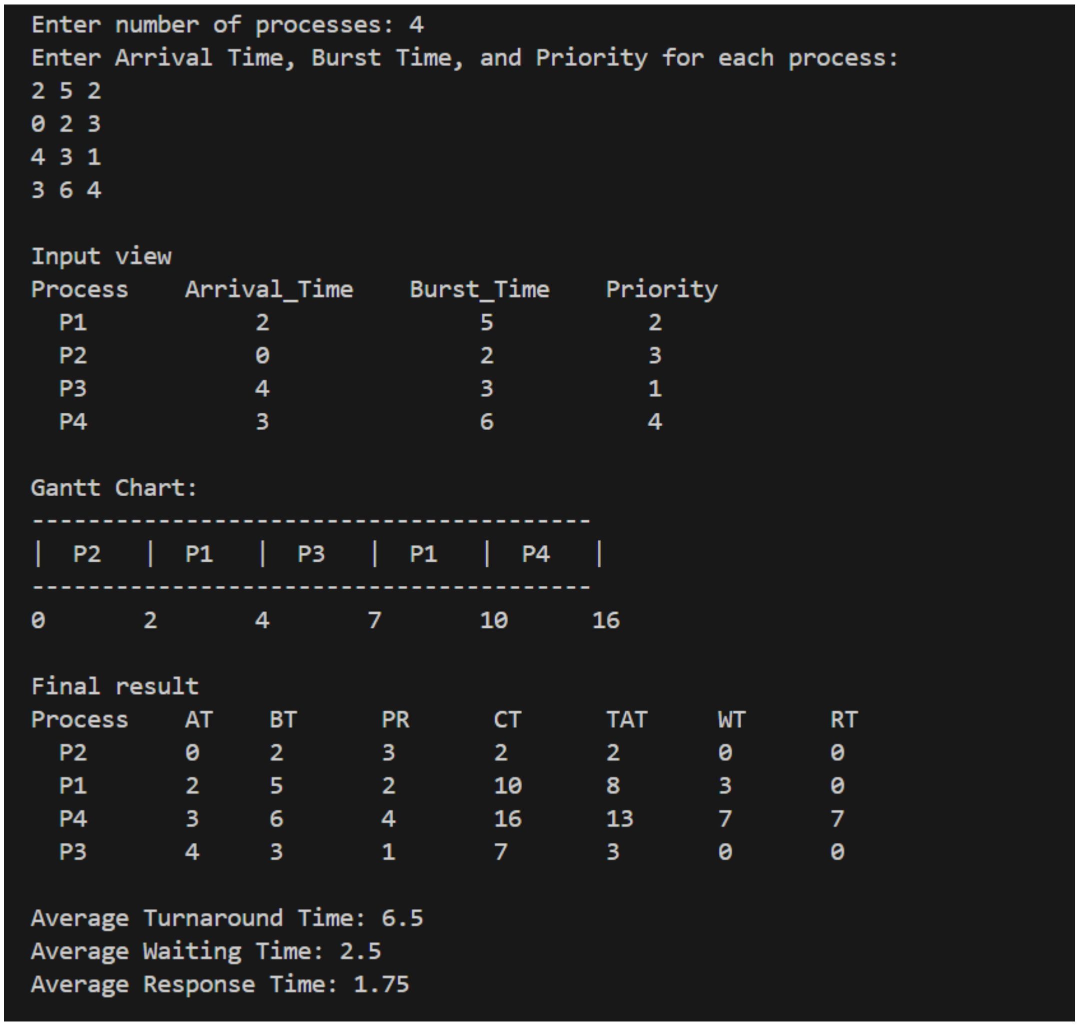
Implementation Code

```
#include <bits/stdc++.h>
using namespace std;
struct Process
{
    int id, AT, BT, priority, remaining_time;
    int CT, TAT, WT, RT;
    bool started = false;
};
vector<Process> processes;
vector<pair<int, int>> gantt_chart;
bool arrival_priority_sort(Process a, Process b)
{
    if (a.AT == b.AT)
        return a.priority < b.priority;
    return a.AT < b.AT;
}
void ganttChart()
{
    cout << "\nGantt Chart:\n";
    int timeline = 0;
    for (int i = 0; i < gantt_chart.size(); i++)
    {
        cout << "--------";
    }
    cout << endl;

    for (auto g : gantt_chart)
    {
        cout << "|  P" << g.first << "    ";
    }
    cout << "|" << endl;
    for (int i = 0; i < gantt_chart.size(); i++)
    {
        cout << "--------";
    }
    cout << endl;
    cout << timeline;
    for (auto g : gantt_chart)
    {
        cout << "\t" << (timeline + g.second);
        timeline += g.second;
    }
    cout << "\n";
```

```cpp
}
void inputView()
{
    cout << "\nInput view" << endl;
    cout << "Process    Arrival_Time    Burst_Time    Priority" << endl;
    for (auto &proc : processes)
    {
        cout << "  P" << proc.id << "\t\t" << proc.AT << "\t\t" << proc.BT << "\t" <<
"     " << proc.priority << endl;
    }
}

int main()
{
    cout << "Enter number of processes: ";
    int n;
    cin >> n;
    processes.resize(n);
    cout << "Enter Arrival Time, Burst Time, and Priority for each process:\n";
    for (int i = 0; i < n; i++)
    {
        processes[i].id = i + 1;
        cin >> processes[i].AT >> processes[i].BT >> processes[i].priority;
        processes[i].remaining_time = processes[i].BT;
    }

    inputView();

    sort(processes.begin(), processes.end(), arrival_priority_sort);
    int time = 0, completed = 0;
    int prev_process = -1, start_time = 0;
    while (completed < n)
    {
        int idx = -1, min_priority = 1e9;
        for (int i = 0; i < n; i++)
        {
            if (processes[i].AT <= time && processes[i].remaining_time > 0)
            {
                if (processes[i].priority < min_priority)
                {
                    min_priority = processes[i].priority;
                    idx = i;
                }
            }
        }
        if (idx != -1)
        {
            if (prev_process != processes[idx].id)
            {
                if (prev_process != -1)
                    gantt_chart.push_back({prev_process, time - start_time});
                start_time = time;
                prev_process = processes[idx].id;
                if (!processes[idx].started)
                {
                    processes[idx].RT = time - processes[idx].AT;
                    processes[idx].started = true;
                }
            }
            processes[idx].remaining_time--;
            time++;

            if (processes[idx].remaining_time == 0)
            {
                completed++;
                processes[idx].CT = time;
                processes[idx].TAT = processes[idx].CT - processes[idx].AT;
                processes[idx].WT = processes[idx].TAT - processes[idx].BT;
            }
        }
        else
        {
            if (prev_process != -1)
                gantt_chart.push_back({prev_process, time - start_time});
            start_time = time;
            prev_process = -1;
            time++;
        }
    }
    if (prev_process != -1)
        gantt_chart.push_back({prev_process, time - start_time});
    // Printing Gantt Chart
    ganttChart();
    // Displaying results
    cout << "\nFinal result\n";
    cout << "Process    AT\t BT\t PR\t CT\t TAT\t WT\t RT\n";
    double total_tat = 0, total_wt = 0, total_rt = 0;
    for (auto p : processes)
    {
        cout << "  P" << p.id << "       " << p.AT << "\t " << p.BT << "\t " <<
p.priority << "\t "
             << p.CT << "\t " << p.TAT << "\t " << p.WT << "\t " << p.RT << "\n";
        total_tat += p.TAT;
        total_wt += p.WT;
        total_rt += p.RT;
    }
    cout << "\nAverage Turnaround Time: " << (total_tat / n);
    cout << "\nAverage Waiting Time: " << (total_wt / n);
    cout << "\nAverage Response Time: " << (total_rt / n) << endl;

    return 0;
}
```

Result Analysis



```
Enter number of processes: 4
Enter Arrival Time, Burst Time, and Priority for each process:
2 5 2
0 2 3
4 3 1
3 6 4

Input view
Process     Arrival_Time     Burst_Time     Priority
  P1             2               5              2
  P2             0               2              3
  P3             4               3              1
  P4             3               6              4

Gantt Chart:
----------------------------------------
|  P2   |  P1   |  P3   |  P1   |  P4   |
----------------------------------------
0       2       4       7       10      16

Final result
Process     AT     BT      PR      CT      TAT     WT      RT
  P2        0      2       3       2       2       0       0
  P1        2      5       2       10      8       3       0
  P4        3      6       4       16      13      7       7
  P3        4      3       1       7       3       0       0

Average Turnaround Time: 6.5
Average Waiting Time: 2.5
Average Response Time: 1.75
```