Name : Md Abdur Rahman

ID : 213-15-4343

SEC : 60_D

CPU SCHEDULING (CODE WITH INPUT AND OUTPUT)

1.First Come First Serve(FCFS):

CODE:

```
#include <bits/stdc++.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

bool compareArrival(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}
```

```
bool compareID(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;

    for(int i = 0; i < n; i++) {
        cout<<"Enter arrival time of process "<<i+1<<": ";
        cin>>p[i].arrival_time;
```

```
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

sort(p,p+n,compareArrival);

for(int i = 0; i < n; i++) {
    p[i].start_time = (i == 0)?p[i].arrival_time:max(p[i-1].completion_time,p[i].arrival_time);
    p[i].completion_time = p[i].start_time + p[i].burst_time;
    p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
    p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
    p[i].response_time = p[i].start_time - p[i].arrival_time;

    total_turnaround_time += p[i].turnaround_time;
    total_waiting_time += p[i].waiting_time;
    total_response_time += p[i].response_time;
    total_idle_time += (i == 0)?(p[i].arrival_time):(p[i].start_time - p[i-1].completion_time);
}

avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;
avg_response_time = (float) total_response_time / n;
throughput = float(n) / (p[n-1].completion_time - p[0].arrival_time);

sort(p,p+n,compareID);
```

```
    cout<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"<<
p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].respo
nse_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;



}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:
```

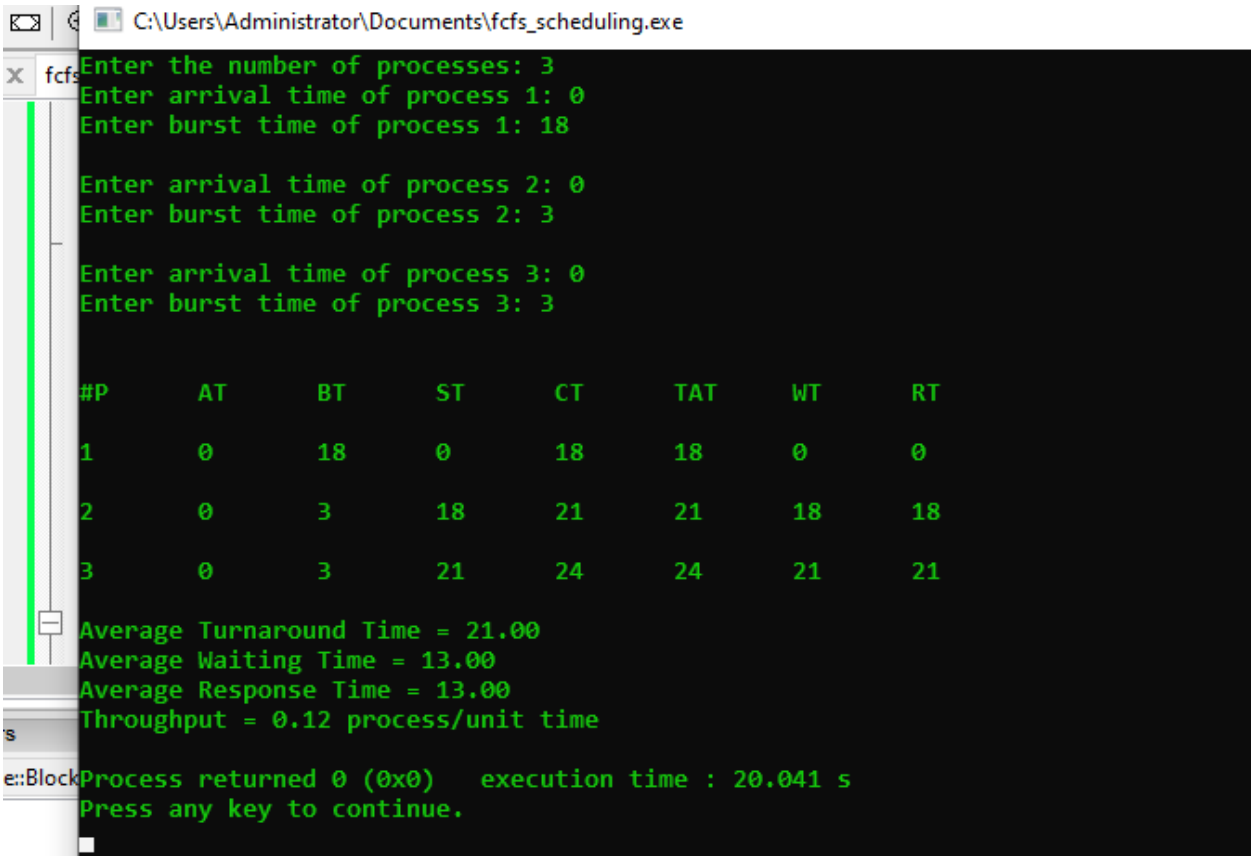
TAT = CT - AT

WT = TAT - BT

RT = ST - AT

*/

INPUT AND OUTPUT:



2.Shortest Job First Premitive (SJF):

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int is_completed[100];
    memset(is_completed,0,sizeof(is_completed));
```

```
cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    p[i].pid = i+1;
    burst_remaining[i] = p[i].burst_time;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mn = 10000000;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(burst_remaining[i] < mn) {
                mn = burst_remaining[i];
                idx = i;
            }
```

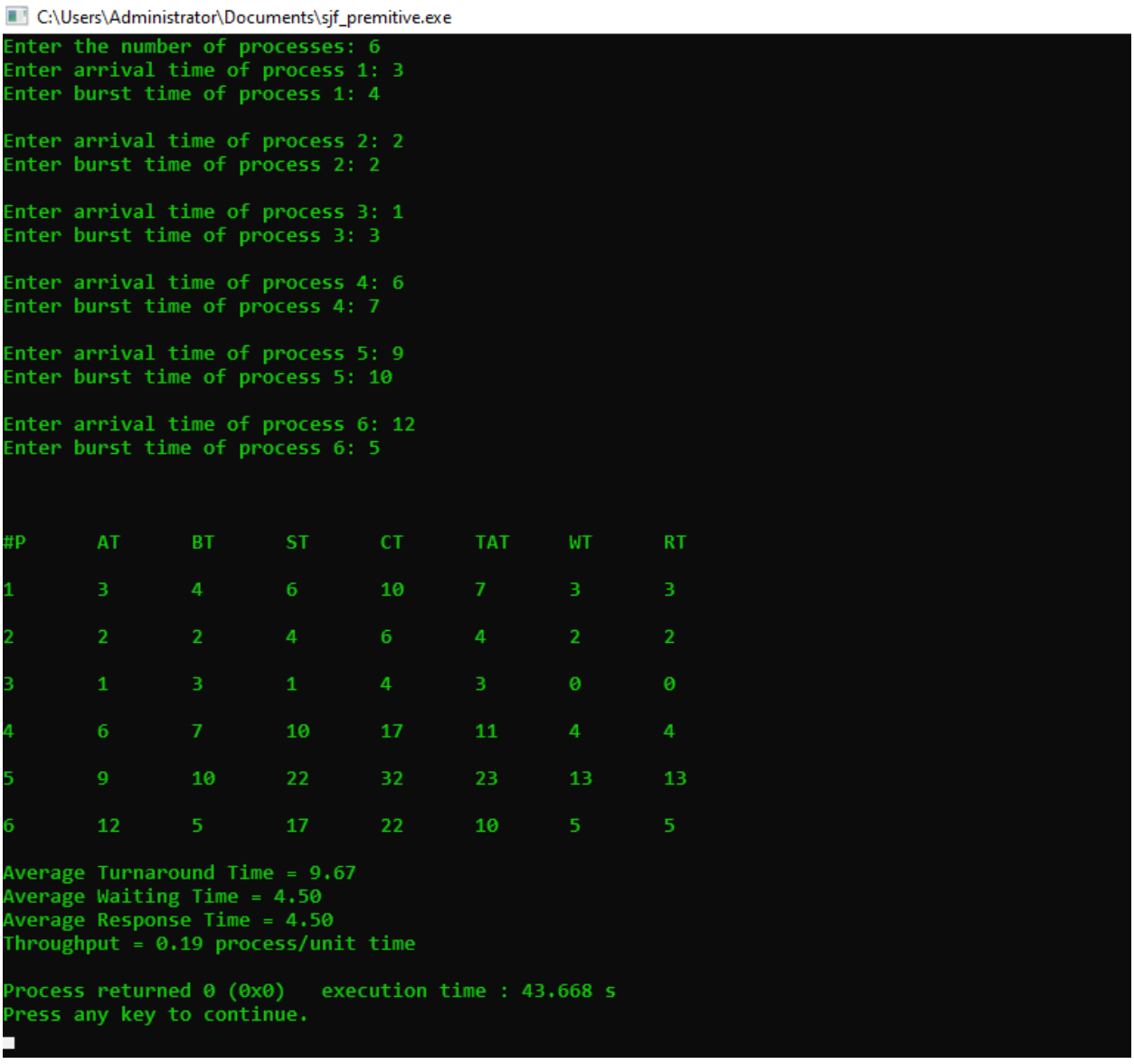
```
        if(burst_remaining[i] == mn) {
            if(p[i].arrival_time < p[idx].arrival_time) {
                mn = burst_remaining[i];
                idx = i;
            }
        }
    }
}

if(idx != -1) {
    if(burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = current_time;
        total_idle_time += p[idx].start_time - prev;
    }
    burst_remaining[idx] -= 1;
    current_time++;
    prev = current_time;

    if(burst_remaining[idx] == 0) {
        p[idx].completion_time = current_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;
```

```
            is_completed[idx] = 1;
            completed++;
          }
        }
        else {
            current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time
)*100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout<<endl<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {
```

```
cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"<<
p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].respo
nse_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;



}


/*


AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process


Formulas used:


TAT = CT - AT
WT = TAT - BT
RT = ST - AT
```

*/

INPUT AND OUTPUT:



3.Shortest Job First Nonpremitive (SJF):

```
#include <bits/stdc++.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
```

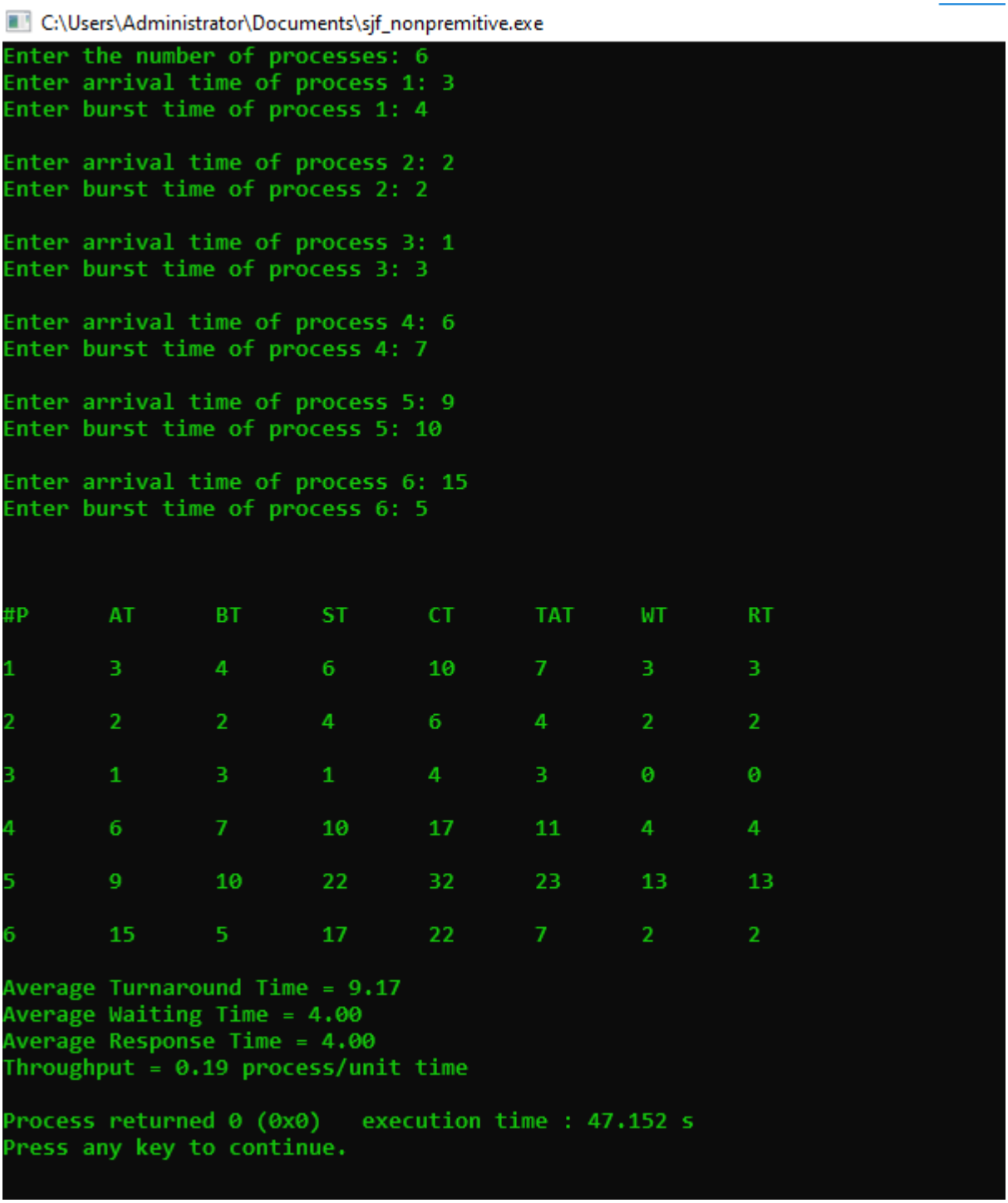
```
float throughput;
int is_completed[100];
memset(is_completed,0,sizeof(is_completed));

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mn = 10000000;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].burst_time < mn) {
```

```
            mn = p[i].burst_time;
            idx = i;
        }
        if(p[i].burst_time == mn) {
            if(p[i].arrival_time < p[idx].arrival_time) {
                mn = p[i].burst_time;
                idx = i;
            }
        }
    }
}
if(idx != -1) {
    p[idx].start_time = current_time;
    p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
    p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
    p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
    p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

    total_turnaround_time += p[idx].turnaround_time;
    total_waiting_time += p[idx].waiting_time;
    total_response_time += p[idx].response_time;
    total_idle_time += p[idx].start_time - prev;

    is_completed[idx] = 1;
    completed++;
    current_time = p[idx].completion_time;
    prev = current_time;
}
```

```
        else {
            current_time++;
        }

    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time
)*100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout<<endl<<endl;


cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"<<
p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].respo
nse_time<<"\t"<<"\n"<<endl;
```

```
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;


}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

INPUT AND OUTPUT:



4. Nonpremitive Priority Scheduling:

```
#include <bits/stdc++.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
```

```
int total_idle_time = 0;
float throughput;
int is_completed[100];
memset(is_completed,0,sizeof(is_completed));

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    cout<<"Enter priority of the process "<<i+1<<": ";
    cin>>p[i].priority;
    p[i].pid = i+1;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mx = -1;
```

```
for(int i = 0; i < n; i++) {
    if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
        if(p[i].priority > mx) {
            mx = p[i].priority;
            idx = i;
        }
        if(p[i].priority == mx) {
            if(p[i].arrival_time < p[idx].arrival_time) {
                mx = p[i].priority;
                idx = i;
            }
        }
    }
}
if(idx != -1) {
    p[idx].start_time = current_time;
    p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
    p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
    p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
    p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

    total_turnaround_time += p[idx].turnaround_time;
    total_waiting_time += p[idx].waiting_time;
    total_response_time += p[idx].response_time;
    total_idle_time += p[idx].start_time - prev;

    is_completed[idx] = 1;
    completed++;
```

```
            current_time = p[idx].completion_time;
            prev = current_time;
        }
        else {
            current_time++;
        }

    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout<<endl<<endl;


cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n
"<<endl;

    for(int i = 0; i < n; i++) {
```

```
cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].priority<<"\t"<<p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;



}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT
```

*/

INPUT AND OUTPUT:

```
C:\Users\Administrator\Documents\nonpremitive_priority_scheduling.exe
Enter the number of processes: 5
Enter arrival time of process 1: 0
Enter burst time of process 1: 3
Enter priority of the process 1: 3

Enter arrival time of process 2: 2
Enter burst time of process 2: 3
Enter priority of the process 2: 1

Enter arrival time of process 3: 3
Enter burst time of process 3: 7
Enter priority of the process 3: 4

Enter arrival time of process 4: 4
Enter burst time of process 4: 4
Enter priority of the process 4: 5

Enter arrival time of process 5: 1
Enter burst time of process 5: 2
Enter priority of the process 5: 2



#P      AT      BT      PRI     ST      CT      TAT     WT      RT

1       0       3       3       0       3       3       0       0

2       2       3       1       16      19      17      14      14

3       3       7       4       3       10      7       0       0

4       4       4       5       10      14      10      6       6

5       1       2       2       14      16      15      13      13

Average Turnaround Time = 10.40
Average Waiting Time = 6.60
Average Response Time = 6.60
Throughput = 0.26 process/unit time

Process returned 0 (0x0)   execution time : 48.664 s
Press any key to continue.
```

5. Premitive Priority Scheduling:

```
#include <bits/stdc++.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
```

```
int total_response_time = 0;
int total_idle_time = 0;
float throughput;
int burst_remaining[100];
int is_completed[100];
memset(is_completed,0,sizeof(is_completed));

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    cout<<"Enter priority of the process "<<i+1<<": ";
    cin>>p[i].priority;
    p[i].pid = i+1;
    burst_remaining[i] = p[i].burst_time;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;
```

```
while(completed != n) {
    int idx = -1;
    int mx = -1;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].priority > mx) {
                mx = p[i].priority;
                idx = i;
            }
            if(p[i].priority == mx) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mx = p[i].priority;
                    idx = i;
                }
            }
        }
    }

    if(idx != -1) {
        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time;
            total_idle_time += p[idx].start_time - prev;
        }
        burst_remaining[idx] -= 1;
        current_time++;
        prev = current_time;

        if(burst_remaining[idx] == 0) {
```

```
            p[idx].completion_time = current_time;
            p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
            p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;

            is_completed[idx] = 1;
            completed++;
        }
    }
    else {
        current_time++;
    }
}

int min_arrival_time = 10000000;
int max_completion_time = -1;
for(int i = 0; i < n; i++) {
    min_arrival_time = min(min_arrival_time,p[i].arrival_time);
    max_completion_time = max(max_completion_time,p[i].completion_time);
}

avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;
avg_response_time = (float) total_response_time / n;
```

```
    throughput = float(n) / (max_completion_time - min_arrival_time);


    cout<<endl<<endl;



cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n
"<<endl;


    for(int i = 0; i < n; i++) {


cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].priority<<"\t"<<p[i
].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_tim
e<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;



}


/*


AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
```

RT - Response time of the process

Formulas used:

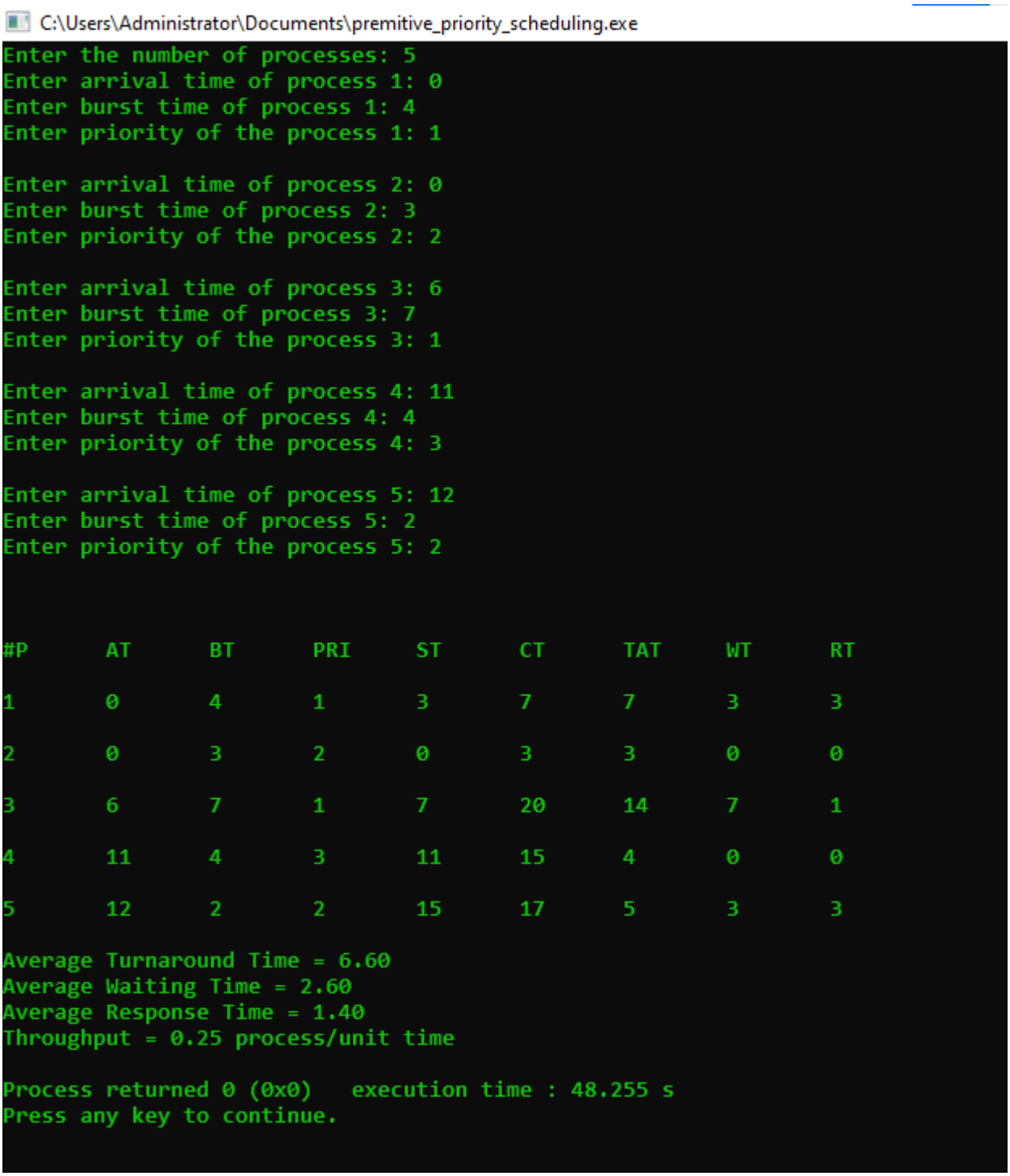
TAT = CT - AT

WT = TAT - BT

RT = ST - AT

*/

INPUT AND OUTPUT:



6. Round Robin Scheduling:

```
#include <bits/stdc++.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

bool compare1(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
    return p1.pid < p2.pid;
}
```

```
int main() {

    int n;
    int tq;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int idx;

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;
    cout<<"Enter time quantum: ";
    cin>>tq;

    for(int i = 0; i < n; i++) {
        cout<<"Enter arrival time of process "<<i+1<<": ";
        cin>>p[i].arrival_time;
        cout<<"Enter burst time of process "<<i+1<<": ";
```

```
    cin>>p[i].burst_time;
    burst_remaining[i] = p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

sort(p,p+n,compare1);

queue<int> q;
int current_time = 0;
q.push(0);
int completed = 0;
int mark[100];
memset(mark,0,sizeof(mark));
mark[0] = 1;

while(completed != n) {
    idx = q.front();
    q.pop();

    if(burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = max(current_time,p[idx].arrival_time);
        total_idle_time += p[idx].start_time - current_time;
        current_time = p[idx].start_time;
    }

    if(burst_remaining[idx]-tq > 0) {
        burst_remaining[idx] -= tq;
```

```
    current_time += tq;
}
else {
    current_time += burst_remaining[idx];
    burst_remaining[idx] = 0;
    completed++;

    p[idx].completion_time = current_time;
    p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
    p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
    p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

    total_turnaround_time += p[idx].turnaround_time;
    total_waiting_time += p[idx].waiting_time;
    total_response_time += p[idx].response_time;
}

for(int i = 1; i < n; i++) {
    if(burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i] == 0) {
        q.push(i);
        mark[i] = 1;
    }
}
if(burst_remaining[idx] > 0) {
    q.push(idx);
}

if(q.empty()) {
```

```
        for(int i = 1; i < n; i++) {
            if(burst_remaining[i] > 0) {
                q.push(i);
                mark[i] = 1;
                break;
            }
        }
      }



    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    throughput = float(n) / (p[n-1].completion_time - p[0].arrival_time);

    sort(p,p+n,compare2);

    cout<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"<<
p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].respo
nse_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
```

```
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;


}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

INPUT AND OUTPUT: