Lab-1

1. C++ Program for FCFS Scheduling

```
#include <iostream>
#include <algorithm>
#include <iomanip>
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;
};

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;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;

    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;

        total_turnaround_time += p[i].turnaround_time;
        total_waiting_time += p[i].waiting_time;
```

```
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;

    sort(p,p+n,compareID);

    cout<<endl;
    cout<<"P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\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].completion_tim
e<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
}
```

Output:

```
Enter the number of processes: 5
Enter arrival time of process 1: 2
Enter burst time of process 1: 2

Enter arrival time of process 2: 0
Enter burst time of process 2: 1

Enter arrival time of process 3: 2
Enter burst time of process 3: 3

Enter arrival time of process 4: 3
Enter burst time of process 4: 5

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


P       AT      BT      CT      TAT     WT

1       2       2       4       2       0

2       0       1       1       1       0

3       2       3       7       5       2

4       3       5       12      9       4

5       4       4       16      12      8

Average Turnaround Time = 5.80
Average Waiting Time = 2.80
```

2. C++ program for non-pre-emptive shortest job first scheduling

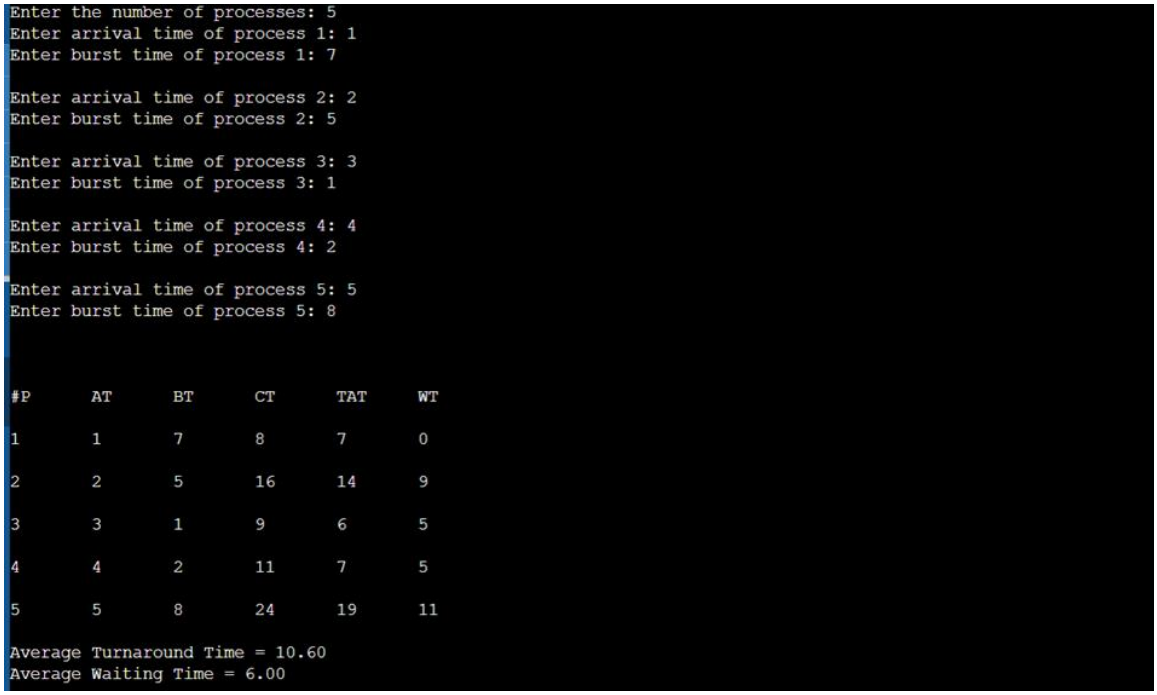
```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

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

int main() {

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

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



        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;

    cout<<endl<<endl;

    cout<<"P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\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].completion_tim
e<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
}
```

Output:



Submitted by: Gelle Hruthesh Reddy, 20BCB7031