Write a program that implements RR scheduling algorithm.

```
import time

class Process:
    def __init__(self, pid, arrival_time, burst_time):
    self.pid = pid
    self.arrival_time = arrival_time
    self.burst_time = burst_time
    self.remaining_time = burst_time

def rr_scheduling(processes, time_quantum):
    """
    Implements the Round Robin scheduling algorithm.

    Args:
    processes: A list of Process objects.
    time_quantum: The time quantum for each process.
    """

    current_time = 0
    waiting_time = 0
    turnaround_time = 0

    while any(process.remaining_time > 0 for process in processes):
    for process in processes:
    if process.arrival_time <= current_time and process.remaining_time > 0:
        if process.remaining_time <= time_quantum:
        current_time += process.remaining_time
        waiting_time += current_time - process.arrival_time - process.burst_time
        turnaround_time += current_time - process.arrival_time
        process.remaining_time = 0
        else:
        current_time += time_quantum
        process.remaining_time -= time_quantum

    average_waiting_time = waiting_time / len(processes)
    average_turnaround_time = turnaround_time / len(processes)

    print("Average Waiting Time:", average_waiting_time)
    print("Average Turnaround Time:", average_turnaround_time)

# Example usage
processes = [
    Process(1, 0, 5),
    Process(2, 1, 2),
    Process(3, 2, 1),
    Process(4, 3, 4)
]

time_quantum = 2

rr_scheduling(processes, time_quantum)
```

```
import heapq

class Process:
        def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time

def rr_scheduling(processes, time_quantum):
        """
        Implements the Round Robin scheduling algorithm.

        Args:
        processes: A list of Process objects.
        time_quantum: The time quantum for each process.
        """

        current_time = 0
        waiting_time = 0
        turnaround_time = 0
        ready_queue = []  # Use a heap for efficient priority-based operations

        # Add processes to the ready queue as they arrive
        for process in processes:
        heapq.heappush(ready_queue, (process.arrival_time, process.pid))

        while ready_queue or any(process.remaining_time > 0 for process in processes):
        # If the ready queue is empty, wait for the next arrival
        if not ready_queue:
        current_time = min(process.arrival_time for process in processes if process.remaining_time > 0)
        continue

        # Get the next process from the ready queue
        _, pid = heapq.heappop(ready_queue)
        process = next(process for process in processes if process.pid == pid)

        # Execute the process for the time quantum
        if process.remaining_time <= time_quantum:
        current_time += process.remaining_time
        waiting_time += current_time - process.arrival_time - process.burst_time
        turnaround_time += current_time - process.arrival_time
        process.remaining_time = 0
        else:
        current_time += time_quantum
        process.remaining_time -= time_quantum
        heapq.heappush(ready_queue, (current_time, process.pid))

        average_waiting_time = waiting_time / len(processes)
        average_turnaround_time = turnaround_time / len(processes)

        print("Average Waiting Time:", average_waiting_time)
        print("Average Turnaround Time:", average_turnaround_time)
```

```
# Example usage
processes = [
        Process(1, 0, 5),
        Process(2, 1, 2),
        Process(3, 2, 1),
        Process(4, 3, 4)
]

time_quantum = 2

rr_scheduling(processes, time_quantum)
```