**Practical No:-1**

**Aim :- Implement the following CPU scheduling algorithm.**

1. **:FCFS (First Come First Serve)**
2. **SJF (Preemptive and Non Preemptive)**

**Name : Anas Ahmad Ilyas Ahmad Subject:OS Batch : 1**

**Class : MCA-Sem I Roll No: 222010001 Date:-12/12/2022**

1. **FCFS(First Come First Serve):**

**Source code:**

#include <bits/stdc++.h>

using namespace std;

class Process{

public:

int p\_id;

int arrival\_time;

int burst\_time;

int start\_time; // first time cpu allotment time

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

bool compareArrivalTime(Process p1, Process p2){

if (p1.arrival\_time < p2.arrival\_time){

return true;

}else{

return false;

}

}

bool compareID(Process p1, Process p2){

if (p1.p\_id < p2.p\_id) {

return true;

}else{

return false;

}

}

int main(){

int n;

cout << "Enter number of processes:";

cin >> n;

Process p[n];

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

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].p\_id = i + 1;

cout << endl;

}

sort(p, p + n, compareArrivalTime);

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

if (i == 0){

p[i].start\_time = p[i].arrival\_time;

}else{

p[i].start\_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 = total\_turnaround\_time + p[i].turnaround\_time;

total\_waiting\_time = total\_waiting\_time + p[i].waiting\_time;

total\_response\_time = total\_response\_time + p[i].response\_time;

}

sort(p, p + n, compareID);

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].p\_id << "\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].response\_time << "\t"<< "\n"<<endl;

}

cout << "Average Turnaround Time: " << (float)total\_turnaround\_time / n << endl;

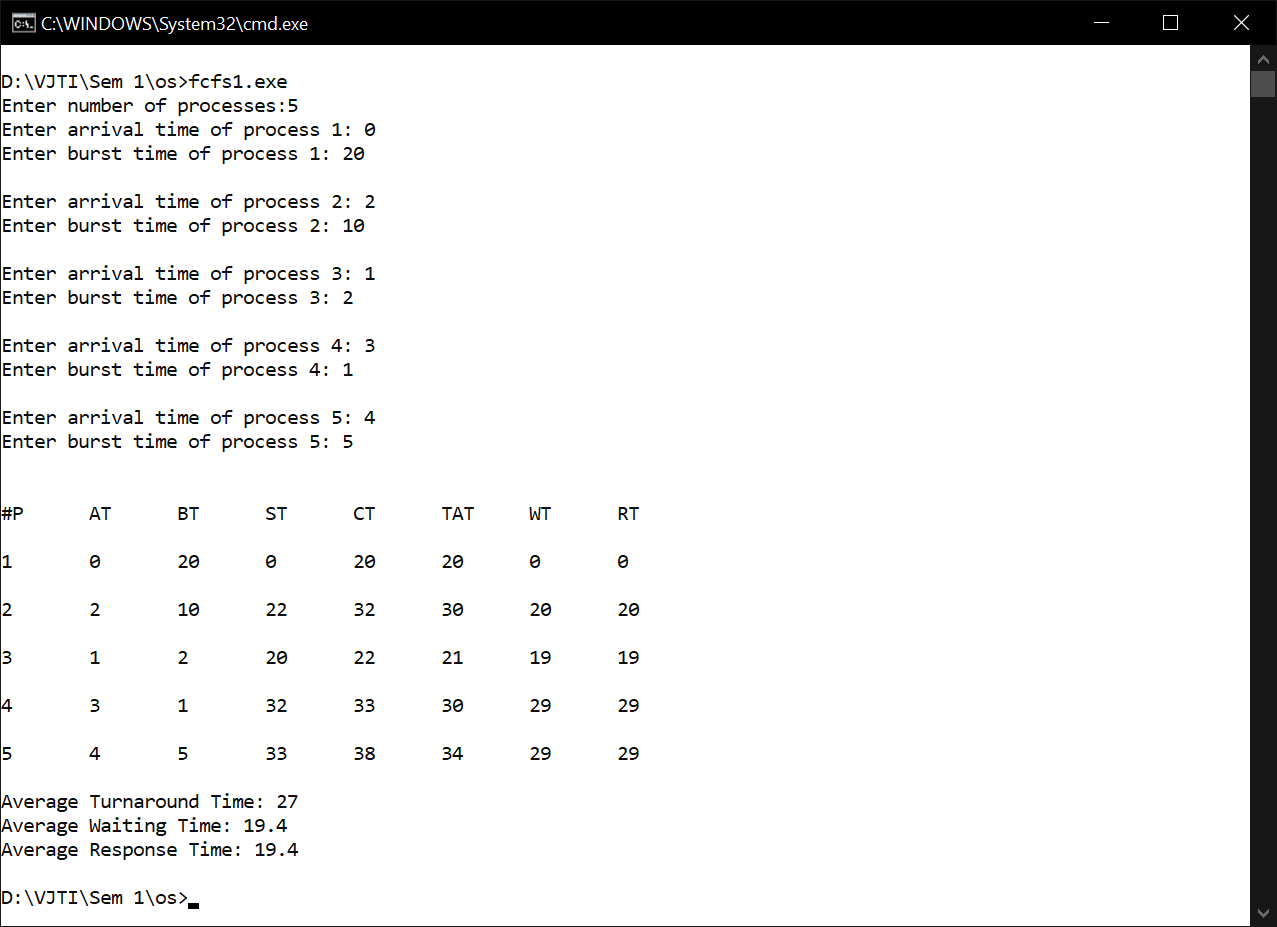
cout << "Average Waiting Time: " << (float)total\_waiting\_time / n << endl;

cout << "Average Response Time: " << (float)total\_response\_time / n << endl;

return 0;

}

**Output:**



1. **i) SJF (Shortest Job First) Non Pre-emptive:-**

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

class Process{

public:

int p\_id;

int arrival\_time;

int burst\_time;

int start\_time; // first time cpu alloted time

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

int main(){

int n;

cout << "Enter number of processes:";

cin >> n;

Process p[n];

bool is\_completed[n] = {false};

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

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].p\_id = i + 1;

cout << endl;

}

int current\_time = 0;

int completed = 0;

while (completed != n){

int curr\_process = -1;

int minimum\_burst\_time = INT\_MAX;

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

if (p[i].arrival\_time <= current\_time && is\_completed[i] == false){

if (p[i].burst\_time < minimum\_burst\_time){

minimum\_burst\_time = p[i].burst\_time;

curr\_process = i;

}

if (p[i].burst\_time == minimum\_burst\_time){

if (p[i].arrival\_time < p[curr\_process].arrival\_time){

minimum\_burst\_time = p[i].burst\_time;

curr\_process = i;

}

}

}

}

if (curr\_process != -1){

p[curr\_process].start\_time = current\_time;

p[curr\_process].completion\_time = p[curr\_process].start\_time + p[curr\_process].burst\_time;

p[curr\_process].turnaround\_time = p[curr\_process].completion\_time - p[curr\_process].arrival\_time;

p[curr\_process].waiting\_time = p[curr\_process].turnaround\_time - p[curr\_process].burst\_time;

p[curr\_process].response\_time = p[curr\_process].start\_time - p[curr\_process].arrival\_time;

total\_turnaround\_time = total\_turnaround\_time + p[curr\_process].turnaround\_time;

total\_waiting\_time = total\_waiting\_time + p[curr\_process].waiting\_time;

total\_response\_time = total\_response\_time + p[curr\_process].response\_time;

completed++;

is\_completed[curr\_process] = true;

current\_time = p[curr\_process].completion\_time;

}else{

current\_time++;

}

}

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].p\_id << "\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].response\_time << "\t"<< "\n" << endl;

}

cout << "Average Turnaround Time: " << (float)total\_turnaround\_time / n << endl;

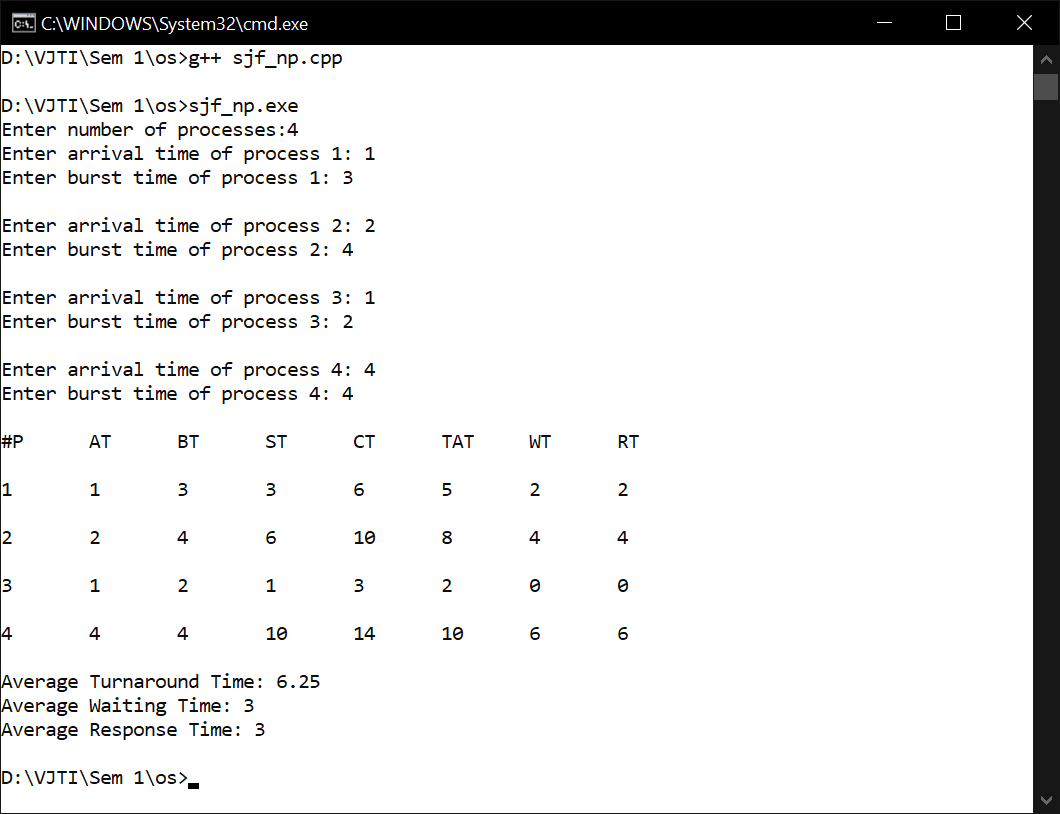
cout << "Average Waiting Time: " << (float)total\_waiting\_time / n << endl;

cout << "Average Response Time: " << (float)total\_response\_time / n << endl;

return 0;

}

**Output:**



1. **ii) SJF (Shortest Job First) Pre-emptive or SRTF:-**

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

class Process{

public:

int p\_id;

int arrival\_time;

int burst\_time;

int remaining\_burst\_time;

int start\_time; // first time cpu alloted time

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

int main(){

int n;

cout << "Enter number of processes:";

cin >> n;

Process p[n];

bool is\_completed[n] = {false};

bool is\_in\_queue[n] = {false};

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

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].remaining\_burst\_time = p[i].burst\_time;

p[i].p\_id = i + 1;

cout << endl;

}

int completed = 0;

int current\_time = 0;

while (completed != n) {

int curr\_process = -1;

int minimum\_burst\_time = INT\_MAX;

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

if (p[i].arrival\_time <= current\_time && is\_completed[i] = false){

if (p[i].remaining\_burst\_time < minimum\_burst\_time){

minimum\_burst\_time = p[i].remaining\_burst\_time;

curr\_process = i;

}

if (p[i].remaining\_burst\_time == minimum\_burst\_time){

if (p[i].arrival\_time < p[curr\_process].arrival\_time){

minimum\_burst\_time = p[i].remaining\_burst\_time;

curr\_process = i;

}

}

}

}

if (curr\_process != -1){

if(p[curr\_process].remaining\_burst\_time == p[curr\_process].burst\_time){

p[curr\_process].start\_time = current\_time;

}

p[curr\_process].remaining\_burst\_time--;

current\_time++;

if(p[curr\_process].remaining\_burst\_time == 0){

p[curr\_process].completion\_time = current\_time;

p[curr\_process].turnaround\_time = p[curr\_process].completion\_time - p[curr\_process].arrival\_time;

p[curr\_process].waiting\_time = p[curr\_process].turnaround\_time - p[curr\_process].burst\_time;

p[curr\_process].response\_time = p[curr\_process].start\_time - p[curr\_process].arrival\_time;

total\_turnaround\_time = total\_turnaround\_time + p[curr\_process].turnaround\_time;

total\_waiting\_time = total\_waiting\_time + p[curr\_process].waiting\_time;

total\_response\_time=total\_response\_time + p[curr\_process].response\_time;

is\_completed[curr\_process]=true;

completed++;

}

}

else{

current\_time++;

}

}

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].p\_id << "\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].response\_time << "\t" << "\n"<< endl;

}

cout << "Average Turnaround Time: " << (float)total\_turnaround\_time / n << endl;

cout << "Average Waiting Time: " << (float)total\_waiting\_time / n << endl;

cout << "Average Response Time: " << (float)total\_response\_time / n << endl;

return 0;

}

**Output:**

