Write a program to simulate the preemptive priority based scheduling

class Process:

    def \_\_init\_\_(self, pid, arrival\_time, burst\_time, priority):

        self.pid = pid

        self.arrival\_time = arrival\_time

        self.burst\_time = burst\_time

        self.priority = priority

        self.remaining\_time = burst\_time

    def \_\_lt\_\_(self, other):

        return self.priority < other.priority

    def execute(self, current\_time):

        print(f"Process {self.pid} started execution at time {current\_time}")

        self.remaining\_time -= 1

        if self.remaining\_time == 0:

            print(f"Process {self.pid} completed execution at time {current\_time+1}")

            return True

        return False

def preemptive\_priority\_scheduling(processes):

    ready\_queue = []

    current\_time = 0

    completed\_processes = []

    while len(processes) > 0 or len(ready\_queue) > 0:

        while len(processes) > 0 and processes[0].arrival\_time == current\_time:

            ready\_queue.append(processes.pop(0))

        if len(ready\_queue) == 0:

            current\_time += 1

            continue

        ready\_queue.sort(reverse=True)

        process = ready\_queue.pop(0)

        if process.execute(current\_time):

            completed\_processes.append(process)

        else:

            ready\_queue.append(process)

        current\_time += 1

    return completed\_processes

processes = [

    Process(3, 0, 6, 2),

    Process(6, 1, 4, 3),

    Process(4, 2, 9, 5),

    Process(2, 3, 8, 7),

    Process(8, 4, 7, 3)

]

completed\_processes = preemptive\_priority\_scheduling(processes)

print("Completed Processes:")

for process in completed\_processes:

    print(f"Process {process.pid}: Arrival Time={process.arrival\_time}, Burst Time={process.burst\_time}, Priority={process.priority}")

OUTPUT:

