 4 | 7 | 6 | 3 |
| P3 | 2 | 1 | 7 | 8 | 6 | 5 |
| P4 | 3 | 2 | 8 | 10 | 7 | 5 |
| P5 | 5 | 4 | 10 | 14 | 9 | 5 |

AVERAGE TAT = 6.4
AVERAGE WT = 3.6

**Gannt Chart:** | P1 | P2 | P3 | P4 | P5 |
(Example) 0 4 7 8 10 14

## SOURCE CODE

```
import java.util.*;
import java.util.InputMismatchException;

public class SNFN {

    static class Process {
        String name;
        int arrivalTime;
        int burstTime;
        int startTime;
        int completionTime;
        int turnaroundTime;
        int waitingTime;

        Process(String name, int arrivalTime, int burstTime) {
            this.name = name;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
        }
    }

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

        do {
            System.out.println("=== FCFS SCHEDULING ALGORITHM ===\n");

            int numProcesses = getNumberOfProcesses(scanner);
            Process[] processes = new Process[numProcesses];

            inputProcessDetails(scanner, processes, numProcesses);
            sortProcessesByArrivalTime(processes);
            calculateSchedulingMetrics(processes);
            displayResults(processes);
            displayGanttChart(processes);

            System.out.print("\nDo you want to try again? (yes/no): ");
            continueChoice = scanner.next().toLowerCase();
            System.out.println();

        } while (continueChoice.equals("yes") || continueChoice.equals("y"));

        System.out.println("Thank you for using FCFS Scheduling Algorithm!");
        scanner.close();
    }

    private static int getNumberOfProcesses(Scanner scanner) {
        int numProcesses = 0; // Initialize to 0
```

## SOURCE CODE

```
        do {
            try {
                System.out.print("Enter number of processes (3-5): ");
                numProcesses = scanner.nextInt();

                if (numProcesses < 3 || numProcesses > 5) {
                    System.out.println("Error: Please enter a number between 3 and 5.");
                }
            } catch (InputMismatchException e) {

                // This block runs if the user enters something that is not an integer
                System.out.println("Error: Invalid input. Please enter a valid number.");
                scanner.next(); // Important: Clears the invalid input from the scanner.
                numProcesses = 0; // Reset to ensure the loop continues.
            }
        } while (numProcesses < 3 || numProcesses > 5);

        return numProcesses;
    }

    private static int getValidIntegerInput(Scanner scanner, String prompt) {
        while (true) { // Loop indefinitely until a valid integer is returned.
            try {
                System.out.print(prompt);
                return scanner.nextInt(); // If successful, return the integer and exit the loop.
            } catch (InputMismatchException e) {
                System.out.println("Error: Invalid input. Please enter a whole number.");
                scanner.next(); // Clear the invalid input to prevent an infinite loop.
            }
        }
    }

    private static void inputProcessDetails(Scanner scanner, Process[] processes, int
numProcesses) {
        System.out.println("\nEnter process details:");

        for (int i = 0; i < numProcesses; i++) {
            String processName = "P" + (i + 1);

            int arrivalTime = getValidIntegerInput(scanner, "Enter Arrival Time for " + processName +
": ");
            int burstTime = getValidIntegerInput(scanner, "Enter Burst Time for " + processName + ":
");

            processes[i] = new Process(processName, arrivalTime, burstTime);
        }
    }

    private static void sortProcessesByArrivalTime(Process[] processes) {
        Arrays.sort(processes, new Comparator<Process>() {
```

## SOURCE CODE

```
        @Override
        public int compare(Process p1, Process p2) {
            return Integer.compare(p1.arrivalTime, p2.arrivalTime);
        }
    });

    for (int i = 0; i < processes.length; i++) {
        processes[i].name = "P" + (i + 1);
    }
}

private static void calculateSchedulingMetrics(Process[] processes) {
    int currentTime = 0;

    for (int i = 0; i < processes.length; i++) {
        Process process = processes[i];

      if (currentTime < process.arrivalTime) {
            currentTime = process.arrivalTime;
        }

        process.startTime = currentTime;
        process.completionTime = process.startTime + process.burstTime;
        process.turnaroundTime = process.completionTime - process.arrivalTime;
        process.waitingTime = process.turnaroundTime - process.burstTime;

        currentTime = process.completionTime;
    }
}

private static void displayResults(Process[] processes) {
    System.out.println("\n=== SCHEDULING RESULTS ===");
    System.out.printf("%-8s %-12s %-10s %-15s %-15s %-12s%n",
            "Process", "Arrival Time", "Burst Time", "Completion Time",
            "Turnaround Time", "Waiting Time");
    System.out.println("----------------------------------------------------------------------------------");

    double totalTurnaroundTime = 0;
    double totalWaitingTime = 0;

    for (Process process : processes) {
        System.out.printf("%-8s %-12d %-10d %-15d %-15d %-12d%n",
                process.name, process.arrivalTime, process.burstTime,
                process.completionTime, process.turnaroundTime, process.waitingTime);

        totalTurnaroundTime += process.turnaroundTime;
        totalWaitingTime += process.waitingTime;
    }

    double avgTurnaroundTime = totalTurnaroundTime / processes.length;
```

## SOURCE CODE

```
        double avgWaitingTime = totalWaitingTime / processes.length;

        System.out.println("-------------------------------------------------------------------------------");
        System.out.printf("Average Turnaround Time: %.2f%n", avgTurnaroundTime);
        System.out.printf("Average Waiting Time: %.2f%n", avgWaitingTime);
    }

    private static void displayGanttChart(Process[] processes) {
        System.out.println("\n=== GANTT CHART ===");

        System.out.print("|");
        for (Process process : processes) {
            System.out.printf(" %s |", process.name);
        }

        System.out.println();

        System.out.print(processes[0].startTime);
        for (Process process : processes) {
            int digits = String.valueOf(process.completionTime).length();
            int spaces = 4 - digits;
            for (int i = 0; i < spaces; i++) {
                System.out.print(" ");
            }
            System.out.print(process.completionTime);
        }
        System.out.println();
    }
}
```

## SCREEN SHOT/SAMPLE OUTPUT

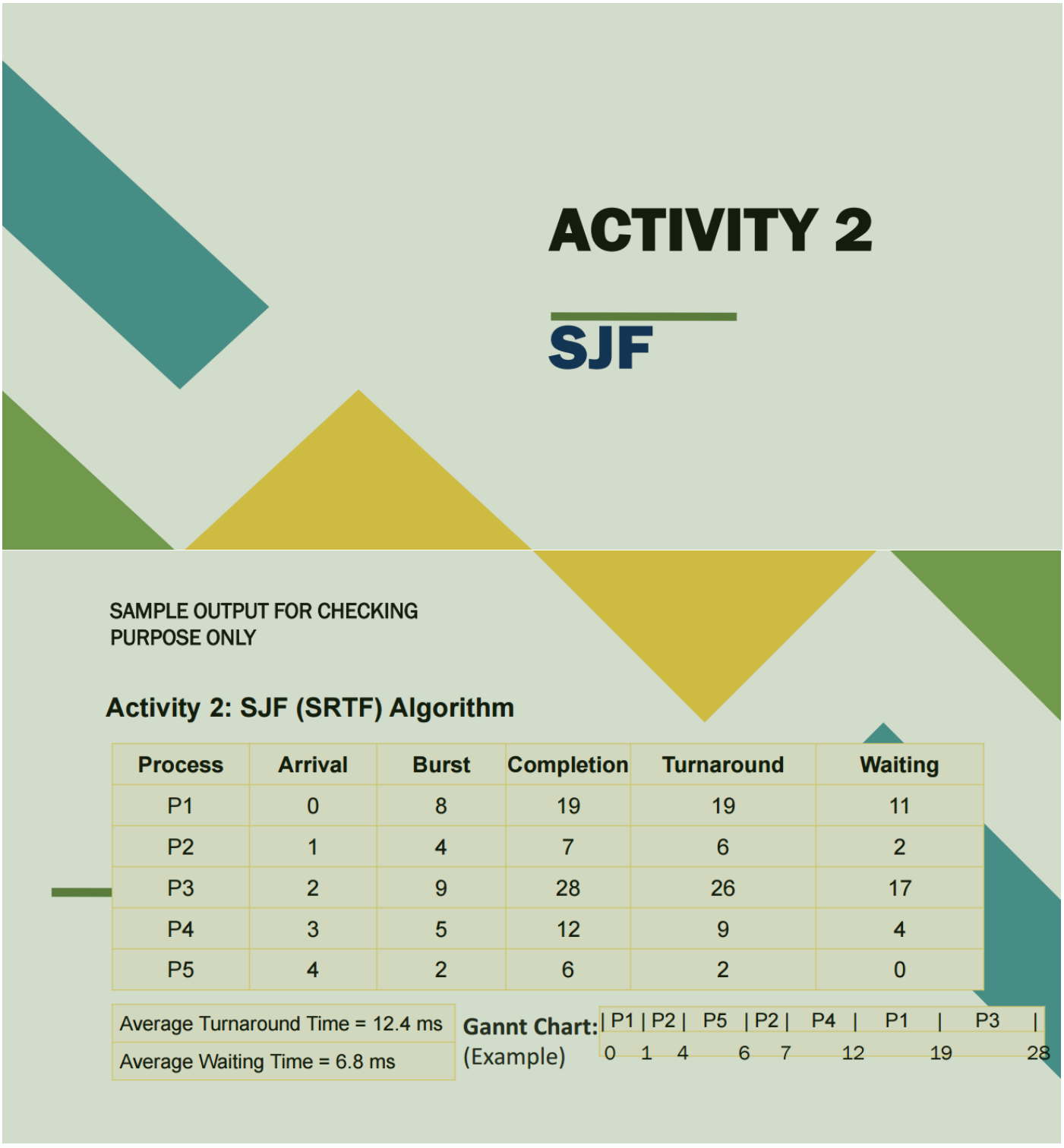
```
Enter number of processes (3-5): 5

Enter process details:
Enter Arrival Time for P1: 0
Enter Burst Time for P1: 4
Enter Arrival Time for P2: 1
Enter Burst Time for P2: 3
Enter Arrival Time for P3: 2
Enter Burst Time for P3: 1
Enter Arrival Time for P4: 3
Enter Burst Time for P4: 2
Enter Arrival Time for P5: 5
Enter Burst Time for P5: 4

=== SCHEDULING RESULTS ===
Process  Arrival Time Burst Time Completion Time Turnaround Time Waiting Time
--------------------------------------------------------------------------------
P1       0            4          4               4               0
P2       1            3          7               6               3
P3       2            1          8               6               5
P4       3            2          10              7               5
P5       5            4          14              9               5
--------------------------------------------------------------------------------
```


```
=== SCHEDULING RESULTS ===
Process  Arrival Time Burst Time Completion Time Turnaround Time Waiting Time
--------------------------------------------------------------------------------
P1       0            4          4               4               0
P2       1            3          7               6               3
P3       2            1          8               6               5
P4       3            2          10              7               5
P5       5            4          14              9               5
--------------------------------------------------------------------------------
```


```
Average Turnaround Time: 6.40
Average Waiting Time: 3.60

=== GANTT CHART ===
| P1 | P2 | P3 | P4 | P5 |
0  4  7  8  10  14

Do you want to try again? (yes/no): yes

=== FCFS SCHEDULING ALGORITHM ===
```

```
=== FCFS SCHEDULING ALGORITHM ===

Enter number of processes (3-5): 4

Enter process details:
Enter Arrival Time for P1: 3
Enter Burst Time for P1: 4
Enter Arrival Time for P2: 1
Enter Burst Time for P2: 3
Enter Arrival Time for P3: 4
Enter Burst Time for P3: 2
Enter Arrival Time for P4: 2
Enter Burst Time for P4: 1

=== SCHEDULING RESULTS ===
Process  Arrival Time Burst Time Completion Time Turnaround Time Waiting Time
--------------------------------------------------------------------------------
P1       1            3          4               3               0
P2       2            1          5               3               2
P3       3            4          9               6               2
P4       4            2          11              7               5
--------------------------------------------------------------------------------
Enter Burst Time for P2: 3
Enter Arrival Time for P3: 4
Enter Burst Time for P3: 2
Enter Arrival Time for P4: 2
Enter Burst Time for P4: 1

=== SCHEDULING RESULTS ===
Process  Arrival Time Burst Time Completion Time Turnaround Time Waiting Time
--------------------------------------------------------------------------------
P1       1            3          4               3               0
P2       2            1          5               3               2
P3       3            4          9               6               2
P4       4            2          11              7               5
--------------------------------------------------------------------------------
Average Turnaround Time: 4.75
Average Waiting Time: 2.25

=== GANTT CHART ===
| P1 | P2 | P3 | P4 |
1   4   5   9  11

Do you want to try again? (yes/no):
```

| | |
|---|---|
| **ACTIVITY NO** | **# 2** |
| **TITLE** | Shortest Job First (SRTF) Algorithm |



## SOURCE CODE

```
import java.util.*;

// Process class to represent each process with its attributes
class Process {
    int processId;                  // Unique identifier for the process
    int arrivalTime;                // Time at which process arrives in ready queue
    int burstTime;                  // CPU burst time required by the process
    int remainingTime;                // Remaining burst time (used in preemptive)
    int completionTime;       // Time at which process completes execution
    int turnaroundTime;       // Total time from arrival to completion
    int waitingTime;                // Time spent waiting in ready queue

    public Process(int processId, int arrivalTime, int burstTime) {
        this.processId = processId;
        this.arrivalTime = arrivalTime;
        this.burstTime = burstTime;
        this.remainingTime = burstTime;
        this.completionTime = 0;
        this.turnaroundTime = 0;
        this.waitingTime = 0;
    }
}

/**
 * Main class implementing SJF Scheduling Algorithm
 */
public class SNFN {

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

        System.out.println("=====================================");
        System.out.println("   SJF SCHEDULING ALGORITHM");
        System.out.println("   Operating Systems Lab - ACT2");
        System.out.println("=====================================\n");

        System.out.print("Enter the number of processes: ");
        int n = scanner.nextInt();

        Process[] processes = new Process[n];

        System.out.println("\nEnter process details:");
        for (int i = 0; i < n; i++) {
            System.out.println("\nProcess " + (i + 1) + ":");
            System.out.print("  Arrival Time: ");
            int arrivalTime = scanner.nextInt();
            System.out.print("  Burst Time: ");
            int burstTime = scanner.nextInt();

            processes[i] = new Process(i + 1, arrivalTime, burstTime);
        }
```

## SOURCE CODE

```
    // Menu for selecting scheduling type
    System.out.println("\n====================================");
    System.out.println("Select Scheduling Type:");
    System.out.println("1. Non-Preemptive SJF");
    System.out.println("2. Preemptive SJF (SRTF)");
    System.out.print("Enter your choice (1 or 2): ");
    int choice = scanner.nextInt();

    switch (choice) {
        case 1:
            nonPreemptiveSJF(processes);
            break;
        case 2:
            preemptiveSJF(processes);
            break;
        default:
            System.out.println("Invalid choice! Running Non-Preemptive SJF by default.");
            nonPreemptiveSJF(processes);
    }

    scanner.close();
}

public static void nonPreemptiveSJF(Process[] processes) {
    int n = processes.length;
    boolean[] completed = new boolean[n];
    int currentTime = 0;
    int completedCount = 0;

    System.out.println("\n====================================");
    System.out.println("NON-PREEMPTIVE SJF SCHEDULING");
    System.out.println("====================================");
    System.out.println("\nGantt Chart:");
    System.out.print("| ");

    while (completedCount < n) {
        int shortestIndex = -1;
        int shortestBurst = Integer.MAX_VALUE;

        for (int i = 0; i < n; i++) {
            if (!completed[i] &&
                    processes[i].arrivalTime <= currentTime &&
                    processes[i].burstTime < shortestBurst) {
                shortestBurst = processes[i].burstTime;
                shortestIndex = i;
            }
        }

        if (shortestIndex == -1) {
            currentTime++;
```

## SOURCE CODE

```
            continue;
        }

        Process currentProcess = processes[shortestIndex];
        System.out.print("P" + currentProcess.processId + " | ");

        currentTime += currentProcess.burstTime;
        currentProcess.completionTime = currentTime;
        currentProcess.turnaroundTime = currentProcess.completionTime -
currentProcess.arrivalTime;
        currentProcess.waitingTime = currentProcess.turnaroundTime - currentProcess.burstTime;

        completed[shortestIndex] = true;
        completedCount++;
    }

    System.out.println("\n");

    // Display results
    displayResults(processes);
}

public static void preemptiveSJF(Process[] processes) {
    int n = processes.length;
    int currentTime = 0;
    int completedCount = 0;
    int prevProcess = -1;

    Process[] processesCopy = new Process[n];
    for (int i = 0; i < n; i++) {
        processesCopy[i] = new Process(
                processes[i].processId,
                processes[i].arrivalTime,
                processes[i].burstTime
        );
    }

    System.out.println("\n======================================");
    System.out.println("PREEMPTIVE SJF SCHEDULING");
    System.out.println("======================================");
    System.out.println("\nExecution Timeline:");

    // Continue until all processes complete
    while (completedCount < n) {
        int shortestIndex = -1;
        int shortestRemaining = Integer.MAX_VALUE;

        // Find process with shortest remaining time
        for (int i = 0; i < n; i++) {
            if (processesCopy[i].arrivalTime <= currentTime &&
```

## SOURCE CODE

```
                processesCopy[i].remainingTime > 0 &&
                processesCopy[i].remainingTime < shortestRemaining) {
            shortestRemaining = processesCopy[i].remainingTime;
            shortestIndex = i;
        }
    }

    // If no process has arrived, advance time
    if (shortestIndex == -1) {
        currentTime++;
        continue;
    }

    // Print process switch if different from previous
    if (prevProcess != shortestIndex) {
        System.out.println("Time " + currentTime + ": Process P" +
                processesCopy[shortestIndex].processId + " starts/resumes");
        prevProcess = shortestIndex;
    }

    // Execute current process for one time unit
    processesCopy[shortestIndex].remainingTime--;
    currentTime++;

    // Check if process completed
    if (processesCopy[shortestIndex].remainingTime == 0) {
        completedCount++;
        processesCopy[shortestIndex].completionTime = currentTime;
        processesCopy[shortestIndex].turnaroundTime =
                processesCopy[shortestIndex].completionTime -
                        processesCopy[shortestIndex].arrivalTime;
        processesCopy[shortestIndex].waitingTime =
                processesCopy[shortestIndex].turnaroundTime -
                        processesCopy[shortestIndex].burstTime;

        System.out.println("Time " + currentTime + ": Process P" +
                processesCopy[shortestIndex].processId + " completed");
    }
  }

  System.out.println();

  // Display results
  displayResults(processesCopy);
}

public static void displayResults(Process[] processes) {
  System.out.println("====================================");
  System.out.println("SCHEDULING RESULTS");
  System.out.println("====================================");
```

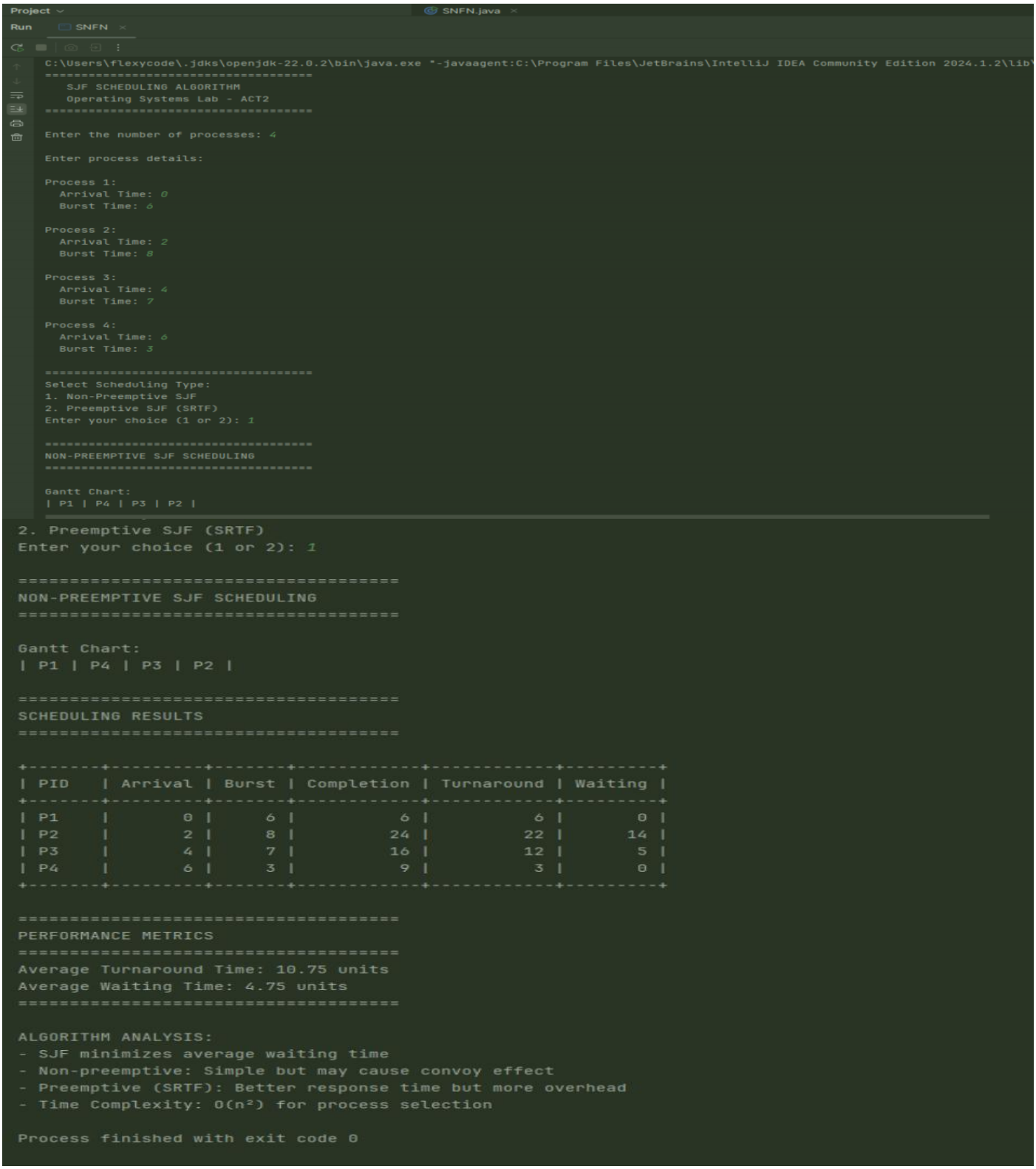
## SOURCE CODE

```
        // Table header
        System.out.println("\n+-------+---------+-------+------------+------------+---------+");
        System.out.println("| PID   | Arrival | Burst | Completion | Turnaround | Waiting |");
        System.out.println("+-------+---------+-------+------------+------------+---------+");

        double totalTurnaroundTime = 0;
        double totalWaitingTime = 0;

        // Display each process details
        for (Process p : processes) {
            System.out.printf("| P%-4d | %7d | %5d | %10d | %10d | %7d |\n",
                    p.processId, p.arrivalTime, p.burstTime,
                    p.completionTime, p.turnaroundTime, p.waitingTime);

            totalTurnaroundTime += p.turnaroundTime;
            totalWaitingTime += p.waitingTime;
        }

        System.out.println("+-------+---------+-------+------------+------------+---------+");

        // Calculate and display averages
        double avgTurnaroundTime = totalTurnaroundTime / processes.length;
        double avgWaitingTime = totalWaitingTime / processes.length;

        System.out.println("\n========================================");
        System.out.println("PERFORMANCE METRICS");
        System.out.println("========================================");
        System.out.printf("Average Turnaround Time: %.2f units\n", avgTurnaroundTime);
        System.out.printf("Average Waiting Time: %.2f units\n", avgWaitingTime);
        System.out.println("========================================");

        // Additional analysis
        System.out.println("\nALGORITHM ANALYSIS:");
        System.out.println("- SJF minimizes average waiting time");
        System.out.println("- Non-preemptive: Simple but may cause convoy effect");
        System.out.println("- Preemptive (SRTF): Better response time but more overhead");
        System.out.println("- Time Complexity: O(n²) for process selection");
    }
}
```

NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

## SCREEN SHOT/SAMPLE OUTPUT

Number of processes: 4
Process details: (Arrival Time, Burst Time)
Selection of Scheduling Type: Non-Preemptive SJF
Choices: Non-Preemptive SJF or Preemptive SJF (SRTF) – Shortest Remaining Time First



```
Project
Run  SNFN
C:\Users\flexycode\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1.2\lib
====================================
   SJF SCHEDULING ALGORITHM
   Operating Systems Lab - ACT2
====================================

Enter the number of processes: 4

Enter process details:

Process 1:
  Arrival Time: 0
  Burst Time: 6

Process 2:
  Arrival Time: 2
  Burst Time: 8

Process 3:
  Arrival Time: 4
  Burst Time: 7

Process 4:
  Arrival Time: 6
  Burst Time: 3

====================================
Select Scheduling Type:
1. Non-Preemptive SJF
2. Preemptive SJF (SRTF)
Enter your choice (1 or 2): 1

====================================
NON-PREEMPTIVE SJF SCHEDULING
====================================

Gantt Chart:
| P1 | P4 | P3 | P2 |
```

```
2. Preemptive SJF (SRTF)
Enter your choice (1 or 2): 1

====================================
NON-PREEMPTIVE SJF SCHEDULING
====================================

Gantt Chart:
| P1 | P4 | P3 | P2 |

====================================
SCHEDULING RESULTS
====================================

+-------+---------+-------+------------+------------+---------+
| PID   | Arrival | Burst | Completion | Turnaround | Waiting |
+-------+---------+-------+------------+------------+---------+
| P1    |       0 |     6 |          6 |          6 |       0 |
| P2    |       2 |     8 |         24 |         22 |      14 |
| P3    |       4 |     7 |         16 |         12 |       5 |
| P4    |       6 |     3 |          9 |          3 |       0 |
+-------+---------+-------+------------+------------+---------+

====================================
PERFORMANCE METRICS
====================================
Average Turnaround Time: 10.75 units
Average Waiting Time: 4.75 units
====================================

ALGORITHM ANALYSIS:
- SJF minimizes average waiting time
- Non-preemptive: Simple but may cause convoy effect
- Preemptive (SRTF): Better response time but more overhead
- Time Complexity: O(n²) for process selection

Process finished with exit code 0
```

# SCREEN SHOT/SAMPLE OUTPUT

Sample Input / Output Screenshot #2:
Number of processes: 4
Process details: (Arrival Time, Burst Time)
Process 1: 0, 6
Process 2: 2, 8
Process 3: 4, 7
Process 4: 6, 3
Selection of Scheduling Type: Preemptive SJF (SRTF) – Shortest Remaining Time First

```
C:\Users\flexycode\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 20
====================================
   SJF SCHEDULING ALGORITHM
   Operating Systems Lab - ACT2
====================================

Enter the number of processes: 4

Enter process details:

Process 1:
  Arrival Time: 0
  Burst Time: 6

Process 2:
  Arrival Time: 2
  Burst Time: 8

Process 3:
  Arrival Time: 4
  Burst Time: 7

Process 4:
  Arrival Time: 6
  Burst Time: 3

====================================
Select Scheduling Type:
1. Non-Preemptive SJF
2. Preemptive SJF (SRTF)
Enter your choice (1 or 2): 2

====================================
PREEMPTIVE SJF SCHEDULING
====================================

Execution Timeline:
Time 0: Process P1 starts/resumes
Time 6: Process P1 completed
Time 6: Process P4 starts/resumes
Time 9: Process P4 completed
Time 9: Process P3 starts/resumes
Time 16: Process P3 completed
Time 16: Process P2 starts/resumes
Time 24: Process P2 completed
========================================
SCHEDULING RESULTS
========================================

+-------+---------+-------+------------+------------+----------+
| PID   | Arrival | Burst | Completion | Turnaround | Waiting  |
+-------+---------+-------+------------+------------+----------+
| P1    |       0 |     6 |          6 |          6 |        0 |
| P2    |       2 |     8 |         24 |         22 |       14 |
| P3    |       4 |     7 |         16 |         12 |        5 |
| P4    |       6 |     3 |          9 |          3 |        0 |
+-------+---------+-------+------------+------------+----------+

========================================
PERFORMANCE METRICS
========================================
Average Turnaround Time: 10.75 units
Average Waiting Time: 4.75 units
========================================

ALGORITHM ANALYSIS:
- SJF minimizes average waiting time
- Non-preemptive: Simple but may cause convoy effect
- Preemptive (SRTF): Better response time but more overhead
- Time Complexity: O(n²) for process selection

Process finished with exit code 0
```

| ACTIVITY NO | # 3 |
|---|---|
| TITLE | Round Robin Algorithm |

# ACTIVITY 3

# RR ALGORITHM

SAMPLE OUTPUT FOR CHECKING PURPOSE ONLY

## Activity 3: Round Robin Algorithm

**Quantum Time: 4**

| Process | Arrival Time (AT) | Burst Time (BT) | Completion Time (CT) | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 8 | 24 | 24 | 16 |
| P2 | 4 | 5 | 25 | 21 | 16 |
| P3 | 2 | 7 | 28 | 26 | 19 |
| P4 | 5 | 6 | 30 | 25 | 19 |
| P5 | 3 | 4 | 20 | 17 | 13 |

AVERAGE TAT = 22.6
AVERAGE WT = 16.6

**Gantt Chart:** (Example)

| Time Units → | 0 | 4 | 8 | 12 | 16 | 20 | 24 | 25 | 28 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|
| Process → | P1 | P2 | P3 | P4 | P5 | P1 | P2 | P3 | P4 | P4 |
| Execution | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 3 | 2 | 2 |

## SOURCE CODE

```
import java.util.*;

public class SNFN {
    static class Process {
        String name;
        int arrivalTime;
        int burstTime;
        int remainingTime;
        int completionTime;
        int turnaroundTime;
        int waitingTime;

        public Process(String name, int arrivalTime, int burstTime) {
            this.name = name;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
            this.remainingTime = burstTime;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean tryAgain = true;

        while (tryAgain) {
            System.out.println("=====================================");
            System.out.println("   ROUND ROBIN SCHEDULING ALGORITHM");
            System.out.println("   Operating Systems Lab - ACT3");
            System.out.println("=====================================\n");

            // Get number of processes from user (3 to 5)
            int n = 0;
            while (n < 3 || n > 5) {
                System.out.print("Enter the number of processes (3-5): ");
                n = scanner.nextInt();
                if (n < 3 || n > 5) {
                    System.out.println("Please enter between 3 to 5 processes.");
                }
            }

            // Get time quantum
            System.out.print("Enter Quantum Time: ");
            int quantumTime = scanner.nextInt();

            // Create array to store processes
            Process[] processes = new Process[n];

            // Input process details from user
```

## SOURCE CODE

```
        System.out.println("\nEnter process details:");
        for (int i = 0; i < n; i++) {
            System.out.println("\nProcess " + (i + 1) + ":");
            System.out.print("  Arrival Time: ");
            int arrivalTime = scanner.nextInt();
            System.out.print("  Burst Time: ");
            int burstTime = scanner.nextInt();

            processes[i] = new Process("P" + (i + 1), arrivalTime, burstTime);
        }

        // Run Round Robin algorithm
        calculateRoundRobin(processes, quantumTime);

        // Ask if user wants to try again
        System.out.print("\nDo you want to try again? (yes/no): ");
        String response = scanner.next().toLowerCase();
        tryAgain = response.equals("yes") || response.equals("y");
    }

    scanner.close();
    System.out.println("Program ended. Thank you!");
}

public static void calculateRoundRobin(Process[] processes, int quantumTime) {
    int n = processes.length;
    int time = 0;
    int remainingProcesses = n;

    // For Gantt chart
    List<String> ganttProcesses = new ArrayList<>();
    List<Integer> ganttTimes = new ArrayList<>();
    ganttTimes.add(0);

    // Create a copy of processes to preserve original data
    Process[] processesCopy = new Process[n];
    for (int i = 0; i < n; i++) {
        processesCopy[i] = new Process(
                processes[i].name,
                processes[i].arrivalTime,
                processes[i].burstTime
        );
        processesCopy[i].remainingTime = processes[i].burstTime;
    }

    System.out.println("\n====================================");
    System.out.println("ROUND ROBIN SCHEDULING");
    System.out.println("Time Quantum: " + quantumTime);
    System.out.println("====================================");
```

## SOURCE CODE

```
        // Main scheduling loop
        while (remainingProcesses > 0) {
            boolean progressMade = false;

            for (int i = 0; i < n; i++) {
                Process p = processesCopy[i];

                if (p.remainingTime > 0 && p.arrivalTime <= time) {
                    progressMade = true;

                    // Add to Gantt chart
                    ganttProcesses.add(p.name);

                    if (p.remainingTime > quantumTime) {
                        time += quantumTime;
                        p.remainingTime -= quantumTime;
                    } else {
                        time += p.remainingTime;
                        p.remainingTime = 0;
                        p.completionTime = time;
                        p.turnaroundTime = p.completionTime - p.arrivalTime;
                        p.waitingTime = p.turnaroundTime - p.burstTime;
                        remainingProcesses--;
                    }

                    // Add end time to Gantt chart
                    ganttTimes.add(time);
                }
            }

            // If no progress was made, advance time
            if (!progressMade) {
                time++;
            }
        }

        // Display results table
        System.out.println("\n+-------+-------------+------------+---------------+----------------+-------------
+");
        System.out.println("| PID   | Arrival Time | Burst Time  | Completion Time| Turnaround Time |
Waiting Time |");
        System.out.println("+-------+-------------+------------+---------------+----------------+-------------+");

        double totalTurnaroundTime = 0;
        double totalWaitingTime = 0;

        for (Process p : processesCopy) {
            System.out.printf("| %-5s | %11d  | %10d  | %13d  | %14d  | %11d  |\n",
                    p.name, p.arrivalTime, p.burstTime,
                    p.completionTime, p.turnaroundTime, p.waitingTime);
```

## SOURCE CODE

```
            totalTurnaroundTime += p.turnaroundTime;
            totalWaitingTime += p.waitingTime;
        }

        System.out.println("+-------+-------------+------------+---------------+----------------+-------------+");

        // Calculate and display averages
        double avgTurnaroundTime = totalTurnaroundTime / n;
        double avgWaitingTime = totalWaitingTime / n;

        System.out.println("\n======================================");
        System.out.println("PERFORMANCE METRICS");
        System.out.println("======================================");
        System.out.printf("Average Turnaround Time: %.2f units\n", avgTurnaroundTime);
        System.out.printf("Average Waiting Time: %.2f units\n", avgWaitingTime);
        System.out.println("======================================");

        // Display Gantt chart
        System.out.println("\nGantt Chart:");
        System.out.print("Time:  ");
        for (int i = 0; i < ganttTimes.size(); i++) {
            System.out.printf("%-4d", ganttTimes.get(i));
        }
        System.out.print("\nProc:  ");
        for (int i = 0; i < ganttProcesses.size(); i++) {
            System.out.printf("%-4s", ganttProcesses.get(i));
        }
        System.out.println();
    }
}
```

## SCREEN SHOT/SAMPLE OUTPUT

```
/home/flexycode/.jdks/openjdk-22.0.2/bin/java -javaagent:/home/flexycode/.lo
=====================================
   ROUND ROBIN SCHEDULING ALGORITHM
   Operating Systems Lab - ACT3
=====================================

Enter the number of processes (3-5): 5
Enter Quantum Time: 4

Enter process details:

Process 1:
  Arrival Time: 0
  Burst Time: 8

Process 2:
  Arrival Time: 4
  Burst Time: 5

Process 3:
  Arrival Time: 2
  Burst Time: 7
```

```
Enter process details:

Process 1:
  Arrival Time: 0
  Burst Time: 8

Process 2:
  Arrival Time: 4
  Burst Time: 5

Process 3:
  Arrival Time: 2
  Burst Time: 7

Process 4:
  Arrival Time: 5
  Burst Time: 6

Process 5:
  Arrival Time: 3
  Burst Time: 4
```

NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

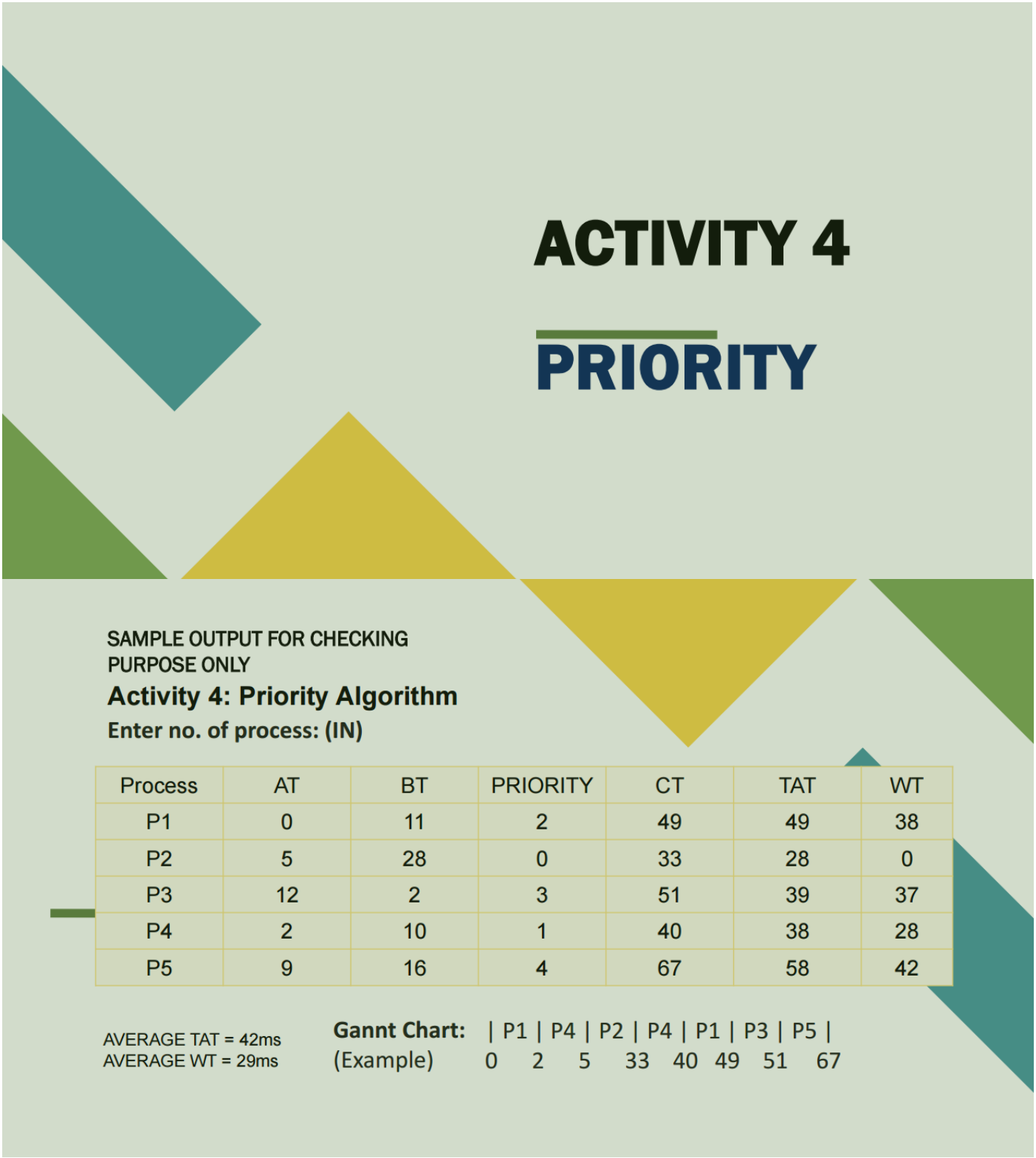
## SCREEN SHOT/SAMPLE OUTPUT

```
====================================
ROUND ROBIN SCHEDULING
Time Quantum: 4
====================================

+-------+--------------+-------------+----------------+-----------------+--------------+
| PID   | Arrival Time | Burst Time  | Completion Time| Turnaround Time | Waiting Time |
+-------+--------------+-------------+----------------+-----------------+--------------+
| P1    |            0 |           8 |             24 |              24 |           16 |
| P2    |            4 |           5 |             25 |              21 |           16 |
| P3    |            2 |           7 |             28 |              26 |           19 |
| P4    |            5 |           6 |             30 |              25 |           19 |
| P5    |            3 |           4 |             20 |              17 |           13 |
+-------+--------------+-------------+----------------+-----------------+--------------+

====================================
PERFORMANCE METRICS
====================================
Average Turnaround Time: 22.60 units
Average Waiting Time: 16.60 units
====================================
```


```
+-------+--------------+-------------+----------------+-----------------+--------------+
| PID   | Arrival Time | Burst Time  | Completion Time| Turnaround Time | Waiting Time |
+-------+--------------+-------------+----------------+-----------------+--------------+
| P1    |            0 |           8 |             24 |              24 |           16 |
| P2    |            4 |           5 |             25 |              21 |           16 |
| P3    |            2 |           7 |             28 |              26 |           19 |
| P4    |            5 |           6 |             30 |              25 |           19 |
| P5    |            3 |           4 |             20 |              17 |           13 |
+-------+--------------+-------------+----------------+-----------------+--------------+

====================================
PERFORMANCE METRICS
====================================
Average Turnaround Time: 22.60 units
Average Waiting Time: 16.60 units
====================================
```


```
Gantt Chart:
Time:  0   4   8   12  16  20  24  25  28  30
Proc:  P1  P2  P3  P4  P5  P1  P2  P3  P4

Do you want to try again? (yes/no):
```

| ACTIVITY NO | # 4 |
|---|---|
| TITLE | Priority Algorithm |



## SOURCE CODE

```
import java.util.*;

public class SNFN {
    static class Process {
        String name;
        int arrivalTime;
        int burstTime;
        int priority;
        int remainingTime;
        int completionTime;
        int turnaroundTime;
        int waitingTime;
        int executedTime;

        public Process(String name, int arrivalTime, int burstTime, int priority) {
            this.name = name;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
            this.priority = priority;
            this.remainingTime = burstTime;
            this.executedTime = 0;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean tryAgain = true;

        while (tryAgain) {
            System.out.print("Enter the number of processes (3-5): ");
            int n = scanner.nextInt();

            // Create array to store processes
            Process[] processes = new Process[n];

            // Input process details from user
            System.out.println("\nEnter process details:");
            for (int i = 0; i < n; i++) {
                System.out.println("\nProcess " + (i + 1) + ":");
                System.out.print("  Arrival Time: ");
                int arrivalTime = scanner.nextInt();
                System.out.print("  Burst Time: ");
                int burstTime = scanner.nextInt();
                System.out.print("  Priority: ");
                int priority = scanner.nextInt();

                processes[i] = new Process("P" + (i + 1), arrivalTime, burstTime, priority);
            }
```

## SOURCE CODE

```
        // Run Priority algorithm
        calculatePriority(processes);

        // Ask if user wants to try again
        System.out.print("\nDo you want to try again? (yes/no): ");
        String response = scanner.next().toLowerCase();
        tryAgain = response.equals("yes") || response.equals("y");
    }

    scanner.close();
    System.out.println("Program ended. Thank you!");
}

public static void calculatePriority(Process[] processes) {
    int n = processes.length;
    int currentTime = 0;
    int completedProcesses = 0;

    // For Gantt chart
    List<String> ganttProcesses = new ArrayList<>();
    List<Integer> ganttTimes = new ArrayList<>();
    ganttTimes.add(0);

    // Create a copy of processes to preserve original data
    Process[] processesCopy = new Process[n];
    for (int i = 0; i < n; i++) {
        processesCopy[i] = new Process(
                processes[i].name,
                processes[i].arrivalTime,
                processes[i].burstTime,
                processes[i].priority
        );
        processesCopy[i].remainingTime = processes[i].burstTime;
    }

    // Sort by arrival time for initial processing
    Arrays.sort(processesCopy, Comparator.comparingInt(p -> p.arrivalTime));

    Process currentProcess = null;

    while (completedProcesses < n) {
        // Check for new arrivals and update priorities
        Process highestPriorityProcess = null;
        for (Process p : processesCopy) {
            if (p.arrivalTime <= currentTime && p.remainingTime > 0) {
                if (highestPriorityProcess == null || p.priority < highestPriorityProcess.priority) {
                    highestPriorityProcess = p;
                }
            }
        }
```

## SOURCE CODE

```
            if (highestPriorityProcess != null) {
                if (currentProcess != highestPriorityProcess) {
                    if (currentProcess != null && currentProcess.remainingTime > 0) {
                        // Record the end of the current process execution
                        ganttTimes.add(currentTime);
                        ganttProcesses.add(currentProcess.name);
                    }
                    currentProcess = highestPriorityProcess;
                }

                // Execute for 1 time unit
                currentProcess.remainingTime--;
                currentProcess.executedTime++;
                currentTime++;

                if (currentProcess.remainingTime == 0) {
                    currentProcess.completionTime = currentTime;
                    currentProcess.turnaroundTime = currentProcess.completionTime -
currentProcess.arrivalTime;
                    currentProcess.waitingTime = currentProcess.turnaroundTime -
currentProcess.burstTime;
                    completedProcesses++;

                    ganttTimes.add(currentTime);
                    ganttProcesses.add(currentProcess.name);
                    currentProcess = null;
                }
            } else {
                currentTime++;
            }
        }

        // Display results table
        System.out.println("\n+-------+-------------+------------+---------+---------------+----------------+------
--------+");
        System.out.println("| PID   | Arrival Time | Burst Time  | Priority | Completion Time|
Turnaround Time | Waiting Time |");
        System.out.println("+-------+-------------+------------+---------+---------------+----------------+--------
------+");

        double totalTurnaroundTime = 0;
        double totalWaitingTime = 0;

        // Reset to original order for display
        Arrays.sort(processesCopy, Comparator.comparing(p -> p.name));

        for (Process p : processesCopy) {
            System.out.printf("| %-5s | %11d  | %10d  | %7d  | %13d  | %14d  | %11d  |\n",
                    p.name, p.arrivalTime, p.burstTime, p.priority,
```

## SOURCE CODE

```
                p.completionTime, p.turnaroundTime, p.waitingTime);

            totalTurnaroundTime += p.turnaroundTime;
            totalWaitingTime += p.waitingTime;
        }

        System.out.println("+-------+-------------+------------+----------+---------------+----------------+--------
------+");

        // Calculate and display averages
        double avgTurnaroundTime = totalTurnaroundTime / n;
        double avgWaitingTime = totalWaitingTime / n;

        System.out.println("\nPERFORMANCE METRICS");
        System.out.println("====================================");
        System.out.printf("Average Turnaround Time: %.2f ms\n", avgTurnaroundTime);
        System.out.printf("Average Waiting Time: %.2f ms\n", avgWaitingTime);
        System.out.println("====================================");

        // Display Gantt chart in the correct format
        System.out.println("\nGantt Chart:");
        System.out.print("Proc:  ");
        for (int i = 0; i < ganttProcesses.size(); i++) {
            System.out.printf("%-4s", ganttProcesses.get(i));
            if (i < ganttProcesses.size() - 1) {
                System.out.print("| ");
            }
        }
        System.out.print("\nTime:  ");
        for (Integer ganttTime : ganttTimes) {
            System.out.printf("%-4d", ganttTime);
        }
        System.out.println();
    }
}
```

## SCREEN SHOT/SAMPLE OUTPUT

```
C:\Users\flexycode\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
Enter the number of processes (3-5): 5

Enter process details:

Process 1:
  Arrival Time: 0
  Burst Time: 11
  Priority: 2

Process 2:
  Arrival Time: 5
  Burst Time: 28
  Priority: 0

Process 3:
  Arrival Time: 12
  Burst Time: 2
  Priority: 3

Process 4:
  Arrival Time: 2
  Burst Time: 10
  Priority: 1

Process 5:
  Arrival Time: 9
  Burst Time: 16
  Priority: 4
```

| PID | Arrival Time | Burst Time | Priority | Completion Time | Turnaround Time | Waiting Time |
|---|---|---|---|---|---|---|
| P1 | 0 | 11 | 2 | 49 | 49 | 38 |
| P2 | 5 | 28 | 0 | 33 | 28 | 0 |
| P3 | 12 | 2 | 3 | 51 | 39 | 37 |
| P4 | 2 | 10 | 1 | 40 | 38 | 28 |
| P5 | 9 | 16 | 4 | 67 | 58 | 42 |

```
PERFORMANCE METRICS
===================================
Average Turnaround Time: 42.40 ms
Average Waiting Time: 29.00 ms
===================================

Gantt Chart:
Proc:  P1  | P4  | P2  | P4  | P1  | P3  | P5
Time:  0   2   5   33  40  49  51  67

Do you want to try again? (yes/no): yes
```

| ACTIVITY NO | # 5 |
|---|---|
| TITLE | Highest Response Ratio Next Algorithm |

# ACTIVITY 5

## HRRN

SAMPLE OUTPUT FOR CHECKING PURPOSE ONLY

**Activity 5: HRRN Algorithm**

**Enter no. of process: (IN)**

| Process | AT | BT | CT | TAT | WT | RT |
|---|---|---|---|---|---|---|
| P1 | 1 | 3 | 4 | 3 | 0 | 0 |
| P2 | 3 | 6 | 10 | 7 | 1 | 1 |
| P3 | 5 | 8 | 27 | 22 | 14 | 14 |
| P4 | 7 | 4 | 14 | 7 | 3 | 3 |
| P5 | 8 | 5 | 19 | 11 | 6 | 6 |

AVERAGE TAT = 10ms
AVERAGE WT = 4.8ms

**Gannt Chart:** (Example)

| | P1 | P2 | P4 | P5 | P3 | |
|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 10 | 14 | 19 | 27 |

## SOURCE CODE

```
import java.util.*;

public class SNFN {
    static class Process {
        String name;
        int arrivalTime;
        int burstTime;
        int completionTime;
        int turnaroundTime;
        int waitingTime;
        int responseTime;
        boolean started;

        public Process(String name, int arrivalTime, int burstTime) {
            this.name = name;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
            this.started = false;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean tryAgain = true;

        while (tryAgain) {
            System.out.print("Enter the number of processes (3-5): ");
            int n = scanner.nextInt();

            // Validate input range
            if (n < 3 || n > 5) {
                System.out.println("Please enter 3 to 5 processes only.");
                continue;
            }

            // Create array to store processes
            Process[] processes = new Process[n];

            // Input process details from user
            System.out.println("\nEnter process details:");
            for (int i = 0; i < n; i++) {
                System.out.println("\nProcess " + (i + 1) + ":");
                System.out.print("  Arrival Time: ");
                int arrivalTime = scanner.nextInt();
                System.out.print("  Burst Time: ");
                int burstTime = scanner.nextInt();

                processes[i] = new Process("P" + (i + 1), arrivalTime, burstTime);
            }
```

## SOURCE CODE

```
        // Run HRRN algorithm
        calculateHRRN(processes);

        // Ask if user wants to try again
        System.out.print("\nDo you want to try again? (yes/no): ");
        String response = scanner.next().toLowerCase();
        tryAgain = response.equals("yes") || response.equals("y");
    }

    scanner.close();
    System.out.println("Program ended. Thank you!");
}

public static void calculateHRRN(Process[] processes) {
    int n = processes.length;
    int currentTime = 0;
    int completedProcesses = 0;

    // For Gantt chart
    List<String> ganttProcesses = new ArrayList<>();
    List<Integer> ganttTimes = new ArrayList<>();
    ganttTimes.add(0);

    // Create a copy of processes to preserve original data
    Process[] processesCopy = new Process[n];
    for (int i = 0; i < n; i++) {
        processesCopy[i] = new Process(
                processes[i].name,
                processes[i].arrivalTime,
                processes[i].burstTime
        );
    }

    while (completedProcesses < n) {
        Process selectedProcess = null;
        double highestResponseRatio = -1;

        // Find process with highest response ratio that has arrived and not completed
        for (Process p : processesCopy) {
            if (p.completionTime == 0 && p.arrivalTime <= currentTime) {
                int waitingTime = currentTime - p.arrivalTime;
                double responseRatio = (double) (waitingTime + p.burstTime) / p.burstTime;

                if (responseRatio > highestResponseRatio) {
                    highestResponseRatio = responseRatio;
                    selectedProcess = p;
                }
            }
        }
```

## SOURCE CODE

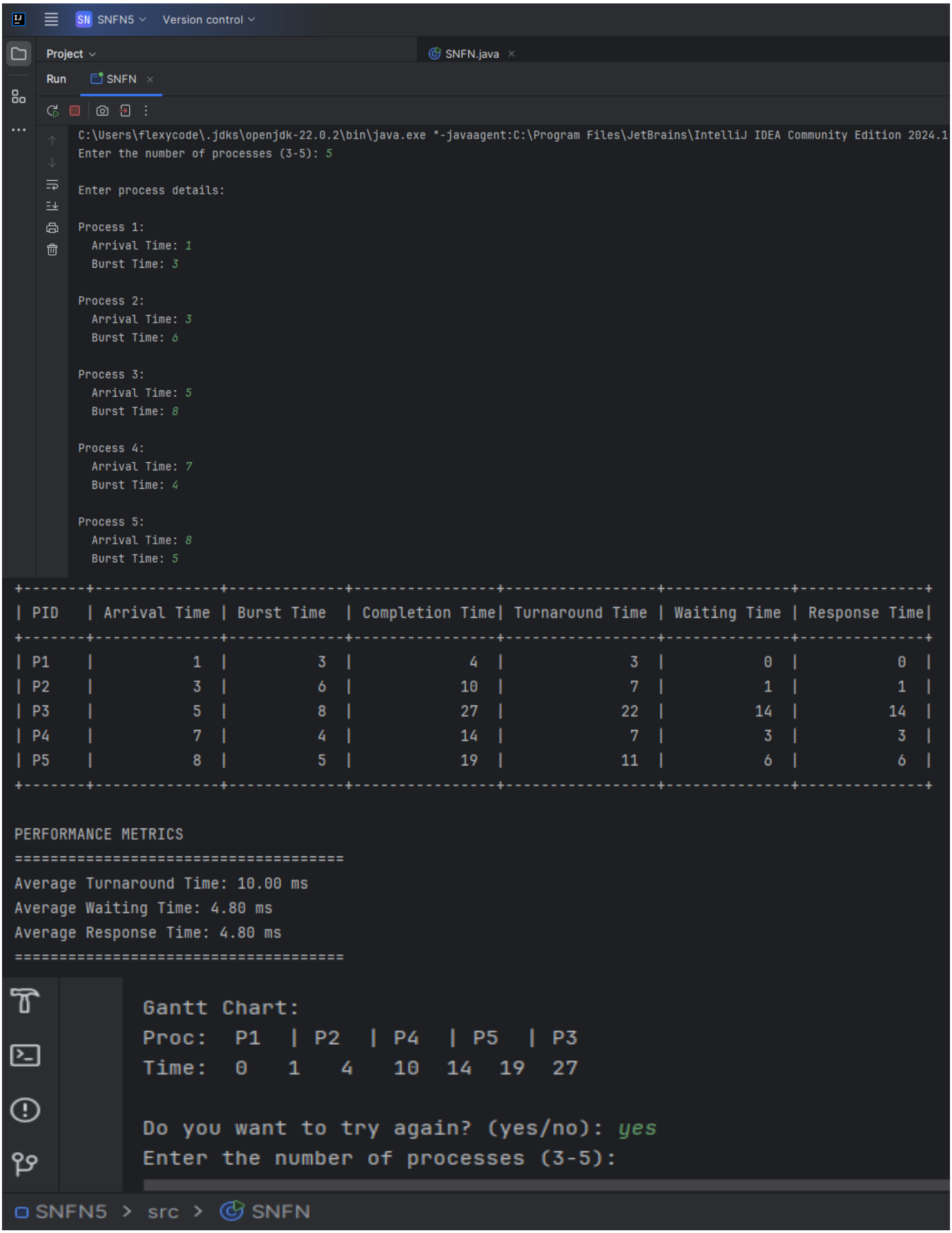
```
            // If no process found, advance time
            if (selectedProcess == null) {
                currentTime++;
                continue;
            }

            // Record start time for response time calculation (first time process runs)
            if (!selectedProcess.started) {
                selectedProcess.responseTime = currentTime - selectedProcess.arrivalTime;
                selectedProcess.started = true;
            }

            // Add to Gantt chart
            ganttProcesses.add(selectedProcess.name);
            ganttTimes.add(currentTime);

            // Execute the selected process to completion (non-preemptive)
            currentTime += selectedProcess.burstTime;
            selectedProcess.completionTime = currentTime;
            selectedProcess.turnaroundTime = selectedProcess.completionTime -
selectedProcess.arrivalTime;
            selectedProcess.waitingTime = selectedProcess.turnaroundTime -
selectedProcess.burstTime;
            completedProcesses++;

            // Add completion time to Gantt chart
            ganttTimes.add(currentTime);
        }

        // Display results table
        System.out.println("\n+-------+-------------+------------+---------------+----------------+-------------+--
-----------+");
        System.out.println("| PID   | Arrival Time | Burst Time  | Completion Time| Turnaround Time |
Waiting Time | Response Time|");
        System.out.println("+-------+-------------+------------+---------------+----------------+-------------+----
----------+");

        double totalTurnaroundTime = 0;
        double totalWaitingTime = 0;
        double totalResponseTime = 0;

        // Sort by process name for display
        Arrays.sort(processesCopy, Comparator.comparing(p -> p.name));

        for (Process p : processesCopy) {
            System.out.printf("| %-5s | %11d  | %10d  | %13d  | %14d  | %11d  | %11d  |\n",
                    p.name, p.arrivalTime, p.burstTime,
                    p.completionTime, p.turnaroundTime, p.waitingTime, p.responseTime);
```

## SOURCE CODE

```
            totalTurnaroundTime += p.turnaroundTime;
            totalWaitingTime += p.waitingTime;
            totalResponseTime += p.responseTime;
        }

        System.out.println("+-------+-------------+------------+---------------+----------------+-------------+----
----------+");

        // Calculate and display averages
        double avgTurnaroundTime = totalTurnaroundTime / n;
        double avgWaitingTime = totalWaitingTime / n;
        double avgResponseTime = totalResponseTime / n;

        System.out.println("\nPERFORMANCE METRICS");
        System.out.println("====================================");
        System.out.printf("Average Turnaround Time: %.2f ms\n", avgTurnaroundTime);
        System.out.printf("Average Waiting Time: %.2f ms\n", avgWaitingTime);
        System.out.printf("Average Response Time: %.2f ms\n", avgResponseTime);
        System.out.println("====================================");

        // Display Gantt chart without duplicates
        System.out.println("\nGantt Chart:");
        System.out.print("Proc:  ");
        for (int i = 0; i < ganttProcesses.size(); i++) {
            System.out.printf("%-4s", ganttProcesses.get(i));
            if (i < ganttProcesses.size() - 1) {
                System.out.print("| ");
            }
        }
        System.out.print("\nTime:  ");
        // Remove duplicate times from Gantt chart
        List<Integer> uniqueTimes = new ArrayList<>();
        for (int i = 0; i < ganttTimes.size(); i++) {
            if (i == 0 || !ganttTimes.get(i).equals(ganttTimes.get(i - 1))) {
                uniqueTimes.add(ganttTimes.get(i));
            }
        }
        for (int time : uniqueTimes) {
            System.out.printf("%-4d", time);
        }
        System.out.println();
    }
}
```

NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

## SCREEN SHOT/SAMPLE OUTPUT



```
C:\Users\flexycode\.jdks\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.1
Enter the number of processes (3-5): 5

Enter process details:

Process 1:
  Arrival Time: 1
  Burst Time: 3

Process 2:
  Arrival Time: 3
  Burst Time: 6

Process 3:
  Arrival Time: 5
  Burst Time: 8

Process 4:
  Arrival Time: 7
  Burst Time: 4

Process 5:
  Arrival Time: 8
  Burst Time: 5
```

| PID | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
|---|---|---|---|---|---|---|
| P1 | 1 | 3 | 4 | 3 | 0 | 0 |
| P2 | 3 | 6 | 10 | 7 | 1 | 1 |
| P3 | 5 | 8 | 27 | 22 | 14 | 14 |
| P4 | 7 | 4 | 14 | 7 | 3 | 3 |
| P5 | 8 | 5 | 19 | 11 | 6 | 6 |

```
PERFORMANCE METRICS
====================================
Average Turnaround Time: 10.00 ms
Average Waiting Time: 4.80 ms
Average Response Time: 4.80 ms
====================================

Gantt Chart:
Proc:  P1  | P2  | P4  | P5  | P3
Time:  0   1   4   10  14  19  27

Do you want to try again? (yes/no): yes
Enter the number of processes (3-5):
```

SNFN5 > src > SNFN