3. Process Scheduling Algorithm

Aim:

To implement process scheduling for the given scenario.

Program:

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Demonstrating non-preemptive SJF and preemptive SJF i.e. SRTF
 *
 * @author jimil
 */
public class SJF {

    static List<Process> processes = new ArrayList<>();
    static List<Process> scheduledList = new ArrayList<>();

    /**
     * Driver function for program
     *
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        int n;
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter the number of processes in the system:");
        n = Integer.parseInt(br.readLine());
        for(int i = 1; i <= n; i++) {
            System.out.print("Arrival Time: ");
            int at = Integer.parseInt(br.readLine());
            System.out.println();
            System.out.print("Burst Time: ");
            int bt = Integer.parseInt(br.readLine());
            System.out.println();
            Process p = new Process();
            p.setId(i);
            p.setAt(at);
            p.setBt(bt);
            processes.add(p);
        }
        System.out.println("Processes Initialized...");
        Collections.sort(processes, new Comparator<Process>() {
            @Override
            public int compare(Process p1, Process p2) {
                return p1.getAt() - p2.getAt();
            }
        });
        System.out.println("Sorted list of processes according to AT: ");
        for(Process p: processes) {
            System.out.print(p.getId() + "\t");
        }
        System.out.println();
        System.out.println("Select algorithm for scheduling: ");
        System.out.println("1.      Non-Preemptive SJF");
        System.out.println("2.      Preemptive SJF");
        int ch = Integer.parseInt(br.readLine());
        switch(ch) {
            case 1:
System.out.println("--------------------------NON-PREEMPTIVE
SJF-----------------------------");
                nonPreemptiveSJF();
                printTable();
                break;
            case 2:
System.out.println("------------------------------PREEMPTIVE
SJF------------------------------");
                preemptiveSJF();
                printTable();
                break;
            default: System.out.println("Invalid Choice...");
                break;
        }
    }
```

```java
/**
 * At each value of the clock tick represented by the currentTime,
 * fetch a list of all possible candidates that can be scheduled.
 * This list of candidates is a list of all processes having arrival
 * time less than or equal to the current time.
 *
 * @param currentTime
 * @return
 */
public static List<Process> getPossibleChoices(int currentTime) {
    List<Process> result = new ArrayList<Process>();
    for(Process p: processes) {
        if(p.getAt() <= currentTime) {
            result.add(p);
        }
    }
    return result;
}

/**
 * Out of a list of all possible candidates, this function selects
 * the next process that is to be scheduled, and returns the index
 * of the process in the processes list.
 *
 * Criteria for Decision: Shortest Remaining Time
 * In Case of Conflict: 1. Arrival Time; 2. PID
 *
 * @param choices
 * @return
 */
public static int toScheduleOnRemaining(List<Process> choices) {
    int min = 10000, index= -10;
    Process selected = choices.get(0);
    if(!choices.isEmpty()) {
        for(Process p: choices) {
            if(p.getRemaining() < min) {
                min = p.getRemaining();
                selected = p;
            } else if(p.getRemaining() == min) {
                if(p.getAt() < selected.getAt()) {
                    min = p.getRemaining();
                    selected = p;
                } else if(p.getAt() == selected.getAt()) {
                    if(p.getId() < selected.getId()) {
                        min = p.getRemaining();
                        selected = p;
                    }
                }
            }
        }
        index = processes.indexOf(selected);
    }
    return index;
}

/**
 * Out of a list of all possible candidates, this function selects
 * the next process that is to be scheduled, and returns the index
 * of the process in the processes list.
 *
 * Criteria for Decision: Shortest Burst Time
 * In Case of Conflict: 1. Arrival Time; 2. PID
 *
 * @param choices
 * @return
 */
public static int toSchedule(List<Process> choices) {
    int min = 10000, index= -10;
    Process selected = processes.get(0);
    if(!choices.isEmpty()) {
        for(Process p: choices) {
            if(p.getBt() < min) {
                min = p.getBt();
                selected = p;
            } else if(p.getBt() == min) {
                if(p.getAt() < selected.getAt()) {
                    min = p.getBt();
                    selected = p;
                } else if(p.getAt() == selected.getAt()) {
                    if(p.getId() < selected.getId()) {
                        min = p.getBt();
                        selected = p;
                    }
                }
            }
        }
        index = processes.indexOf(selected);
    }
    return index;
}

/**
 * Non-preemptive SJF implementation for scheduling the processes
 * created by the user.
 */
public static void nonPreemptiveSJF() {
    int currentTime = 0;
    List<Process> choices = getPossibleChoices(currentTime);
    do {
        int index = toSchedule(choices);
        if(index >= 0) {
            Process scheduled = processes.get(index);
            scheduled.setStartedAt(currentTime);
            System.out.println("Scheduled Process " +
scheduled.getId());
            currentTime += scheduled.getBt();
            scheduled.setEndedAt(currentTime);
            processes.remove(scheduled);
            scheduledList.add(scheduled);
        } else {
            currentTime++;
        }
        choices = getPossibleChoices(currentTime);
    } while(!choices.isEmpty());
    System.out.println("Scheduling done...");
    for(Process p: scheduledList) {
        p.setCt(p.getEndedAt());
        p.setTat(p.getCt() - p.getAt());
        p.setWt(p.getTat() - p.getBt());
        p.setRt(p.getWt());
    }
}

/**
 * Preemptive SJF implementation to schedule the processes
```

```
     * created by the user.
     */
    public static void preemptiveSJF() {
        int currentTime = 0;
        List<Process> choices = getPossibleChoices(currentTime);
        do {
            int index = toScheduleOnRemaining(choices);
            if(index >= 0) {
                System.out.println("Scheduling Process " + (index+1));
                if(processes.get(index).getPrev() == false) {
                    processes.get(index).setPrev(true);
                    processes.get(index).setStartedAt(currentTime);
                }
                int remaining = processes.get(index).getRemaining()-1;
                currentTime++;
                if(remaining != 0) {
                    processes.get(index).setRemaining(remaining);
                } else {
                    processes.get(index).setEndedAt(currentTime);
                    scheduledList.add(processes.get(index));
                    processes.remove(processes.get(index));
                }
            } else {
                currentTime++;
            }
            choices = getPossibleChoices(currentTime);
        } while(!choices.isEmpty());
        System.out.println("Scheduling done...");
        for(Process p: scheduledList) {
            p.setCt(p.getEndedAt());
            p.setTat(p.getCt() - p.getAt());
            p.setWt(p.getTat() - p.getBt());
            p.setRt(p.getStartedAt() - p.getAt());
        }
    }

    /**
     * For printing the output
     */
    public static void printTable() {
        System.out.println("ID\tAT\tBT\tCT\tTAT\tWT\tRT");
        int sumTat = 0;
        int sumWt = 0;
        for(Process p: scheduledList) {
            System.out.println(p.getId() + "\t" + p.getAt() + "\t" +
p.getBt()
                    + "\t" + p.getCt() + "\t" + p.getTat() + "\t" +
p.getWt()
                    + "\t" + p.getRt());
            sumTat += p.getTat();
            sumWt += p.getWt();
        }

System.out.println("-------------------------------------------------------
--");
        System.out.println("Average Turnaround Time = " +
((float)sumTat/scheduledList.size()));
        System.out.println("Average Wait Time = " +
((float)sumWt/scheduledList.size()));
    }

}

/**
 * A POJO class that represents a process in CPU.
 */
class Process {

    private int id;
    private int at;
    private int bt;
    private int ct;
    private int tat;
    private int wt;
    private int rt;
    private int startedAt;
    private int endedAt;
    private int remaining;
    private boolean prev = false;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getAt() {
        return at;
    }
    public void setAt(int at) {
        this.at = at;
    }
    public int getBt() {
        return bt;
    }
    public void setBt(int bt) {
        this.bt = bt;
        setRemaining(bt);
    }
    public int getCt() {
        return ct;
    }
    public void setCt(int ct) {
        this.ct = ct;
    }
    public int getTat() {
        return tat;
    }
    public void setTat(int tat) {
        this.tat = tat;
    }
    public int getWt() {
        return wt;
    }
    public void setWt(int wt) {
        this.wt = wt;
    }
    public int getRt() {
        return rt;
    }
    public void setRt(int rt) {
        this.rt = rt;
    }
    public int getStartedAt() {
        return startedAt;
    }
    public void setStartedAt(int startedAt) {
        this.startedAt = startedAt;
    }
    public int getEndedAt() {
        return endedAt;
    }
    public void setEndedAt(int endedAt) {
        this.endedAt = endedAt;
    }
    public boolean getPrev() {
        return prev;
    }
    public void setPrev(boolean prev) {
        this.prev = prev;
    }
    public int getRemaining() {
        return remaining;
    }
    public void setRemaining(int remaining) {
        this.remaining = remaining;
    }

}
```

Output for non-preemptive SJF:

```
Enter the number of processes in the system:
5
Arrival Time: 1
Burst Time: 7

Arrival Time: 2
Burst Time: 5

Arrival Time: 3
Burst Time: 1

Arrival Time: 4
Burst Time: 2

Arrival Time: 5
Burst Time: 8

Processes Initialized...
Sorted list of processes according to AT:
1     2     3     4     5
Select algorithm for scheduling:
1.    Non-Preemptive SJF
2.    Preemptive SJF
1
--------------------------NON-PREEMPTIVE SJF--------------------------
Scheduled Process 1
Scheduled Process 3
Scheduled Process 4
Scheduled Process 2
Scheduled Process 5
Scheduling done...
ID    AT    BT    CT    TAT   WT    RT
1     1     7     8     7     0     0
3     3     1     9     6     5     5
4     4     2     11    7     5     5
2     2     5     16    14    9     9
5     5     8     24    19    11    11
---------------------------------------------------------
Average Turnaround Time = 10.6
Average Wait Time = 6.0
```

Output for preemptive SJF:

```
Enter the number of processes in the system:
5
Arrival Time: 1
Burst Time: 7

Arrival Time: 2
Burst Time: 5

Arrival Time: 3
Burst Time: 1

Arrival Time: 4
Burst Time: 2

Arrival Time: 5
Burst Time: 8

Processes Initialized...
Sorted list of processes according to AT:
1     2     3     4     5
Select algorithm for scheduling:
1.    Non-Preemptive SJF
2.    Preemptive SJF
2
------------------------------PREEMPTIVE SJF-----------------------------
Scheduling Process 1
Scheduling Process 2
Scheduling Process 3
Scheduling Process 3
Scheduling Process 3
Scheduling Process 2
Scheduling Process 2
Scheduling Process 2
Scheduling Process 2
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling Process 1
Scheduling done...
ID     AT     BT     CT     TAT    WT     RT
3      3      1      4      1      0      0
4      4      2      6      2      0      0
2      2      5      10     8      3      0
1      1      7      16     15     8      0
5      5      8      24     19     11     11
---------------------------------------------------------
Average Turnaround Time = 9.0
Average Wait Time = 4.4
```

Conclusion:

Thus, through this experiment, I understood about process scheduling algorithms and implemented Shortest Job First (SJF) algorithm along with both it's variations - non-preemptive SJF, as well as, preemptive SJF (also called as Shortest Remaining Time First or SRTF).
SJF increases the throughput since, by definition, shortest jobs are executed before longer jobs. However, SJF is theoretical and cannot be implemented, since CPU must know the burst time of the processes to be scheduled, beforehand. This is practically impossible to achieve. Static (Prediction on process size and process type) and Dynamic prediction algorithms(Simple & Exponential Averaging) have been proposed to predict the burst time of a process, but these algorithms are far from practical implementation.