Assignment – B5

Code

FCFS

```
 #include<iostream>
 using namespace std;

 struct fcfs
 {
     int burst, arrival, id, completion, waiting, turnaround, response;
 };
 fcfs meh[30];

 class FCFS
 {
 public:
     int n;
     void fcfsIn(){
         cout<<"\nEnter number of processes: ";
         cin>>n;
         for(int i = 0; i < n; i++){
             cout<<"\nEnter arrival time of P"<<i<<": ";
             cin>>meh[i].arrival;
             cout<<"\nEnter burst time of P"<<i<<": ";
             cin>>meh[i].burst;
             meh[i].id = i;
         }
         cout<<"\n  | Arrival | Burst\n";
         for(int j = 0; j < n; j++) {
             cout<<"P"<<j<<"| "<<meh[j].arrival<<"       |
"<<meh[j].burst<<"\n";
         }
     }

     void process() {
         cout<<"\nSequence of processes is: ";
         int currentTime = 0;

         for(int i = 0; i < n; i++){
             if(currentTime < meh[i].arrival){
                 while(currentTime < meh[i].arrival){
       cout<<" NULL ";
                     currentTime++;
                 }
             }
             meh[i].response = currentTime - meh[i].arrival;
             cout<<"P"<<meh[i].id<<" ";
             currentTime += meh[i].burst;
             meh[i].completion = currentTime;
             meh[i].turnaround = meh[i].completion - meh[i].arrival;
```

```
            meh[i].waiting = meh[i].turnaround - meh[i].burst;
        }
    }

    void displayMetrics()
    {
        double totalWaiting = 0, totalTurnaround = 0, totalCompletion =
0;

        cout<<"\n\n  | Completion time | Waiting time | Turnaround time
| Response time\n";
        for(int j = 0; j < n; j++) {
            totalWaiting += meh[j].waiting;
            totalTurnaround += meh[j].turnaround;
            totalCompletion += meh[j].completion;
            cout<<"P"<<j<<"| "<<meh[j].completion
                <<"             | "<<meh[j].waiting
                <<"             | "<<meh[j].turnaround
                <<"             | "<<meh[j].response<<"\n";
        }
        cout<<"\nAverage completion time: "<<totalCompletion/n;
        cout<<"\nAverage waiting time: "<<totalWaiting/n;
        cout<<"\nAverage turnaround time: "<<totalTurnaround/n;
    }
};
int main()
{
    FCFS obj;
    obj.fcfsIn();
    obj.process();
    obj.displayMetrics();
    return 0;
}
```

SJF (Preemptive)

```
#include<iostream>
 #include<limits.h> // for INT_MAX
 using namespace std;

 struct sjf{
     int burst, arrival, id, completion, waiting, turnaround, response;
     bool active;
 };

 sjf meh[30];

 class lesgo{
 public:
     int n;
     void sjfIn(){
         cout<<"\nEnter number of processes: ";
         cin>>n;
         for(int i = 1; i <= n; i++){
             cout<<"\nEnter arrival time of P"<<i<<": ";
             cin>>meh[i].arrival;
             cout<<"\nEnter burst time of P"<<i<<": ";
             cin>>meh[i].burst;
             meh[i].id = i;
             meh[i].active = false;
         }
         cout<<"\n  | Arrival | Burst\n";
         for(int j = 1; j <= n; j++) {
             cout<<"P"<<j<<"| "<<meh[j].arrival<<"       |
"<<meh[j].burst<<"\n";
         }
     }
     void sjfProcess(){
         int k = 0;
         int completed = 0;
         cout<<"\nSequence of processes is: ";
         while(completed < n){
             int burst1 = INT_MAX; // Initialize to the maximum possible
value
             int iddd = -1;
             for(int i = 1; i <= n; i++){
                 if(meh[i].arrival <= k && meh[i].burst > 0){
if(meh[i].burst < burst1){
                         burst1 = meh[i].burst;
                         iddd = i;
                     }
                 }
             }            if(iddd != -1){
                 // Mark the process as active
                 if(!meh[iddd].active) {
                    meh[iddd].response = k - meh[iddd].arrival;
                     meh[iddd].active = true;
                 }
                 cout<<"P"<<iddd<<" ";
                 meh[iddd].burst--;
                 k++;
                 if(meh[iddd].burst == 0){
                     meh[iddd].completion = k;
                     meh[iddd].turnaround = meh[iddd].completion -
meh[iddd].arrival;
                     meh[iddd].waiting = meh[iddd].turnaround -
(meh[iddd].response + 1); // +1 for the final unit burst time
                     completed++;
                 }
             } else{
                 k++;
             }
         }
     }
     void displayMetrics(){
         double totalWaiting = 0, totalTurnaround = 0, totalCompletion =
0;
```

```
        cout<<"\n\n  | Completion time | Waiting time | Turnaround time
| Response time\n";
        for(int j = 1; j <= n; j++) {
            totalWaiting += meh[j].waiting;
            totalTurnaround += meh[j].turnaround;
            totalCompletion += meh[j].completion;
            cout<<"P"<<j<<"| "<<meh[j].completion
                <<"             | "<<meh[j].waiting
                <<"             | "<<meh[j].turnaround
                <<"             | "<<meh[j].response<<"\n";
        }
        cout<<"\nAverage completion time: "<<totalCompletion/n;
        cout<<"\nAverage waiting time: "<<totalWaiting/n;
        cout<<"\nAverage turnaround time: "<<totalTurnaround/n;
    }
};

int main(){
    lesgo obj;
    obj.sjfIn();
    obj.sjfProcess();
    obj.displayMetrics();
    return 0;
}
```

Priority (Non-Preemptive)

```
#include<iostream>
 #include<limits.h> // for INT_MAX
 using namespace std;

 struct sjf {
     int burst, arrival, id, completion, waiting, turnaround, response,
priority;
     bool active;
 };
 sjf meh[30];
 class lesgo {
 public:
     int n;
     void priorityIn() {
         cout << "\nEnter number of processes: ";
         cin >> n;
         for (int i = 1; i <= n; i++) {
             cout << "\nEnter arrival time of P" << i << ": ";
             cin >> meh[i].arrival;
             cout << "\nEnter burst time of P" << i << ": ";
             cin >> meh[i].burst;
             cout << "\nEnter priority of P" << i << ": ";
             cin >> meh[i].priority;
             meh[i].id = i;
             meh[i].active = false;
         }
```

```
        cout << "\n  | Arrival | Burst | Priority\n";
        for (int j = 1; j <= n; j++) {
            cout << "P" << j << " | " << meh[j].arrival << "       | "
<< meh[j].burst << " | " << meh[j].priority << "\n";
        }
    }

    void priorityProcess() {
        int k = 0; // Current time
        int completed = 0; // Number of completed processes

        while (completed < n) {
            int highestPriority = INT_MAX;
            int selectedProcess = -1;

            // Find the process with the highest priority (smallest
priority number) that has arrived and is not completed
            for (int i = 1; i <= n; i++) {
                if (meh[i].arrival <= k && !meh[i].active &&
meh[i].priority < highestPriority) {
                    highestPriority = meh[i].priority;
                    selectedProcess = i;
                }
           }
            if (selectedProcess != -1) {
                // Mark the process as active
                meh[selectedProcess].active = true;
                // If the process is starting now, calculate response
time
                if (meh[selectedProcess].response == 0) {
                    meh[selectedProcess].response = k -
meh[selectedProcess].arrival;
                }
                // Execute the process
                k += meh[selectedProcess].burst;
                meh[selectedProcess].completion = k;
                meh[selectedProcess].turnaround =
meh[selectedProcess].completion - meh[selectedProcess].arrival;
                meh[selectedProcess].waiting =
meh[selectedProcess].turnaround - meh[selectedProcess].burst;
                completed++;
            } else {
                // If no process is ready to run, just increment time
                k++;
            }
        }
    }
    void displayMetrics() {
        double totalWaiting = 0, totalTurnaround = 0, totalCompletion =
0;
        cout << "\n\n  | Completion time | Waiting time | Turnaround
time | Response time\n";
        for (int j = 1; j <= n; j++) {
            totalWaiting += meh[j].waiting;
            totalTurnaround += meh[j].turnaround;
            totalCompletion += meh[j].completion;
            cout << "P" << j << " | " << meh[j].completion
                 << "             | " << meh[j].waiting
                 << "             | " << meh[j].turnaround
                 << "             | " << meh[j].response << "\n";
        }
        cout << "\nAverage completion time: " << totalCompletion / n;
        cout << "\nAverage waiting time: " << totalWaiting / n;
        cout << "\nAverage turnaround time: " << totalTurnaround / n;
    }
};
int main() {
    lesgo obj;
    obj.priorityIn();
    obj.priorityProcess();
    obj.displayMetrics();
    return 0;
}
```

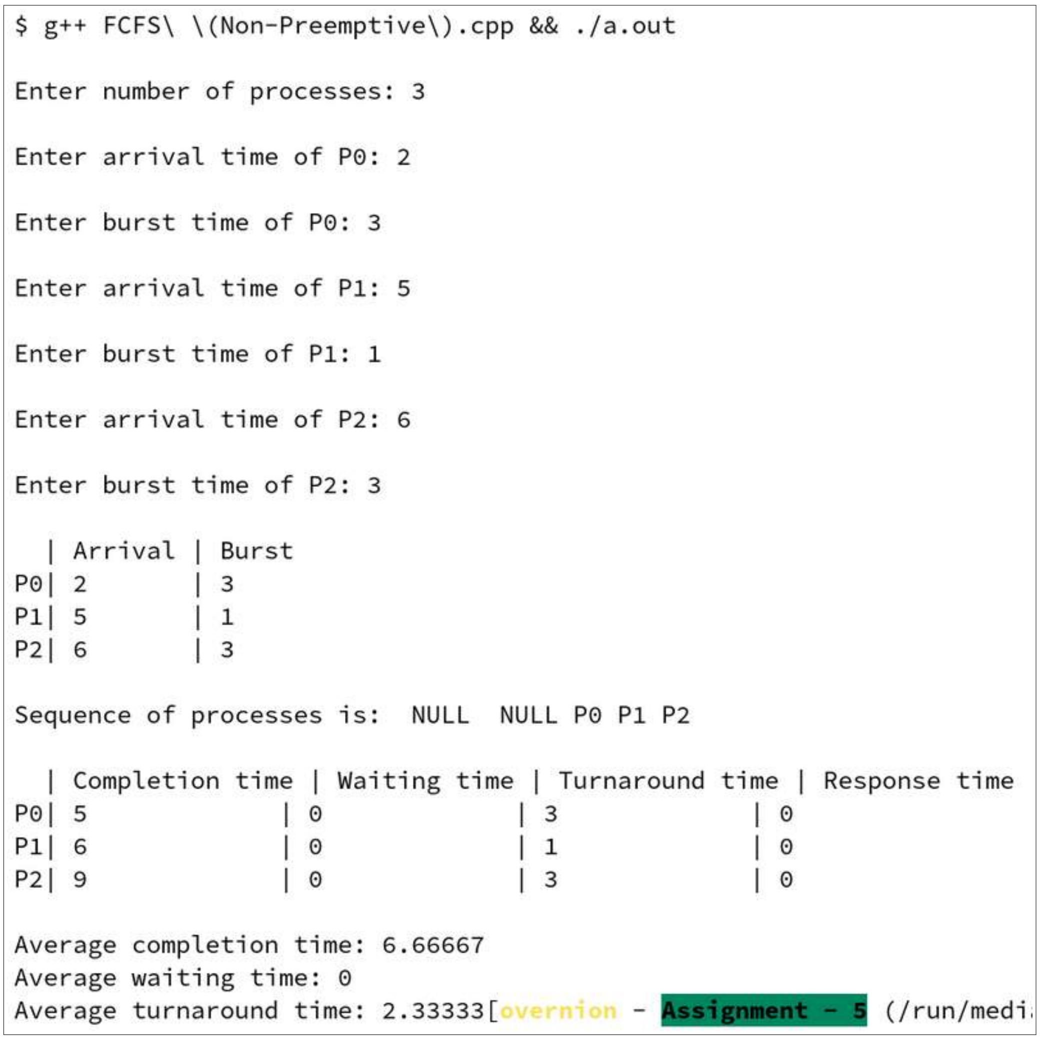
Round Robin (Preemptive)

```
#include <iostream>
 #include <queue>
 #include <vector>
 #include <iomanip> // For formatting output
 using namespace std;

 struct process {
     int burst, arrival, id, completion, priority, waiting, turnaround,
response, remainingBurst;
     bool active;
 };

 process meh[30];

 class RoundRobin {
 public:
     int n;
     int timeQuantum;

     void inputProcesses() {
         cout << "\nEnter number of processes: ";
         cin >> n;
         for (int i = 1; i <= n; i++) {
             cout << "\nEnter arrival time of P" << i << ": ";
             cin >> meh[i].arrival;
             cout << "\nEnter burst time of P" << i << ": ";
             cin >> meh[i].burst;
             cout << "\nEnter priority of P" << i << ": ";
             cin >> meh[i].priority; // Priority is not used in RR, but
kept here for completeness.
             meh[i].id = i;
```

```
            meh[i].remainingBurst = meh[i].burst;
            meh[i].active = false;
        }
        cout << "\nEnter time quantum: ";
        cin >> timeQuantum;
        cout << "\n  | Arrival | Burst | Priority\n";
        for (int j = 1; j <= n; j++) {
            cout << "P" << j << " | " << meh[j].arrival << "       | "
<< meh[j].burst << " | " << meh[j].priority << "\n";
        }
    }
    void roundRobinProcess() {
        int k = 0; // Current time
        int completed = 0; // Number of completed processes
        queue<int> readyQueue;
        vector<bool> isProcessed(n + 1, false); // Track whether a
process has been added to the ready queue
        while (completed < n) {
           // Add processes that have arrived to the ready queue
            for (int i = 1; i <= n; i++) {
                if (meh[i].arrival <= k && !isProcessed[i]) {
                    readyQueue.push(i);
                    isProcessed[i] = true;
                }
            }
            if (readyQueue.empty()) {
                // If no process is in the queue, increment time
                k++;
                continue;
            }
            int currentProcess = readyQueue.front();
            readyQueue.pop();
            // Calculate response time for the process if it starts now
            if (!meh[currentProcess].active) {
                meh[currentProcess].response = k -
meh[currentProcess].arrival;
                meh[currentProcess].active = true;
            }
            int timeSlice = min(timeQuantum,
meh[currentProcess].remainingBurst);
            // Process the current process
            meh[currentProcess].remainingBurst -= timeSlice;
            k += timeSlice;
            if (meh[currentProcess].remainingBurst == 0) {
                meh[currentProcess].completion = k;
                meh[currentProcess].turnaround =
meh[currentProcess].completion - meh[currentProcess].arrival;
                meh[currentProcess].waiting =
meh[currentProcess].turnaround - meh[currentProcess].burst;
                completed++;
            } else {
                // If the process is not finished, re-add it to the
queue
                readyQueue.push(currentProcess);
            }
        }
    }
    void displayMetrics() {
        double totalWaiting = 0, totalTurnaround = 0, totalCompletion =
0;

        cout << "\n\n  | Completion time | Waiting time | Turnaround
time | Response time\n";
        for (int j = 1; j <= n; j++) {
            totalWaiting += meh[j].waiting;
            totalTurnaround += meh[j].turnaround;
            totalCompletion += meh[j].completion;
            cout << "P" << j << " | " << setw(15) << meh[j].completion
                 << " | " << setw(12) << meh[j].waiting
                << " | " << setw(15) << meh[j].turnaround
                 << " | " << setw(12) << meh[j].response << "\n";
        }

        cout << "\nAverage completion time: " << totalCompletion / n;
        cout << "\nAverage waiting time: " << totalWaiting / n;
        cout << "\nAverage turnaround time: " << totalTurnaround / n;
    }
 };

 int main() {
    RoundRobin obj;
    obj.inputProcesses();
    obj.roundRobinProcess();
    obj.displayMetrics();
    return 0;
 }
```

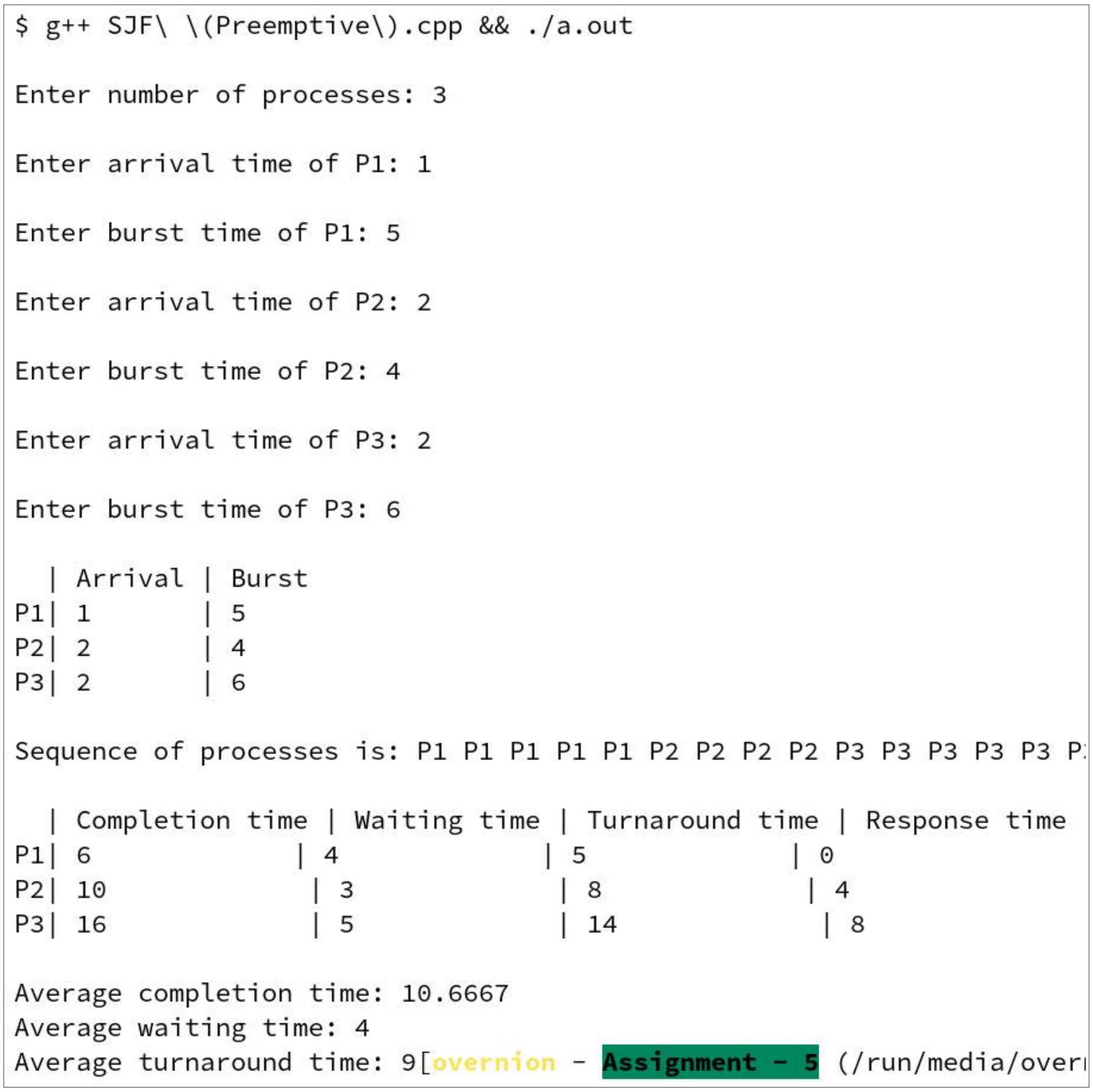
Output

FCFS



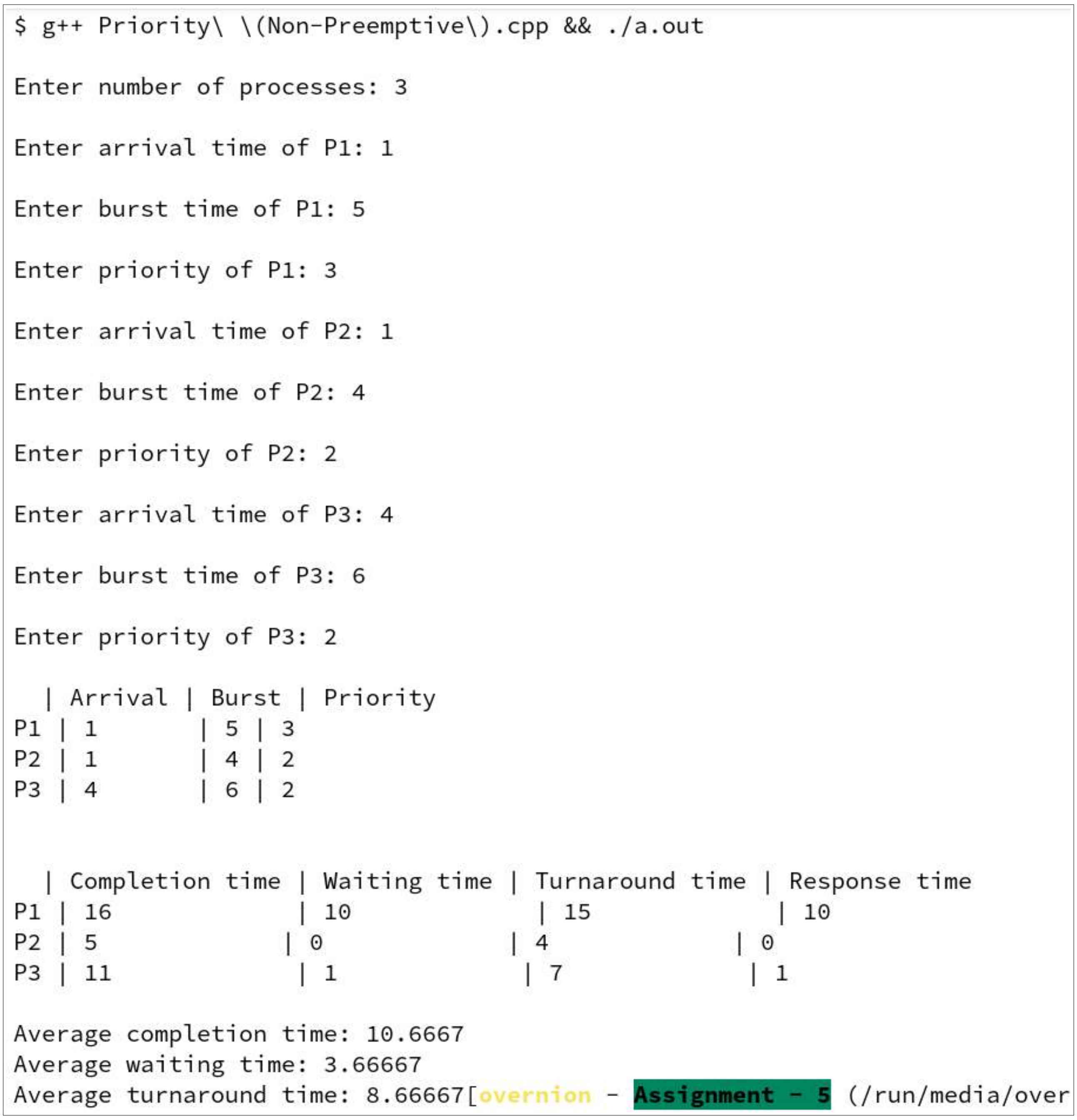
```
$ g++ FCFS\ \(Non-Preemptive\).cpp && ./a.out

Enter number of processes: 3

Enter arrival time of P0: 2

Enter burst time of P0: 3

Enter arrival time of P1: 5

Enter burst time of P1: 1

Enter arrival time of P2: 6

Enter burst time of P2: 3

   | Arrival | Burst
P0| 2        | 3
P1| 5        | 1
P2| 6        | 3

Sequence of processes is:  NULL  NULL P0 P1 P2

   | Completion time | Waiting time | Turnaround time | Response time
P0| 5                | 0             | 3                | 0
P1| 6                | 0             | 1                | 0
P2| 9                | 0             | 3                | 0

Average completion time: 6.66667
Average waiting time: 0
Average turnaround time: 2.33333[overnion - Assignment - 5 (/run/medi
```

SJF (Preemptive)



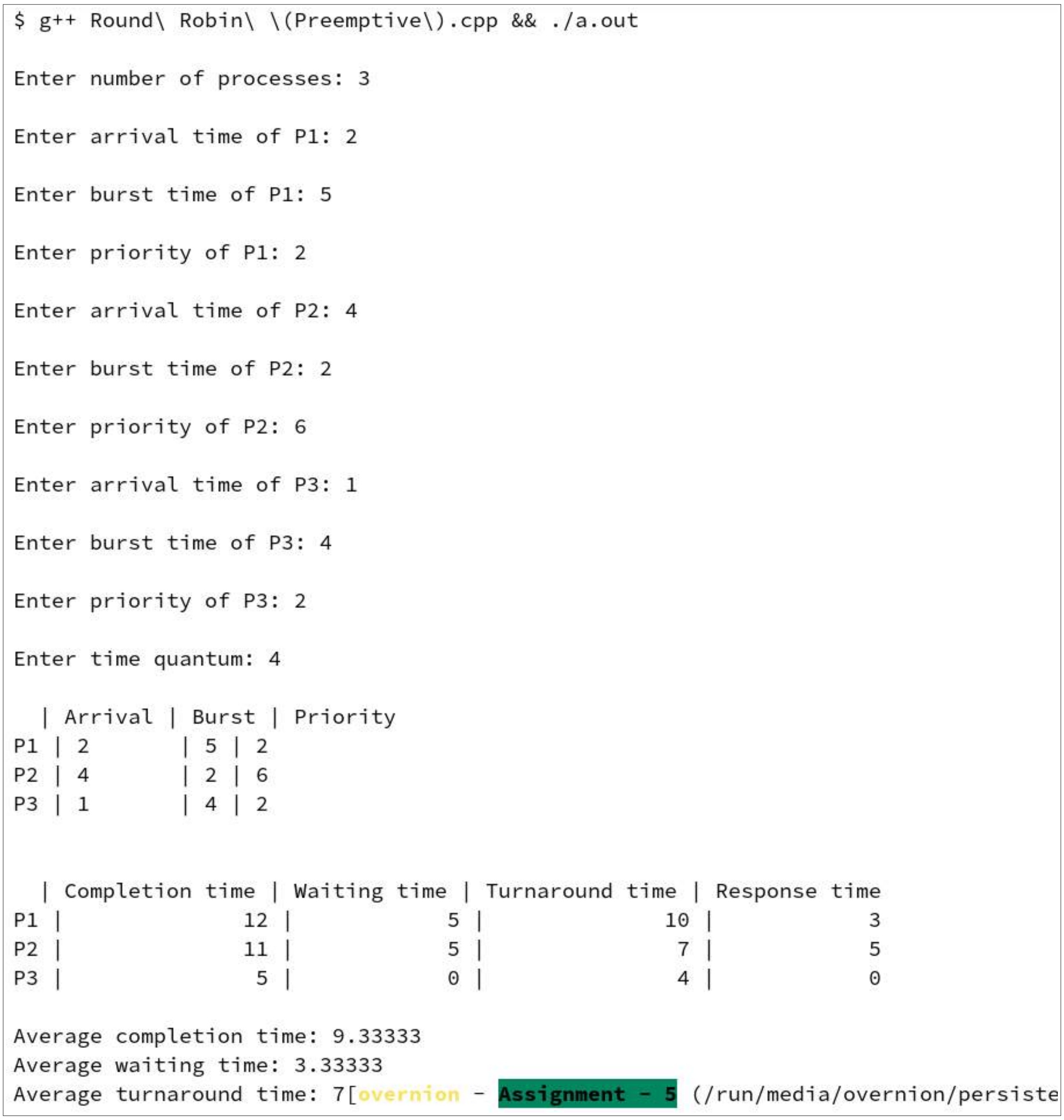
```
$ g++ SJF\ \(Preemptive\).cpp && ./a.out

Enter number of processes: 3

Enter arrival time of P1: 1

Enter burst time of P1: 5

Enter arrival time of P2: 2

Enter burst time of P2: 4

Enter arrival time of P3: 2

Enter burst time of P3: 6

  | Arrival | Burst
P1| 1       | 5
P2| 2       | 4
P3| 2       | 6

Sequence of processes is: P1 P1 P1 P1 P1 P2 P2 P2 P2 P3 P3 P3 P3 P3 P

  | Completion time | Waiting time | Turnaround time | Response time
P1| 6              | 4             | 5              | 0
P2| 10              | 3             | 8              | 4
P3| 16              | 5             | 14              | 8

Average completion time: 10.6667
Average waiting time: 4
Average turnaround time: 9[overnion - Assignment - 5 (/run/media/over
```

Priority (Non-Preemptive)



```
$ g++ Priority\ \(Non-Preemptive\).cpp && ./a.out

Enter number of processes: 3

Enter arrival time of P1: 1

Enter burst time of P1: 5

Enter priority of P1: 3

Enter arrival time of P2: 1

Enter burst time of P2: 4

Enter priority of P2: 2

Enter arrival time of P3: 4

Enter burst time of P3: 6

Enter priority of P3: 2

  | Arrival | Burst | Priority
P1 | 1        | 5 | 3
P2 | 1        | 4 | 2
P3 | 4        | 6 | 2


  | Completion time | Waiting time | Turnaround time | Response time
P1 | 16                | 10               | 15                | 10
P2 | 5                | 0               | 4                | 0
P3 | 11                | 1               | 7                | 1

Average completion time: 10.6667
Average waiting time: 3.66667
Average turnaround time: 8.66667[overnion - Assignment - 5 (/run/media/over
```

Round Robin (Preemptive)



```
$ g++ Round\ Robin\ \(Preemptive\).cpp && ./a.out

Enter number of processes: 3

Enter arrival time of P1: 2

Enter burst time of P1: 5

Enter priority of P1: 2

Enter arrival time of P2: 4

Enter burst time of P2: 2

Enter priority of P2: 6

Enter arrival time of P3: 1

Enter burst time of P3: 4

Enter priority of P3: 2

Enter time quantum: 4

   | Arrival | Burst | Priority
P1 | 2       | 5 | 2
P2 | 4       | 2 | 6
P3 | 1       | 4 | 2


   | Completion time | Waiting time | Turnaround time | Response time
P1 |              12 |            5 |              10 |             3
P2 |              11 |            5 |               7 |             5
P3 |               5 |            0 |               4 |             0

Average completion time: 9.33333
Average waiting time: 3.33333
Average turnaround time: 7[overnion - Assignment - 5 (/run/media/overnion/persiste
```