❑ Implementation:

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt


# Sample scheduling functions - placeholders for real logic
def fcfs(processes):  1 usage
    # Sort by arrival time
    processes.sort(key=lambda x: x['Arrival Time'])
    return calculate_metrics(processes)


def sjf(processes):  1 usage
    # Sort by burst time, then arrival time
    processes.sort(key=lambda x: (x['Burst Time'], x['Arrival Time']))
    return calculate_metrics(processes)


def priority_scheduling(processes):  1 usage
    # Sort by priority, then arrival time
    processes.sort(key=lambda x: (x['Priority'], x['Arrival Time']))
    return calculate_metrics(processes)
def calculate_metrics(processes):  3 usages
    # Calculate completion, waiting, and turnaround times
    current_time = 0
    for process in processes:
        process['Start Time'] = max(current_time, process['Arrival Time'])
        process['Completion Time'] = process['Start Time'] + process['Burst Time']
        process['Turnaround Time'] = process['Completion Time'] - process['Arrival Time']
        process['Waiting Time'] = process['Turnaround Time'] - process['Burst Time']
        current_time = process['Completion Time']
    return processes


def visualize_gantt_chart(processes, algorithm_name):  1 usage
    fig, ax = plt.subplots(figsize=(10, 5))
    for process in processes:
        ax.broken_barh([(process['Start Time'], process['Burst Time'])],
                       (process['Process ID'] * 10, 9), facecolors=('tab:blue'))
    ax.set_ylim(0, (len(processes) + 1) * 10)
    ax.set_xlim(0, max(p['Completion Time'] for p in processes) + 10)
    ax.set_xlabel('Time')
    ax.set_ylabel('Process ID')
    ax.set_yticks([p['Process ID'] * 10 + 5 for p in processes])
    ax.set_yticklabels([f"P{p['Process ID']}" for p in processes])
    ax.set_title(f"Gantt Chart for {algorithm_name} Scheduling")
    return fig
```

```python
# Streamlit app setup
st.title("Dynamic CPU Scheduling Simulator")
st.write("""
This simulator demonstrates and compares the performance of three scheduling algorithms:
- **FCFS** (First-Come, First-Served)
- **SJF** (Shortest Job First)
- **Priority Scheduling**

Input process details, select an algorithm, and view the scheduling results and Gantt chart.
""")

# User inputs
num_processes = st.sidebar.number_input("Number of processes:", min_value=1, max_value=10, step=1)
processes = []

st.sidebar.header("Process Details")
for i in range(num_processes):
    st.sidebar.subheader(f"Process {i + 1}")
    arrival_time = st.sidebar.number_input(f"Arrival Time (P{i + 1})", min_value=0)
    burst_time = st.sidebar.number_input(f"Burst Time (P{i + 1})", min_value=1)
    priority = st.sidebar.number_input(f"Priority (P{i + 1})", min_value=1, max_value=10)
    processes.append({
        'Process ID': i + 1,
        'Arrival Time': arrival_time,
        'Burst Time': burst_time,
        'Priority': priority
    })

# Choose scheduling algorithm
algorithm = st.selectbox("Select Scheduling Algorithm:", ['FCFS', 'SJF', 'Priority'])

# Run simulation on button click
if st.button("Run Simulation"):
    # Execute the chosen scheduling algorithm
    if algorithm == "FCFS":
        results = fcfs(processes)
    elif algorithm == "SJF":
        results = sjf(processes)
    elif algorithm == "Priority":
        results = priority_scheduling(processes)

    # Convert results to a DataFrame for better display
    results_df = pd.DataFrame(results)
    st.write(f"### Results for {algorithm} Scheduling")
    st.write(results_df[['Process ID', 'Arrival Time', 'Burst Time', 'Priority', 'Start Time',
                         'Completion Time', 'Waiting Time', 'Turnaround Time']])
```

```python
# Display Gantt chart
fig = visualize_gantt_chart(results, algorithm)
st.pyplot(fig)

# Comparison report
avg_waiting_time = sum(p['Waiting Time'] for p in results) / len(results)
avg_turnaround_time = sum(p['Turnaround Time'] for p in results) / len(results)

st.write("### Comparison Report")
st.write(f"- **Average Waiting Time**: {avg_waiting_time:.2f}")
st.write(f"- **Average Turnaround Time**: {avg_turnaround_time:.2f}")

st.write("""
These metrics help compare the efficiency of different scheduling algorithms
in minimizing waiting and turnaround times.
""")
```

❑ Test:

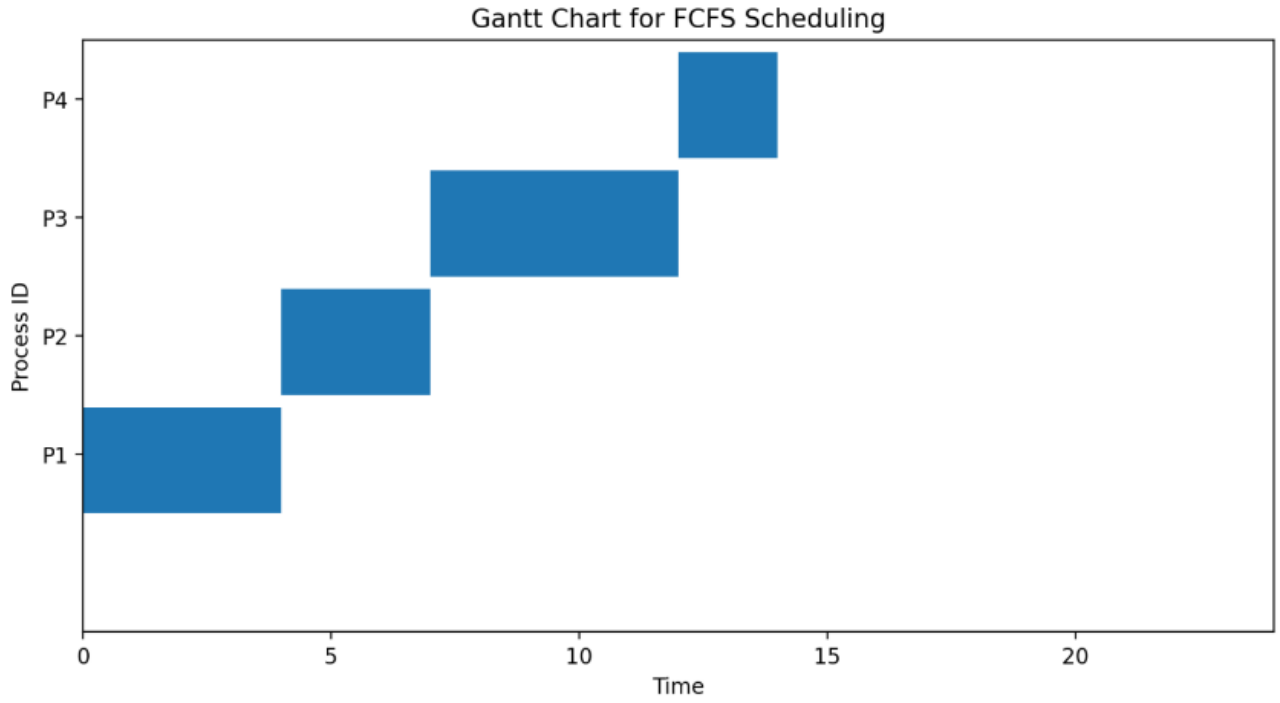
Let's create a test with 4 processes with varying arrival times, burst times, and priorities to see how different algorithms perform:

Process ID	Arrival Time	Burst Time	Priority
P1	0	4	2
P2	1	3	1
P3	2	5	3
P4	3	2	4

❑ Simulation:

Results for FCFS Scheduling

	ID	Arrival Time	Burst Time	Priority	Start Time	Completion Time	Waiting Time	Turnaround Time
0	1	0	4	2	0	4	0	4
1	2	1	3	1	4	7	3	6
2	3	2	5	3	7	12	5	10
3	4	3	2	4	12	14	9	11

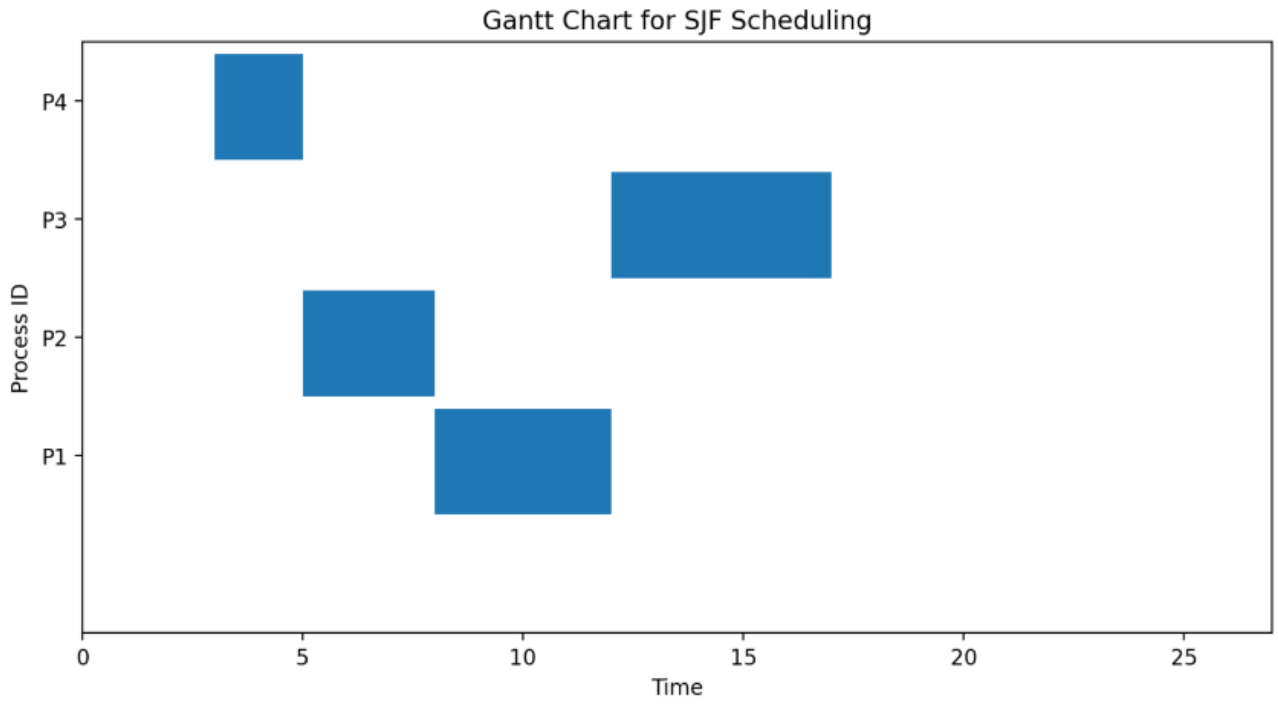


Comparison Report

- **Average Waiting Time:** 4.25
- **Average Turnaround Time:** 7.75

Results for SJF Scheduling

	ID	Arrival Time	Burst Time	Priority	Start Time	Completion Time	Waiting Time	Turnaround Time
0	4	3	2	4	3	5	0	2
1	2	1	3	1	5	8	4	7
2	1	0	4	2	8	12	8	12
3	3	2	5	3	12	17	10	15

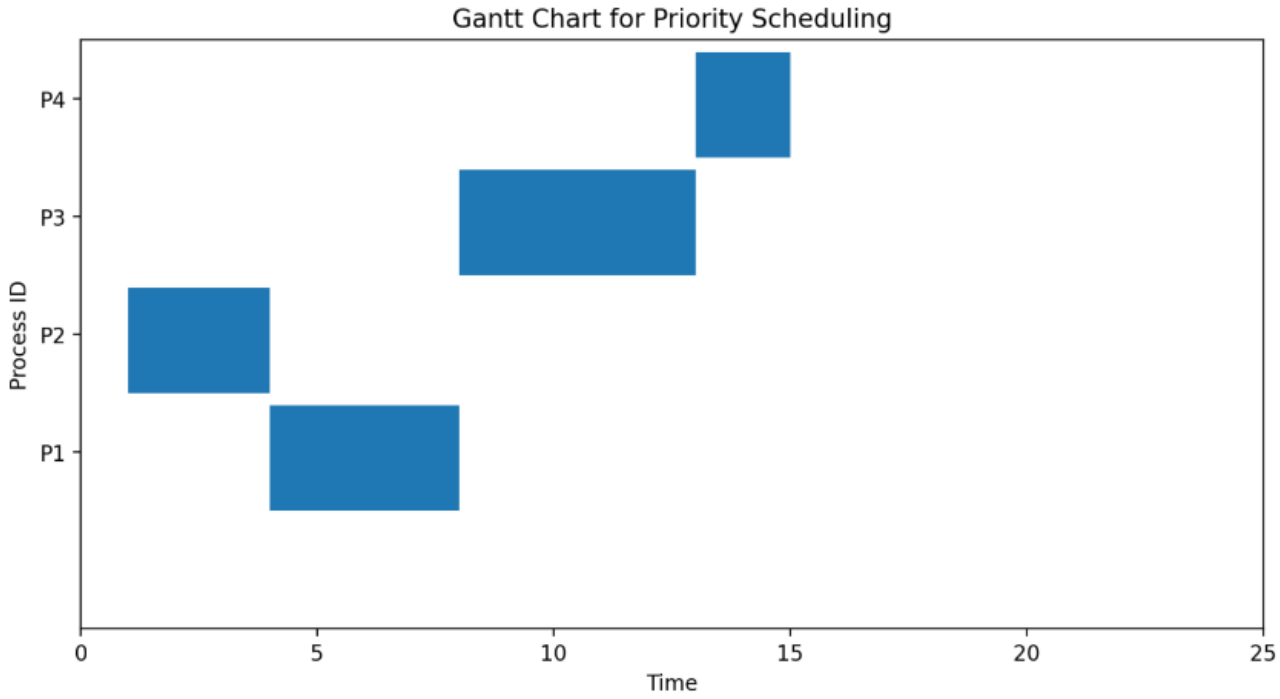


Comparison Report

- **Average Waiting Time:** 5.50
- **Average Turnaround Time:** 9.00

Results for Priority Scheduling

	ID	Arrival Time	Burst Time	Priority	Start Time	Completion Time	Waiting Time	Turnaround Time
0	2	1	3	1	1	4	0	3
1	1	0	4	2	4	8	4	8
2	3	2	5	3	8	13	6	11
3	4	3	2	4	13	15	10	12



Comparison Report

- **Average Waiting Time:** 5.00
- **Average Turnaround Time:** 8.50