Operating System Lab

CEN-404

Lab File

Name : Vicky Gupta

Roll No : 20BCS070

Branch : Computer Engineering

Semester : IV

Subject Code : Cen 404

INDEX

9	Write a program to implement the First fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	*24-Mar-2022*
10	Write a program to implement the Next fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	*24-Mar-2022*
11	Write a program to implement the Best fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	*31-Mar-2022*
12	Write a program to implement the worst fit memory management algorithm. The program should take input total no. of the memory block, their sizes, process name, and process size. The output of the program should give the details about memory allocated to process with fragmentation detail.	*7-Apr-2022*
13	Write a program to implement the First In First Out(FIFO) page replacement algorithm. Program should take input reference string and total no. of pages that can accommodate in memory. Output contains detail about each page fault details and calculate average page fault.	*28-Apr-2022*
14	Write a program to implement the Least Recently Used (LRU) page replacement algorithm. Program should take input reference string and total no. of pages that can accommodate in memory. Output contains detail about each page fault details and calculate average page fault.	*28-Apr-2022*
15	Write a program to implement FCFS and SSTF elevator disk scheduling algorithm. Program should give detail about each disk movement from starting head position (input from user) and calculate average head movement.	*5-May-2022*

Operating System Lab

CEN-493

Program - 1

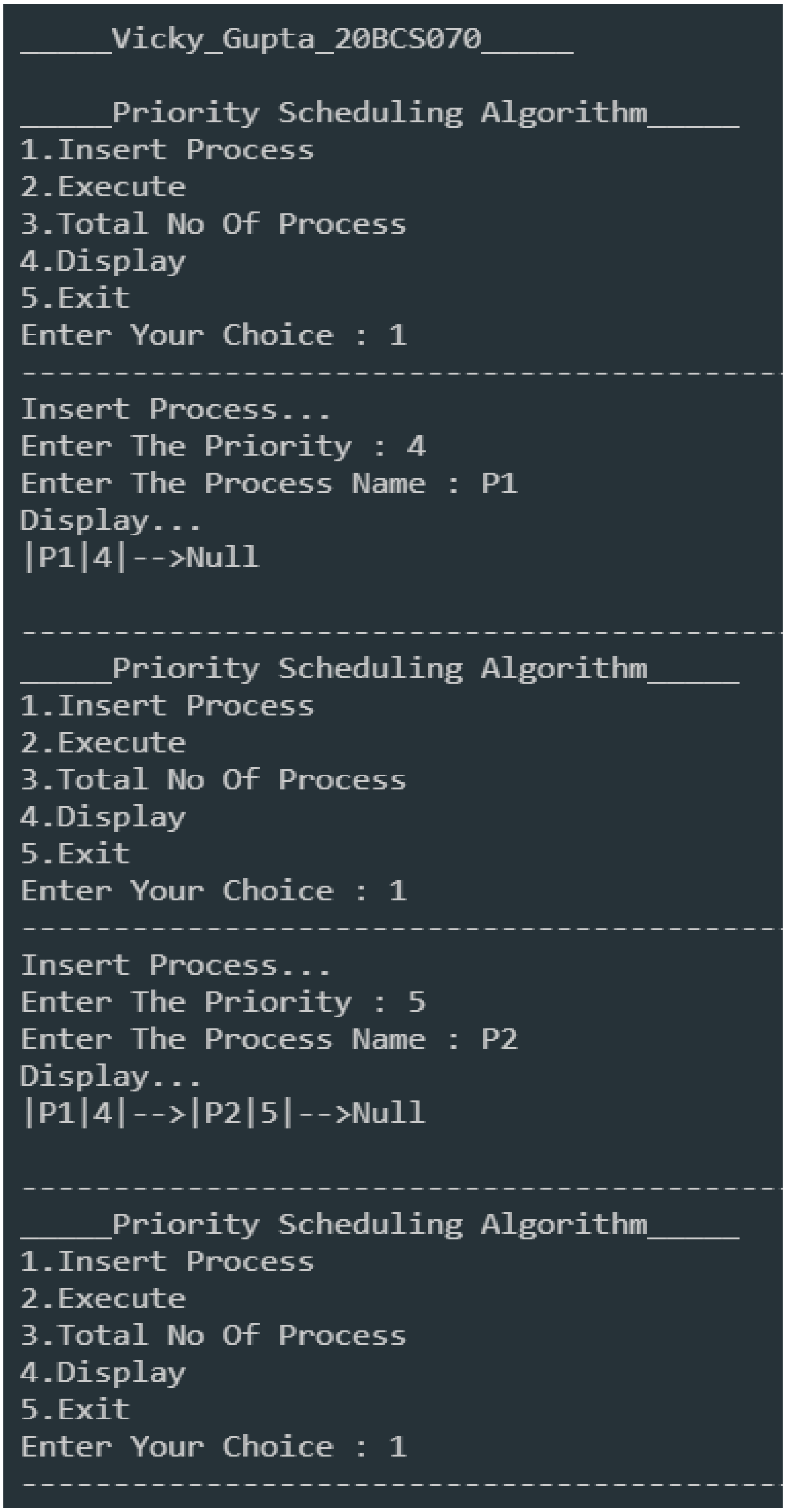
Code :-

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

struct Priority_Queue
{
    char process_name[4];
    int priority;
    Priority_Queue *next;
};

void isEmpty(int size)
{
    cout << "isEmpty...\n";
    if (size == 0)
        cout << "Empty" << endl;
    else
        cout << "Not Empty" << endl;
}

void Display(Priority_Queue *head, int size)
{
    cout << "Display...\n";
    if (size == 0)
    {
        cout << "Queue Is Empty" << endl;
        return;
    }
    while (head != nullptr)
    {
        cout << "|" << head->process_name << "|" << head-
>priority << "|"
             << "-->";
        head = head->next;
    }
    cout << "Null\n";
    cout << endl;
}

void Process_Initialized(Priority_Queue *&new_process)
{
    cout << "Enter The Priority : ";
    cin >> new_process->priority;
    fflush(stdin);
    cout << "Enter The Process Name : ";
    gets(new_process->process_name);
    new_process->next = nullptr;
}

void Insert_Process(Priority_Queue *&head, Priority_Queue
*&tail, int &size)
{
    cout << "Insert Process...\n";
    Priority_Queue *new_process = (Priority_Queue *)malloc(1
* sizeof(Priority_Queue));
```

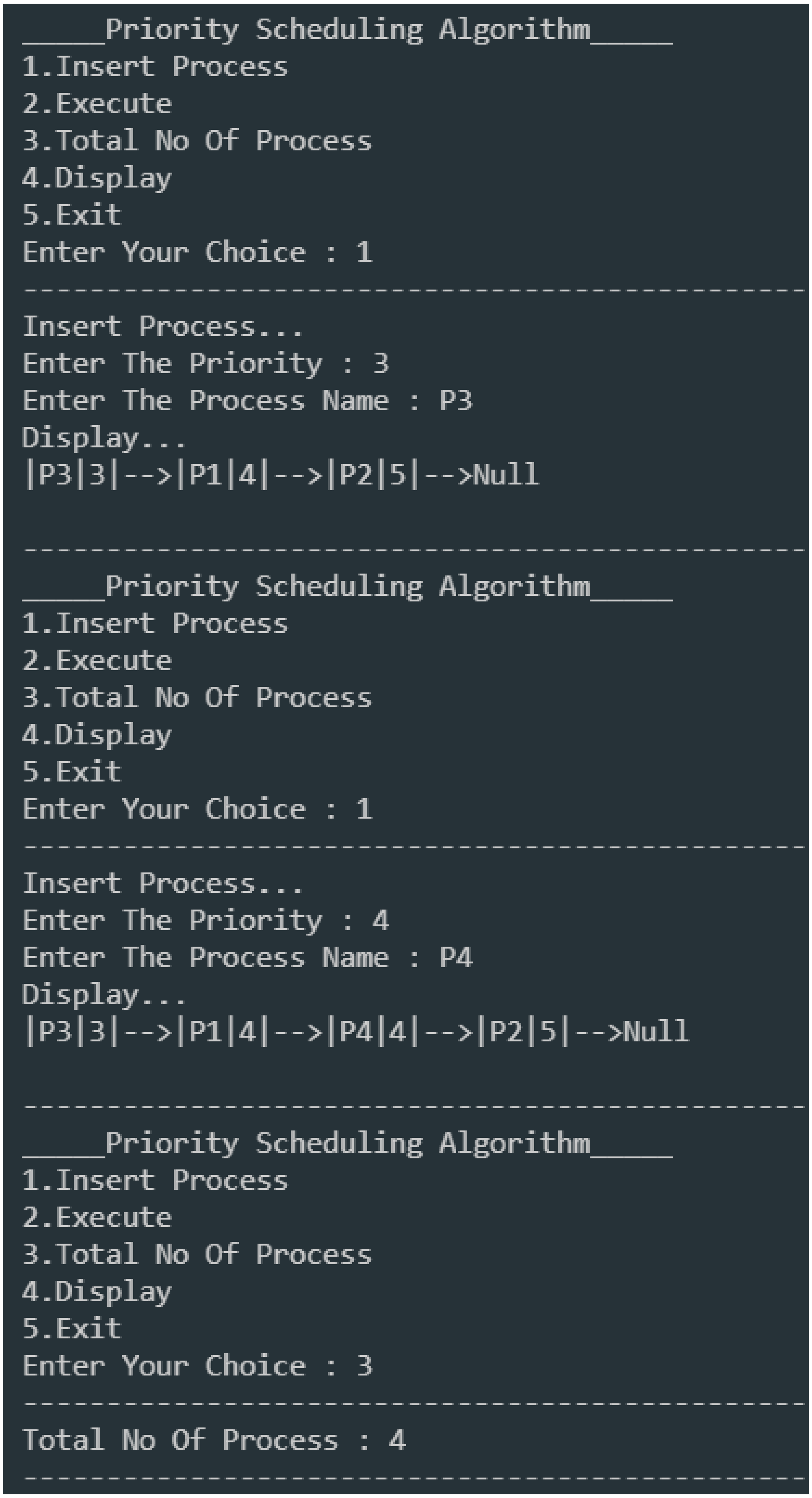
```
    if (new_process == nullptr)
    {
        cout << "Memory Not Assigned" << endl;
        return;
    }
    size++;

    Process_Initialized(new_process);

    Priority_Queue *temp = head;
    if (head == nullptr)
    {
        head = new_process;
        tail = new_process;
    }
    else
    {
        if (temp->priority > new_process->priority)
        {
            new_process->next = head;
            head = new_process;
        }
        else if (tail->priority <= new_process->priority)
        {
            tail->next = new_process;
            tail = tail->next;
        }
        else
        {
            while (temp && temp->next)
            {
                if (temp->next->priority > new_process-
>priority)
                {
                    new_process->next = temp->next;
                    temp->next = new_process;
                    break;
                }
                temp = temp->next;
            }
        }
    }
    Display(head, size);
}

void Execute_Process(Priority_Queue *&head, int &size)
{
    cout << "Execute_Process...\n";
    if (size == 0)
    {
        cout << "Queue Underflow" << endl;
        return;
    }
    cout << "|" << head->process_name << "|" << head-
>priority << "|"
         << "\n";
    size--;
    Priority_Queue *todelete = head;
    head = head->next;
    delete todelete;
    Display(head, size);
}

void Total_Process(int size)
{
    cout << "Total No Of Process : " << size << endl;
}

void Bars()
{
    cout << "-----------------------------------------------
----------------\n";
}
bool Options(Priority_Queue *&head, Priority_Queue *&tail,
int &size)
{
    int opt;
    cin >> opt;
```

```cpp
    Bars();
    switch (opt)
    {
    case 1:
        Insert_Process(head, tail, size);
        break;
    case 2:
        Execute_Process(head, size);
        break;
    case 3:
        Total_Process(size);
        break;
    case 4:
        Display(head, size);
        break;
    case 5:
        cout << "Exit...\n";
        return 0;
    default:
        cout << "Invalid Input!\nTry Again!\n";
    }
    Bars();
    return 1;
}

void Menu()
{
    cout << "_____Priority Scheduling Algorithm_____ \n";
    cout << "1.Insert Process \n";
    cout << "2.Execute \n";
    cout << "3.Total No Of Process \n";
    cout << "4.Display \n";
    cout << "5.Exit \n";
    cout << "Enter Your Choice : ";
}

int main()
{
    system("cls");
    cout << "_____Vicky_Gupta_20BCS070_____\n\n";
    int size = 0;
    Priority_Queue *head = nullptr, *tail = nullptr;
    while (true)
    {
        Menu();
        if (!Options(head, tail, size))
            break;
    }
    cout << "Exiting...\n";
    Bars();
    return 0;
}
```

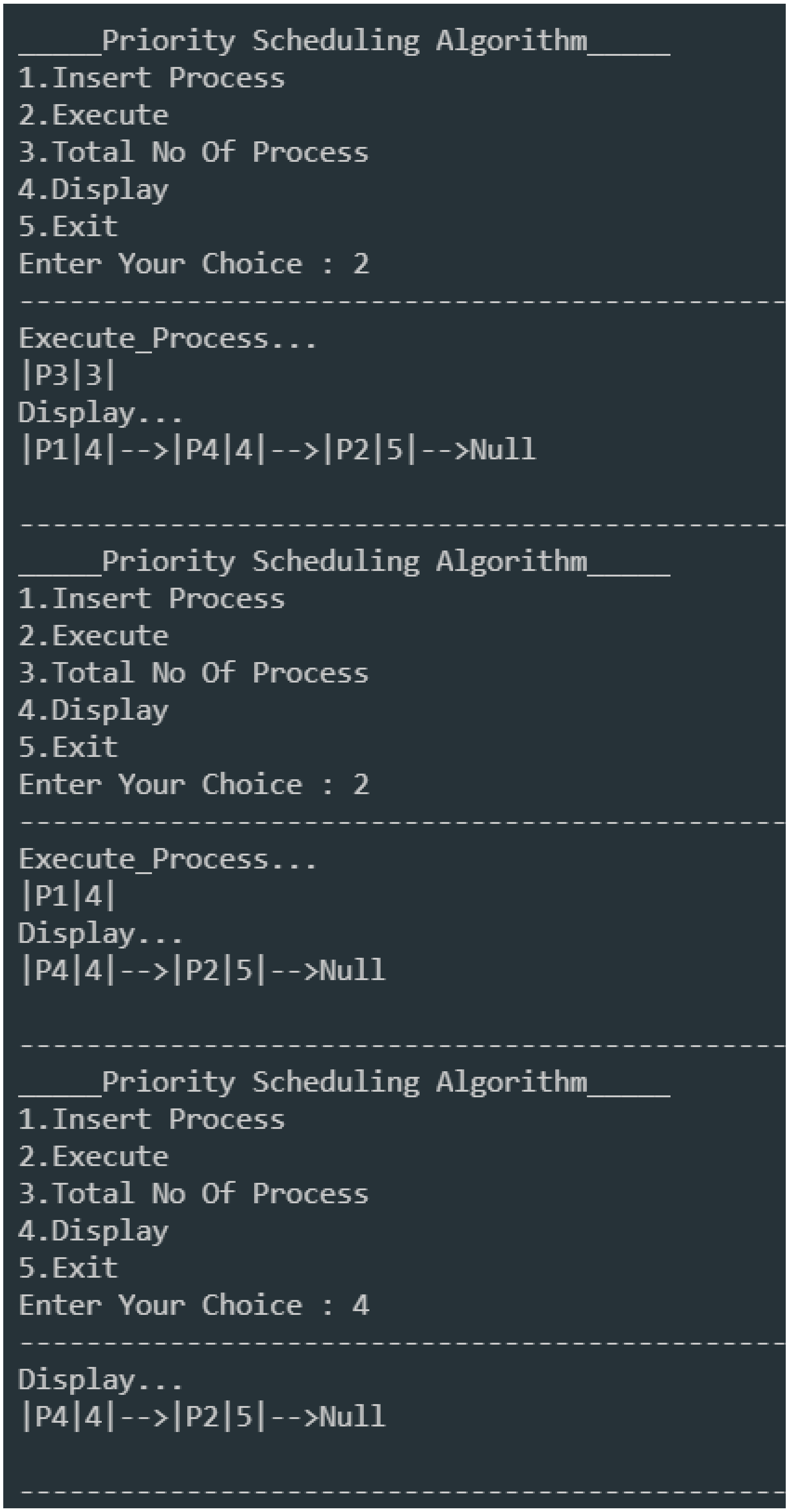
Output :-



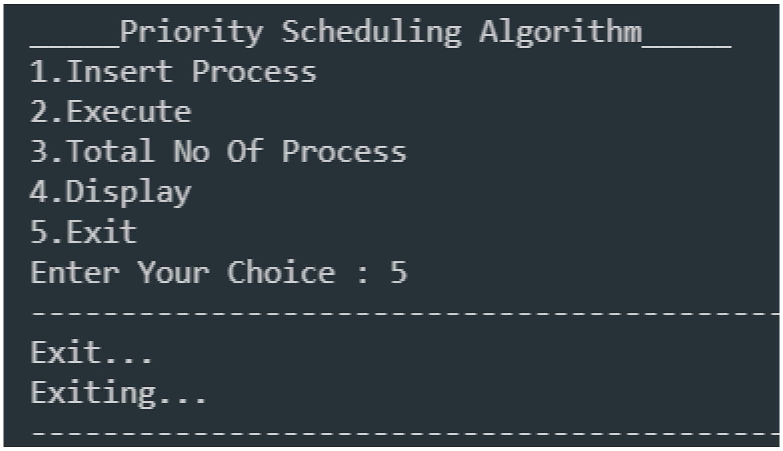
```
_____Vicky_Gupta_20BCS070_____

_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 1
-----------------------------------------
Insert Process...
Enter The Priority : 4
Enter The Process Name : P1
Display...
|P1|4|-->Null

-----------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 1
-----------------------------------------
Insert Process...
Enter The Priority : 5
Enter The Process Name : P2
Display...
|P1|4|-->|P2|5|-->Null

-----------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 1
-----------------------------------------
```



```
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 1
-----------------------------------------------
Insert Process...
Enter The Priority : 3
Enter The Process Name : P3
Display...
|P3|3|-->|P1|4|-->|P2|5|-->Null

-----------------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 1
-----------------------------------------------
Insert Process...
Enter The Priority : 4
Enter The Process Name : P4
Display...
|P3|3|-->|P1|4|-->|P4|4|-->|P2|5|-->Null

-----------------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 3
-----------------------------------------------
Total No Of Process : 4
-----------------------------------------------
```



```
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 2
----------------------------------------------
Execute_Process...
|P3|3|
Display...
|P1|4|-->|P4|4|-->|P2|5|-->Null

----------------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 2
----------------------------------------------
Execute_Process...
|P1|4|
Display...
|P4|4|-->|P2|5|-->Null

----------------------------------------------
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 4
----------------------------------------------
Display...
|P4|4|-->|P2|5|-->Null

----------------------------------------------
```



```
_____Priority Scheduling Algorithm_____
1.Insert Process
2.Execute
3.Total No Of Process
4.Display
5.Exit
Enter Your Choice : 5
--------------------------------------------
Exit...
Exiting...
--------------------------------------------
```

Operating System Lab

CEN-493

Program - 2

Code :-

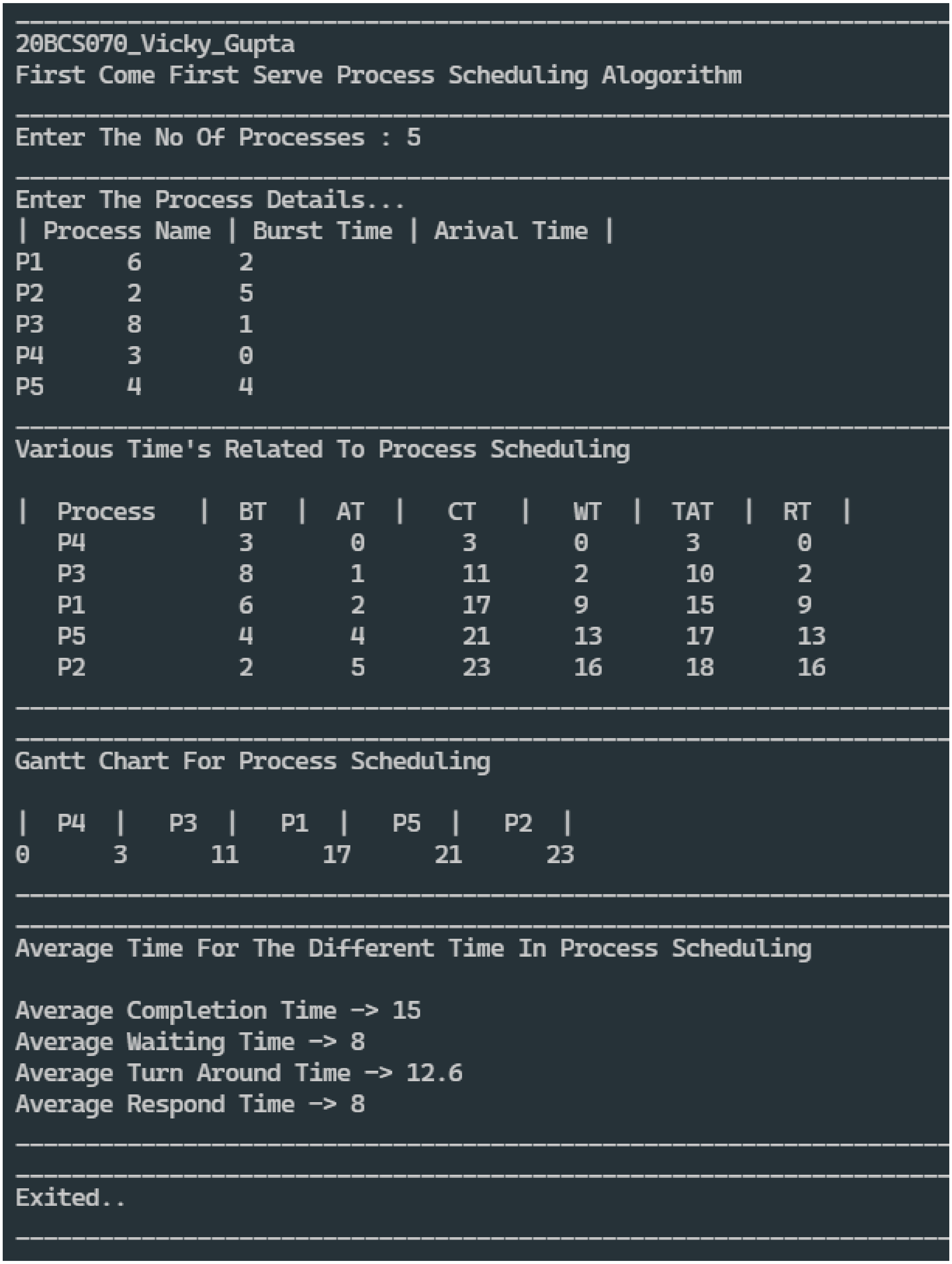
```
#include <iostream>
using namespace std;

struct Process
{
    string pname;
    int arival_time;
    int burst_time;
    int waiting_time;
    int completion_time;
    int response_time;
    int turnaound_time;
};

void Print_Bars()
{
    for (int i = 0; i < 100; i++)
        cout << "_";
    cout << "\n";
}

void Insertion_Sort(Process Process_Array[], int
total_process)
{
    for (int i = 1; i < total_process; i++)
    {
        Process curent = Process_Array[i];
        int j = i - 1;
        while (Process_Array[j].arival_time >
curent.arival_time && j >= 0)
        {
            Process_Array[j + 1] = Process_Array[j];
            j--;
        }
        Process_Array[j + 1] = curent;
    }
}

void Average_Time(Process Process_Array[], int
total_process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < total_process; i++)
    {
        Av_CT += Process_Array[i].completion_time;
        Av_RT += Process_Array[i].response_time;
        Av_TAT += Process_Array[i].turnaound_time;
        Av_WT += Process_Array[i].waiting_time;
    }
    Av_WT /= total_process;
    Av_TAT /= total_process;
    Av_RT /= total_process;
    Av_CT /= total_process;
    cout << "Average Time For The Different Time In
Process Scheduling\n\n";
```

```
    cout << "Average Completion Time -> " << Av_CT <<
"\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT <<
"\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(Process Process_Array[], int
total_process)
{

    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (Process_Array[0].arival_time != 0)
    {
        cout << "|     |  ";
    }
    else
    {
        cout << "|  ";
    }

    for (int i = 0; i < total_process; i++)
    {
        if (i != 0 && Process_Array[i -
1].completion_time < Process_Array[i].arival_time)
        {
            cout << "    |   ";
        }
        cout << Process_Array[i].pname << "  |   ";
    }
    cout << "\n";

    if (Process_Array[0].arival_time != 0)
    {
        cout << " 0     ";
        cout << Process_Array[0].arival_time << "     ";
    }
    else
    {
        cout << Process_Array[0].arival_time << "      ";
    }

    for (int i = 0; i < total_process; i++)
    {
        if (i != 0 && Process_Array[i -
1].completion_time < Process_Array[i].arival_time)
        {
            cout << Process_Array[i].arival_time <<
"      ";
        }
        cout << Process_Array[i].completion_time <<
"      ";
    }
    cout << "\n";
}

void Chart(Process Process_Array[], int total_process)
{
    cout << "Various Time's Related To Process
Scheduling\n\n";
    cout <<
"|  Process   |  BT  |  AT  |   CT   |   WT  |  TAT  |  R
T  |\n";
    for (int i = 0; i < total_process; i++)
    {
        cout << "   " << Process_Array[i].pname << "\t\t"
<< Process_Array[i].burst_time
             << "\t" << Process_Array[i].arival_time <<
"\t" << Process_Array[i].completion_time
             << "\t" << Process_Array[i].waiting_time <<
"\t" << Process_Array[i].turnaound_time
             << "\t" << Process_Array[i].response_time <<
"\n";
    }
}
```

```
void FCFS(Process Process_Array[], int total_process)
{
    Insertion_Sort(Process_Array, total_process); //
Acording To A.T

    int timer = 0;
    for (int i = 0; i < total_process; i++)
    {
        if (timer < Process_Array[i].arival_time)
        {
            timer += (Process_Array[i].arival_time -
timer);
        }
        timer += Process_Array[i].burst_time;

        Process_Array[i].completion_time = timer;

        Process_Array[i].turnaound_time =
            Process_Array[i].completion_time -
            Process_Array[i].arival_time;

        Process_Array[i].waiting_time =
            Process_Array[i].turnaound_time -
            Process_Array[i].burst_time;

        Process_Array[i].response_time =
Process_Array[i].waiting_time;
    }
    Print_Bars();
    Chart(Process_Array, total_process);
    Print_Bars();
    Print_Bars();
    GanttChart(Process_Array, total_process);
    Print_Bars();
    Print_Bars();
    Average_Time(Process_Array, total_process);
    Print_Bars();
}
```

```
int main()
{
    system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "First Come First Serve Process Scheduling
Alogorithm\n";
    Print_Bars();
    int total_process;
    cout << "Enter The No Of Processes : ";
    cin >> total_process;
    fflush(stdin);
    Process Process_Array[total_process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Burst Time | Arival Time |
\n";

    for (int i = 0; i < total_process; i++)
    {
        cin >> Process_Array[i].pname;
        cin >> Process_Array[i].burst_time;
        cin >> Process_Array[i].arival_time;
    }

    FCFS(Process_Array, total_process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
-------------------------------------------------------------------
20BCS070_Vicky_Gupta
First Come First Serve Process Scheduling Alogorithm
-------------------------------------------------------------------
Enter The No Of Processes : 5
-------------------------------------------------------------------
Enter The Process Details...
| Process Name | Burst Time | Arival Time |
P1      6       2
P2      2       5
P3      8       1
P4      3       0
P5      4       4
-------------------------------------------------------------------
Various Time's Related To Process Scheduling

|  Process   |  BT  |  AT  |   CT   |   WT  |  TAT  |  RT  |
   P4           3       0       3       0       3       0
   P3           8       1       11      2       10      2
   P1           6       2       17      9       15      9
   P5           4       4       21      13      17      13
   P2           2       5       23      16      18      16
-------------------------------------------------------------------
-------------------------------------------------------------------
Gantt Chart For Process Scheduling

|  P4  |   P3  |   P1  |   P5  |   P2  |
0      3      11      17      21      23
-------------------------------------------------------------------
-------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 15
Average Waiting Time -> 8
Average Turn Around Time -> 12.6
Average Respond Time -> 8
-------------------------------------------------------------------
-------------------------------------------------------------------
Exited..
-------------------------------------------------------------------
```

Operating System Lab

CEN-493

Program - 3

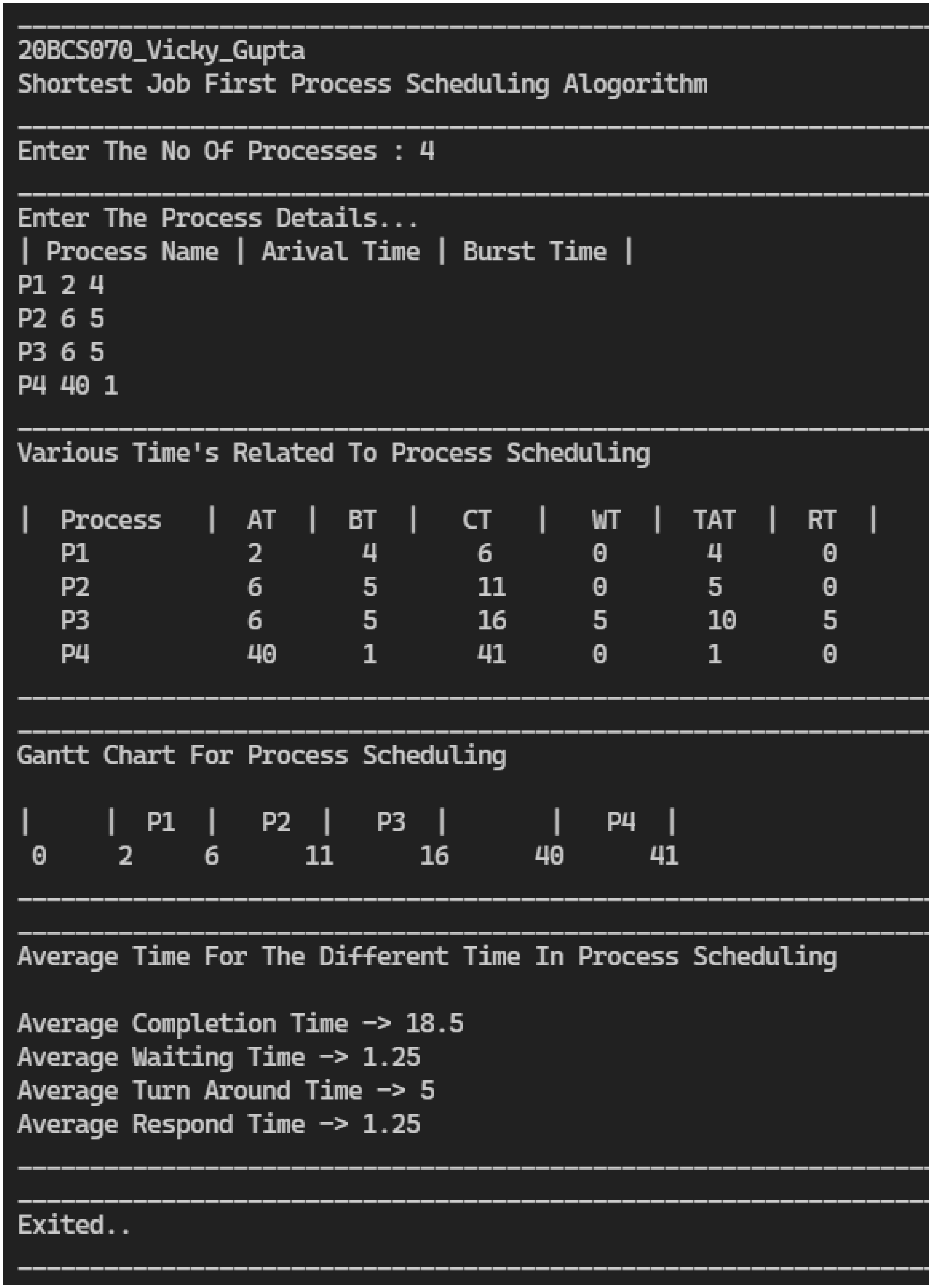
Code :-

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

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int WT;
    int CT;
    int RT;
    int TAT;
};

bool mycomp(Process P1, Process P2)
```

```
{

    if (P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }
    else if (P1.BT != P2.BT)
    {
        return P1.BT < P2.BT;
    }
    else
    {
        int num1 = stoi(P1.P_Name.substr(1));
        int num2 = stoi(P2.P_Name.substr(1));
        return num1 < num2;
    }
}

void Print_Bars()
{
    for (int i = 0; i < 100; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In
Process Scheduling\n\n";

    cout << "Average Completion Time -> " << Av_CT <<
"\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT <<
"\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(Process P_Array[], int T_Process)
{

    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (P_Array[0].AT != 0)
    {
        cout << "|     |  ";
    }
    else
    {
        cout << "|  ";
    }

    for (int i = 0; i < T_Process; i++)
    {
        if (i != 0 && P_Array[i - 1].CT < P_Array[i].AT)
        {
            cout << "    |   ";
        }
        cout << P_Array[i].P_Name << "  |   ";
    }
    cout << "\n";

    if (P_Array[0].AT != 0)
    {
        cout << " 0     ";
        cout << P_Array[0].AT << "     ";
    }
    else
    {
        cout << P_Array[0].AT << "      ";
    }

    for (int i = 0; i < T_Process; i++)
    {
        if (i != 0 && P_Array[i - 1].CT < P_Array[i].AT)
        {
            cout << P_Array[i].AT << "      ";
        }
        cout << P_Array[i].CT << "      ";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process
Scheduling\n\n";
    cout <<
"|  Process   |  AT  |  BT  |   CT   |   WT   |  TAT  |  R
T  |\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "   " << P_Array[i].P_Name << "\t\t" <<
P_Array[i].AT
             << "\t" << P_Array[i].BT << "\t" <<
P_Array[i].CT
             << "\t" << P_Array[i].WT << "\t" <<
P_Array[i].TAT
             << "\t" << P_Array[i].RT << "\n";
    }
}

void New_Process_Array(Process P_Array[], Process
N_P_Array[], int T_Process)
{
    sort(P_Array, P_Array + T_Process, mycomp);
    bool isProcessed[T_Process] = {0};
    int Timer = P_Array[0].AT;
    for (int i = 0; i < T_Process; i++)
    {
        int p_no = -1;
        for (int j = 0; j < T_Process; j++)
        {
            if (Timer >= P_Array[j].AT && isProcessed[j]
== 0)
            {
                if (p_no == -1)
                {
                    p_no = j;
                }
                if (p_no != -1 && P_Array[p_no].BT >
P_Array[j].BT)
                {
                    p_no = j;
                }
            }
        }
        if (p_no == -1) // when the process has gaps
        {
            for (int j = 0; j < T_Process; j++)
            {
                if (isProcessed[j] == 0)
                {
                    p_no = j;
                    break;
                }
            }
        }
        isProcessed[p_no] = 1;
        N_P_Array[i] = P_Array[p_no];
        if (Timer < P_Array[p_no].AT)
        {
            Timer += (P_Array[p_no].AT - Timer);
        }
        Timer += P_Array[p_no].BT;
    }
}

void SJF(Process P_Array[], int T_Process)
{
    Process N_P_Array[T_Process];
    New_Process_Array(P_Array, N_P_Array, T_Process);

    int Timer = 0;
    for (int i = 0; i < T_Process; i++)
    {

        if (Timer < N_P_Array[i].AT)
        {
            Timer += (N_P_Array[i].AT - Timer);
        }
        Timer += N_P_Array[i].BT;

        N_P_Array[i].CT = Timer;

        N_P_Array[i].TAT = N_P_Array[i].CT -
N_P_Array[i].AT;

        N_P_Array[i].WT = N_P_Array[i].TAT -
N_P_Array[i].BT;

        N_P_Array[i].RT = N_P_Array[i].WT;
    }
    Print_Bars();
    Chart(N_P_Array, T_Process);
    Print_Bars();
    Print_Bars();
    GanttChart(N_P_Array, T_Process);
    Print_Bars();
    Print_Bars();
    Average_Time(N_P_Array, T_Process);
    Print_Bars();
}
```

```
int main()
{
    // system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Shortest Job First Process Scheduling
Alogorithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time |
\n";

    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
    }

    SJF(P_Array, T_Process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
---------------------------------------------------------------
20BCS070_Vicky_Gupta
Shortest Job First Process Scheduling Alogorithm

---------------------------------------------------------------
Enter The No Of Processes : 4

---------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time |
P1 2 4
P2 6 5
P3 6 5
P4 40 1

---------------------------------------------------------------
Various Time's Related To Process Scheduling

|  Process   |  AT  |  BT  |   CT   |   WT  |  TAT  |  RT  |
   P1           2       4       6       0       4       0
   P2           6       5       11      0       5       0
   P3           6       5       16      5       10      5
   P4           40      1       41      0       1       0

---------------------------------------------------------------
---------------------------------------------------------------
Gantt Chart For Process Scheduling

|     |  P1  |   P2  |   P3  |       |   P4  |
 0     2     6      11      16      40      41

---------------------------------------------------------------
---------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 18.5
Average Waiting Time -> 1.25
Average Turn Around Time -> 5
Average Respond Time -> 1.25

---------------------------------------------------------------
---------------------------------------------------------------
Exited..

---------------------------------------------------------------
```

Operating System Lab

CEN-493

Program - 4

Code :-

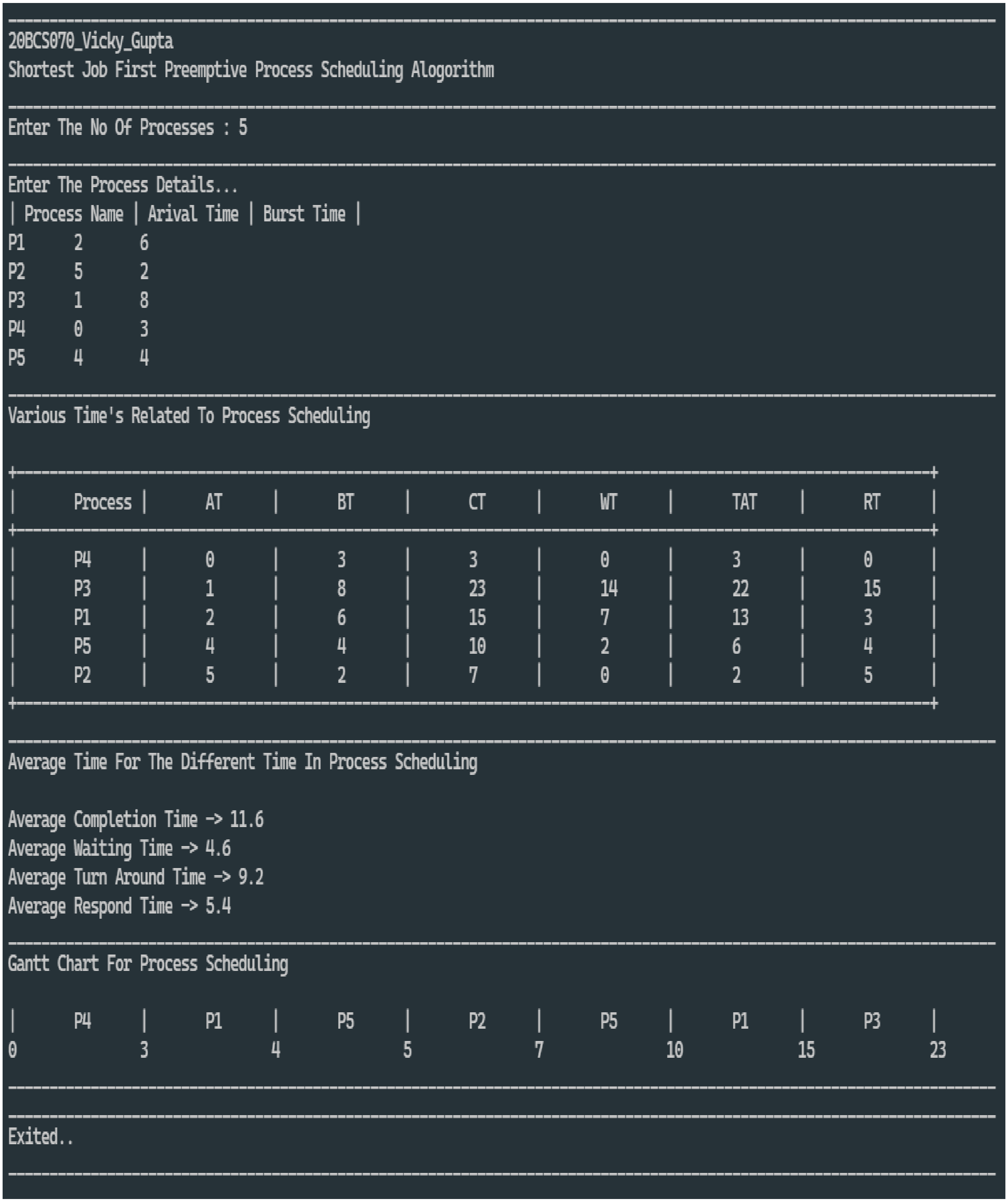
```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <unordered_map>
#include <stack>
using namespace std;

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int WT;
    int CT;
    int RT;
    int TAT;
};
```

```
struct myCompBT
{
    bool operator()(Process &p1, Process const &p2)
    {
        return p1.BT > p2.BT;
    }
};

bool mycomp(Process P1, Process P2)
{

    if (P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }
    else if (P1.BT != P2.BT)
    {
        return P1.BT < P2.BT;
    }
    else
    {
        int num1 = stoi(P1.P_Name.substr(1));
        int num2 = stoi(P2.P_Name.substr(1));
        return num1 < num2;
    }
}

bool mycompInterval(pair<string, pair<int, int>> p1, pair<string,
pair<int, int>> p2)
{
    return p1.second.first < p2.second.first;
}

vector<pair<string, pair<int, int>>>
Merge_Interval_Helper(vector<pair<int, int>> Interval, string
P_Name)
{

    stack<pair<int, int>> helper;
    int Interval_Length = Interval.size();
    helper.push(Interval[0]);
    for (int i = 1; i < Interval_Length; i++)
    {
        if (Interval[i].first <= helper.top().second)
        {
            helper.top().second = Interval[i].second;
        }
        else
        {
            helper.push(Interval[i]);
        }
    }
    vector<pair<string, pair<int, int>>> result;
    while (!helper.empty())
    {
        result.push_back({P_Name, {helper.top().first,
helper.top().second}});
        helper.pop();
    }
    return result;
}

vector<pair<string, pair<int, int>>>
Merge_Interval(unordered_map<string, vector<pair<int, int>>>
&executionTime)
{
    vector<pair<string, pair<int, int>>> Intervals;
    for (auto &x : executionTime)
    {
        vector<pair<string, pair<int, int>>> intvl =
Merge_Interval_Helper(x.second, x.first);
        for (auto &y : intvl)
        {
            Intervals.push_back(y);
        }
    }
    sort(Intervals.begin(), Intervals.end(), mycompInterval);
    return Intervals;
}

void Print_Bars()
{
    for (int i = 0; i < 120; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
```

```cpp
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In Process
Scheduling\n\n";

    cout << "Average Completion Time -> " << Av_CT << "\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT << "\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(vector<pair<string, pair<int, int>>>
&All_Interval)
{
    int size = All_Interval.size();
    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << "|\t\t|  ";
    }
    else
    {
        cout << "|\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << "\t|\t";
        }
        cout << All_Interval[i].first << "\t|\t";
    }
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << " 0\t";
        cout << All_Interval[0].second.first << "\t";
    }
    else
    {
        cout << All_Interval[0].second.first << "\t\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << All_Interval[i].second.first << "\t\t";
        }
        cout << All_Interval[i].second.second << "\t\t";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process Scheduling\n\n";
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
    cout <<
"|\tProcess\t|\tAT\t|\tBT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT      |\n";
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "|\t" << P_Array[i].P_Name << "\t|\t" <<
P_Array[i].AT
             << "\t|\t" << P_Array[i].BT << "\t|\t" <<
P_Array[i].CT
             << "\t|\t" << P_Array[i].WT << "\t|\t" <<
P_Array[i].TAT
             << "\t|\t" << P_Array[i].RT << "\t|\n";
    }
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
}
```

```
void Timing(vector<pair<string, pair<int, int>>> &All_Interval,
Process P_Array[], int T_Process)
{
    int size = All_Interval.size();
    for (int i = 0; i < T_Process; i++)
    {
        for (int j = size - 1; j >= 0; j--)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].CT = All_Interval[j].second.second;
                break;
            }
        }
        P_Array[i].TAT = P_Array[i].CT - P_Array[i].AT;
        P_Array[i].WT = P_Array[i].TAT - P_Array[i].BT;
        for (int j = 0; j < size; j++)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].RT = All_Interval[j].second.first;
                break;
            }
        }
    }
    Print_Bars();
    Chart(P_Array, T_Process);
    Print_Bars();
    Average_Time(P_Array, T_Process);
    Print_Bars();
    GanttChart(All_Interval);
    Print_Bars();
}

void SJF_Preemptive(Process P_Array[], int T_Process)
{
    sort(P_Array, P_Array + T_Process, mycomp);
    priority_queue<Process, vector<Process>, myCompBT> pque;
    unordered_map<string, vector<pair<int, int>>> executionTime;
    int processItertor = 0;
    int timer = P_Array[processItertor].AT;
    pque.push(P_Array[processItertor]);
    if (timer != 0)
    {
        executionTime[P_Array[processItertor].P_Name].push_back({0
, timer});
    }
    processItertor++;
    while (!pque.empty() || processItertor < T_Process)
    {
        timer++;
        if (!pque.empty())
        {
            Process process = pque.top();
            pque.pop();
            process.BT--;
            executionTime[process.P_Name].push_back({timer - 1,
timer});
            if (process.BT != 0)
                pque.push(process);
        }
        while (processItertor < T_Process && timer >=
P_Array[processItertor].AT)
        {
            pque.push(P_Array[processItertor++]);
        }
    }

    vector<pair<string, pair<int, int>>> All_Interval =
Merge_Interval(executionTime);
    Timing(All_Interval, P_Array, T_Process);
}

int main()
{
    system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Shortest Job First Preemptive Process Scheduling
Alogorithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time | \n";
    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
    }

    SJF_Preemptive(P_Array, T_Process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
------------------------------------------------------------------------------------------------------------------------
20BCS070_Vicky_Gupta
Shortest Job First Preemptive Process Scheduling Alogorithm

------------------------------------------------------------------------------------------------------------------------
Enter The No Of Processes : 5

------------------------------------------------------------------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time |
P1      2       6
P2      5       2
P3      1       8
P4      0       3
P5      4       4

------------------------------------------------------------------------------------------------------------------------
Various Time's Related To Process Scheduling


+---------------------------------------------------------------------------------------------------------------+
|       Process |       AT      |       BT      |       CT      |       WT      |       TAT     |       RT      |
+---------------------------------------------------------------------------------------------------------------+
|       P4      |       0       |       3       |       3       |       0       |       3       |       0       |
|       P3      |       1       |       8       |       23      |       14      |       22      |       15      |
|       P1      |       2       |       6       |       15      |       7       |       13      |       3       |
|       P5      |       4       |       4       |       10      |       2       |       6       |       4       |
|       P2      |       5       |       2       |       7       |       0       |       2       |       5       |
+---------------------------------------------------------------------------------------------------------------+

------------------------------------------------------------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 11.6
Average Waiting Time -> 4.6
Average Turn Around Time -> 9.2
Average Respond Time -> 5.4

------------------------------------------------------------------------------------------------------------------------
Gantt Chart For Process Scheduling

|       P4      |       P1      |       P5      |       P2      |       P5      |       P1      |       P3      |
0               3               4               5               7               10              15              23
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
Exited..

------------------------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

Program - 5

Code :-

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int WT;
    int CT;
    int RT;
    int TAT;
};
```

```
bool mycomp(Process P1, Process P2)
{

    if (P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }
    else
    {
        int num1 = stoi(P1.P_Name.substr(1));
        int num2 = stoi(P2.P_Name.substr(1));
        return num1 < num2;
    }
}

void Print_Bars()
{
    for (int i = 0; i < 120; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In
Process Scheduling\n\n";
```

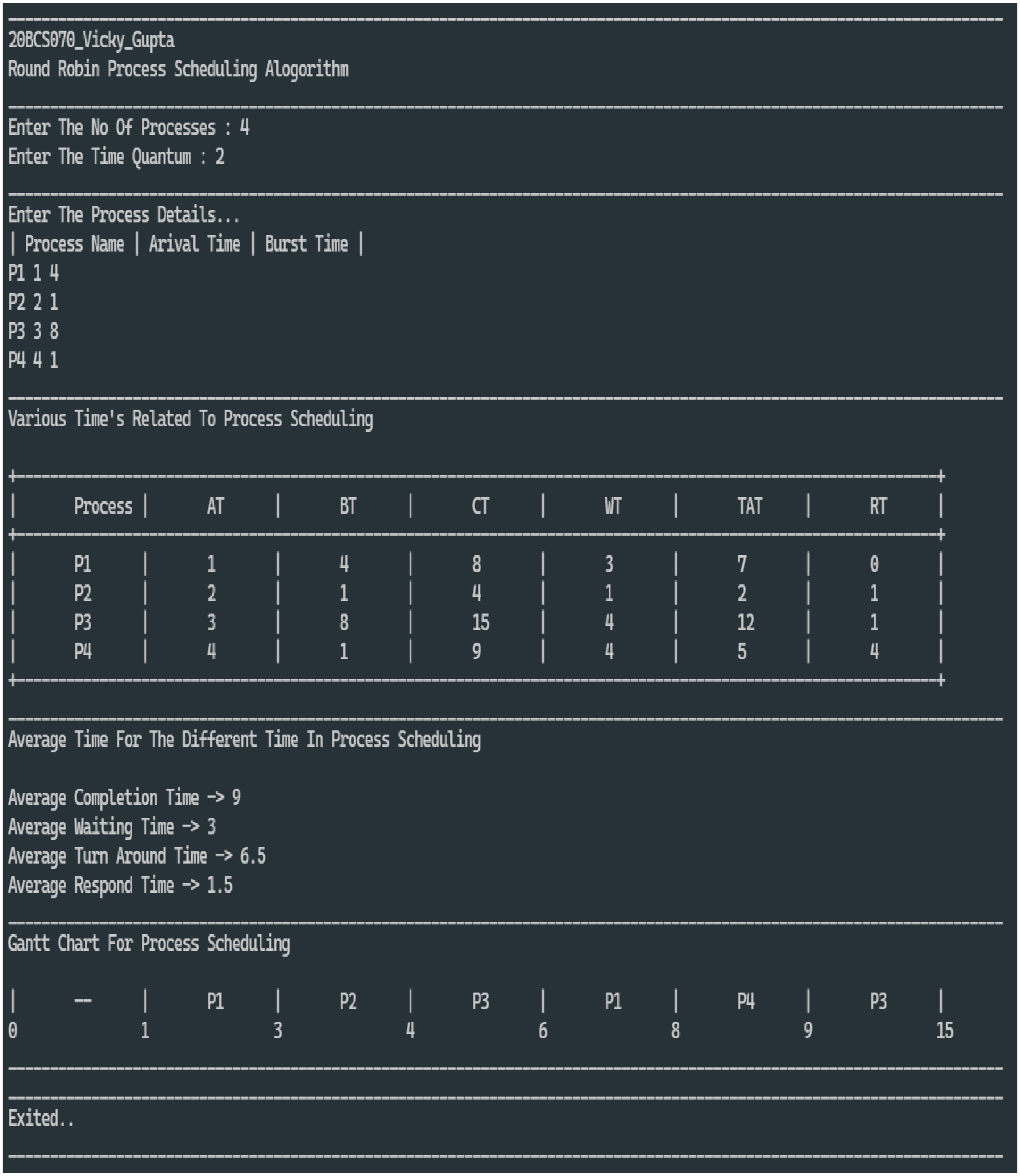
```
    cout << "Average Completion Time -> " << Av_CT << 
"\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT << 
"\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(vector<pair<string, pair<int, int>>> 
&All_Interval)
{
    int size = All_Interval.size();
    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << "|\t\t|  ";
    }
    else
    {
        cout << "|\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second < 
All_Interval[i].second.first)
        {
            cout << "\t|\t";
        }
        cout << All_Interval[i].first << "\t|\t";
    }
    cout << "\n";

    if (All_Interval[0].second.first != 0)
    {
        cout << " 0\t";
        cout << All_Interval[0].second.first << "\t";
    }
    else
    {
        cout << All_Interval[0].second.first << "\t\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << All_Interval[i].second.first <<
"\t\t";
        }
        cout << All_Interval[i].second.second << "\t\t";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process
Scheduling\n\n";
    cout << "+--------------------------------------------
---------------------------------------------------------
-----------+\n";
    cout <<
"|\tProcess\t|\tAT\t|\tBT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT
   |\n";
    cout << "+--------------------------------------------
---------------------------------------------------------
-----------+\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "|\t" << P_Array[i].P_Name << "\t|\t" <<
P_Array[i].AT
             << "\t|\t" << P_Array[i].BT << "\t|\t" <<
P_Array[i].CT
```

```
             << "\t|\t" << P_Array[i].WT << "\t|\t" <<
P_Array[i].TAT
             << "\t|\t" << P_Array[i].RT << "\t|\n";
    }
    cout << "+-------------------------------------------
---------------------------------------------------------
-----------+\n";
}

void Timing(vector<pair<string, pair<int, int>>>
&All_Interval, Process P_Array[], int T_Process)
{
    int size = All_Interval.size();
    for (int i = 0; i < T_Process; i++)
    {
        for (int j = size - 1; j >= 0; j--)
        {
            if (P_Array[i].P_Name ==
All_Interval[j].first)
            {
                P_Array[i].CT =
All_Interval[j].second.second;
                break;
            }
        }
        P_Array[i].TAT = P_Array[i].CT - P_Array[i].AT;
        P_Array[i].WT = P_Array[i].TAT - P_Array[i].BT;
        for (int j = 0; j < size; j++)
        {
            if (P_Array[i].P_Name ==
All_Interval[j].first)
            {
                P_Array[i].RT =
All_Interval[j].second.first - P_Array[i].AT;
                break;
            }
        }
    }
    Print_Bars();
    Chart(P_Array, T_Process);
    Print_Bars();
    Average_Time(P_Array, T_Process);
    Print_Bars();
    GanttChart(All_Interval);
    Print_Bars();
}

vector<pair<string, pair<int, int>>>
Time_Intervals(vector<string> &timeArray)
{
    vector<pair<string, pair<int, int>>>
processTimeInterval;
    for (int i = 0; i < timeArray.size(); i++)
    {
        int end = timeArray.size();
        for (int j = i + 1; j < timeArray.size(); j++)
        {
            if (timeArray[i] != timeArray[j])
            {
                end = j;
                break;
            }
        }
        processTimeInterval.push_back({timeArray[i], {i,
end}});
        i = end - 1;
    }
    return processTimeInterval;
}

void AddTimeToArray(Process process, vector<string>
&timeArray, int timer, int TQ)
{
    for (int i = timer; i < timer + TQ; i++)
    {
        timeArray.push_back(process.P_Name);
    }
}
```

```
void RoundRobin_Preemptive(Process P_Array[], int
T_Process, int TQ)
{
    sort(P_Array, P_Array + T_Process, mycomp);
    queue<Process> que;
    int processIterator = 0;
    vector<string> timeArray;
    que.push(P_Array[0]);
    int timer = P_Array[processIterator].AT;
    if (timer != 0)
    {
        Process pnull;
        pnull.P_Name = "--";
        AddTimeToArray(pnull, timeArray, 0, timer);
    }
    processIterator++;
    while (!que.empty() || processIterator < T_Process)
    {
        if (!que.empty())
        {
            Process processCpuAllocated = que.front();
            que.pop();
            while (processIterator < T_Process && timer +
min(TQ, processCpuAllocated.BT) >=
P_Array[processIterator].AT)
            {
                que.push(P_Array[processIterator++]);
            }
            if (processCpuAllocated.BT > TQ)
            {
                processCpuAllocated.BT -= TQ;
                AddTimeToArray(processCpuAllocated,
timeArray, timer, TQ);
                que.push(processCpuAllocated);
                timer += TQ;
            }
            else
            {
                int remTime = processCpuAllocated.BT;
                AddTimeToArray(processCpuAllocated,
timeArray, timer, remTime);
                timer += remTime;
            }
        }
        else
        {
            timeArray.push_back("--");
            timer++;
            while (processIterator < T_Process && timer
>= P_Array[processIterator].AT)
            {
                que.push(P_Array[processIterator++]);
            }
        }
    }
    vector<pair<string, pair<int, int>>> Intervals =
Time_Intervals(timeArray);
    Timing(Intervals, P_Array, T_Process);
}

int main()
{
    system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Round Robin Process Scheduling
Alogorithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    int TQ;
    cout << "Enter The Time Quantum : ";
    cin >> TQ;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
```

```cpp
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time | \n";

    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
    }

    RoundRobin_Preemptive(P_Array, T_Process, TQ);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
------------------------------------------------------------------------------------------------------------------------
20BCS070_Vicky_Gupta
Round Robin Process Scheduling Alogorithm

------------------------------------------------------------------------------------------------------------------------
Enter The No Of Processes : 4
Enter The Time Quantum : 2

------------------------------------------------------------------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time |
P1 1 4
P2 2 1
P3 3 8
P4 4 1

------------------------------------------------------------------------------------------------------------------------
Various Time's Related To Process Scheduling


+---------------------------------------------------------------------------------------------------------------+
|       Process |       AT       |       BT       |       CT       |       WT       |       TAT      |       RT       |
+---------------------------------------------------------------------------------------------------------------+
|       P1      |       1        |       4        |       8        |       3        |       7        |       0        |
|       P2      |       2        |       1        |       4        |       1        |       2        |       1        |
|       P3      |       3        |       8        |       15       |       4        |       12       |       1        |
|       P4      |       4        |       1        |       9        |       4        |       5        |       4        |
+---------------------------------------------------------------------------------------------------------------+

------------------------------------------------------------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 9
Average Waiting Time -> 3
Average Turn Around Time -> 6.5
Average Respond Time -> 1.5

------------------------------------------------------------------------------------------------------------------------
Gantt Chart For Process Scheduling

|       --      |       P1       |       P2       |       P3       |       P1       |       P4       |       P3       |
0               1                3                4                6                8                9                15

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
Exited..

------------------------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

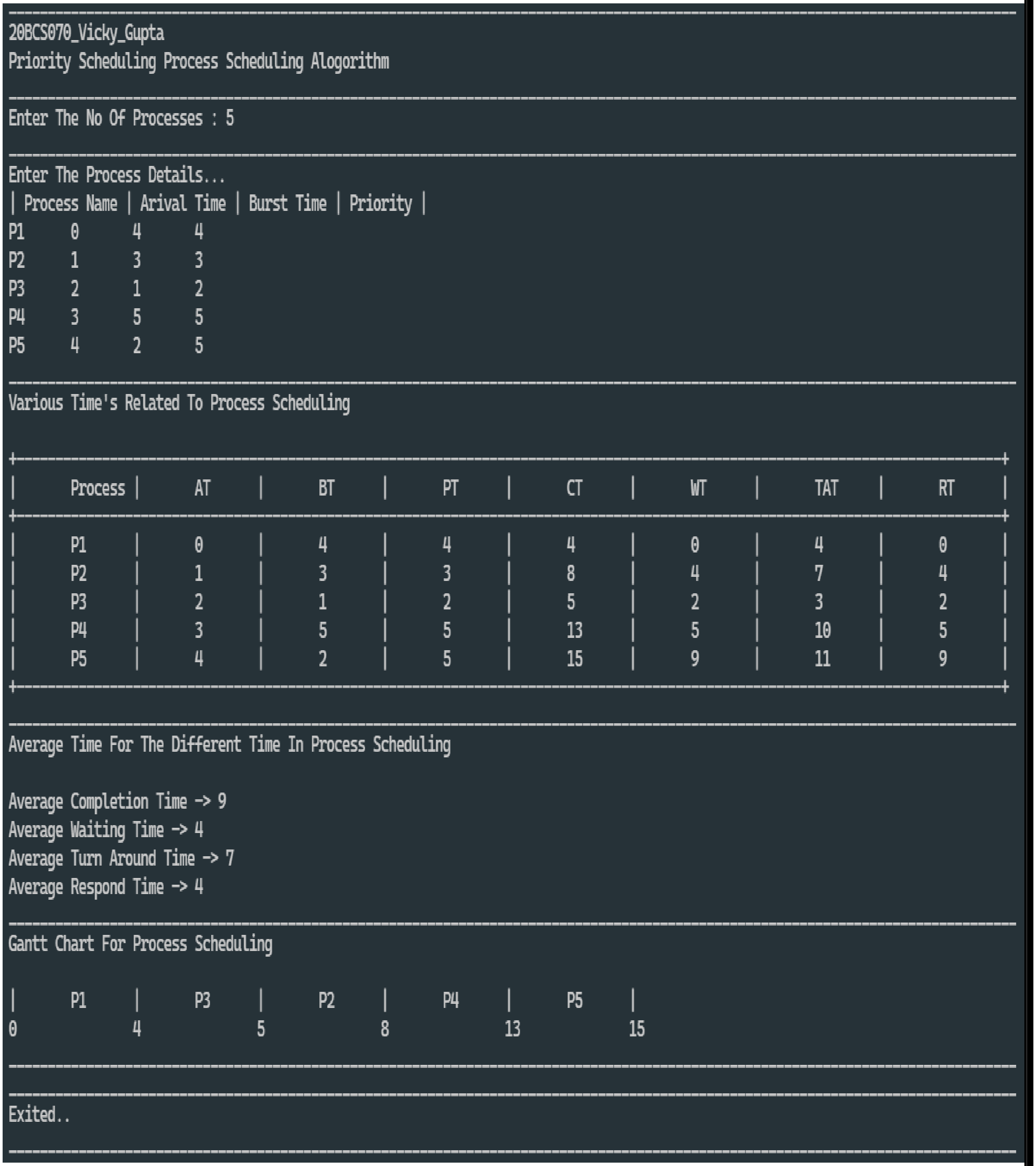
Program - 6

Code :-

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int PT;
    int WT;
    int CT;
    int RT;
    int TAT;
};
```

```
bool mycomp(Process P1, Process P2)
{

    if (P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }
    else if (P1.PT != P2.PT)
    {
        return P1.PT < P2.PT;
    }
    else
    {
        int num1 = stoi(P1.P_Name.substr(1));
        int num2 = stoi(P2.P_Name.substr(1));
        return num1 < num2;
    }
}

struct myCompPT
{
    bool operator()(Process &p1, Process const &p2)
    {
        if (p1.PT != p2.PT)
            return p1.PT > p2.PT;
        else
        {
            int num1 = stoi(p1.P_Name.substr(1));
            int num2 = stoi(p2.P_Name.substr(1));
            return num1 > num2;
        }
    }
};

void Print_Bars()
{
    for (int i = 0; i < 130; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In Process
Scheduling\n\n";

    cout << "Average Completion Time -> " << Av_CT << "\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT << "\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(vector<pair<string, pair<int, int>>>
&All_Interval)
{
    int size = All_Interval.size();
    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << "|\t\t|  ";
    }
    else
    {
        cout << "|\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << "\t|\t";
        }
        cout << All_Interval[i].first << "\t|\t";
    }
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << " 0\t";
        cout << All_Interval[0].second.first << "\t";
    }
    else
    {
        cout << All_Interval[0].second.first << "\t\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << All_Interval[i].second.first << "\t\t";
        }
        cout << All_Interval[i].second.second << "\t\t";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process Scheduling\n\n";
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
    cout <<
"|\tProcess\t|\tAT\t|\tBT\t|\tPT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT
 |\n";
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "|\t" << P_Array[i].P_Name
             << "\t|\t" << P_Array[i].AT
             << "\t|\t" << P_Array[i].BT
             << "\t|\t" << P_Array[i].PT
             << "\t|\t" << P_Array[i].CT
             << "\t|\t" << P_Array[i].WT
             << "\t|\t" << P_Array[i].TAT
             << "\t|\t" << P_Array[i].RT << "\t|\n";
    }
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
}

void Timing(vector<pair<string, pair<int, int>>> &All_Interval,
Process P_Array[], int T_Process)
{
    int size = All_Interval.size();
    for (int i = 0; i < T_Process; i++)
    {
        for (int j = size - 1; j >= 0; j--)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].CT = All_Interval[j].second.second;
                break;
            }
        }
        P_Array[i].TAT = P_Array[i].CT - P_Array[i].AT;
        P_Array[i].WT = P_Array[i].TAT - P_Array[i].BT;
        for (int j = 0; j < size; j++)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].RT = All_Interval[j].second.first -
P_Array[i].AT;
                break;
            }
        }
    }
    Print_Bars();
    Chart(P_Array, T_Process);
    Print_Bars();
    Average_Time(P_Array, T_Process);
    Print_Bars();
    GanttChart(All_Interval);
    Print_Bars();
}

vector<pair<string, pair<int, int>>> Time_Intervals(vector<string>
&timeArray)
{
    vector<pair<string, pair<int, int>>> processTimeInterval;
    for (int i = 0; i < timeArray.size(); i++)
    {
        int end = timeArray.size();
        for (int j = i + 1; j < timeArray.size(); j++)
        {
            if (timeArray[i] != timeArray[j])
            {
                end = j;
                break;
            }
        }
        processTimeInterval.push_back({timeArray[i], {i, end}});
        i = end - 1;
    }
    return processTimeInterval;
}

void AddTimeToArray(Process process, vector<string> &timeArray,
int timer, int BT)
{
    for (int i = timer; i < timer + BT; i++)
    {
        timeArray.push_back(process.P_Name);
    }
}

void Priority_Scheduling(Process P_Array[], int T_Process)
{
    sort(P_Array, P_Array + T_Process, mycomp);
    priority_queue<Process, vector<Process>, myCompPT> pque;
    int processIterator = 0;
    vector<string> timeArray;
    pque.push(P_Array[0]);
    int timer = P_Array[processIterator].AT;
    if (timer != 0)
    {
        Process pnull;
        pnull.P_Name = "--";
        AddTimeToArray(pnull, timeArray, 0, timer);
    }
    processIterator++;
    while (!pque.empty() || processIterator < T_Process)
    {
        if (!pque.empty())
        {
            Process processCpuAllocated = pque.top();
            pque.pop();
            AddTimeToArray(processCpuAllocated, timeArray, timer,
processCpuAllocated.BT);
            timer += processCpuAllocated.BT;
        }
        else
        {
            timeArray.push_back("--");
            timer++;
        }
        while (processIterator < T_Process && timer >=
P_Array[processIterator].AT)
        {
            pque.push(P_Array[processIterator++]);
        }
    }
    vector<pair<string, pair<int, int>>> Intervals =
Time_Intervals(timeArray);
    Timing(Intervals, P_Array, T_Process);
}

int main()
{
    system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Priority Scheduling Process Scheduling Alogoirithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time | Priority
|\n";

    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
        cin >> P_Array[i].PT;
    }
    Priority_Scheduling(P_Array, T_Process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
------------------------------------------------------------------------------------------------------------------------
20BCS070_Vicky_Gupta
Priority Scheduling Process Scheduling Alogorithm

------------------------------------------------------------------------------------------------------------------------
Enter The No Of Processes : 5

------------------------------------------------------------------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time | Priority |
P1      0       4       4
P2      1       3       3
P3      2       1       2
P4      3       5       5
P5      4       2       5

------------------------------------------------------------------------------------------------------------------------
Various Time's Related To Process Scheduling


+---------------------------------------------------------------------------------------------------------------+
|       Process |       AT      |       BT      |       PT      |       CT      |       WT      |       TAT     |       RT      |
+---------------------------------------------------------------------------------------------------------------+
|       P1      |       0       |       4       |       4       |       4       |       0       |       4       |       0       |
|       P2      |       1       |       3       |       3       |       8       |       4       |       7       |       4       |
|       P3      |       2       |       1       |       2       |       5       |       2       |       3       |       2       |
|       P4      |       3       |       5       |       5       |       13      |       5       |       10      |       5       |
|       P5      |       4       |       2       |       5       |       15      |       9       |       11      |       9       |
+---------------------------------------------------------------------------------------------------------------+

------------------------------------------------------------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 9
Average Waiting Time -> 4
Average Turn Around Time -> 7
Average Respond Time -> 4

------------------------------------------------------------------------------------------------------------------------
Gantt Chart For Process Scheduling


|       P1      |       P3      |       P2      |       P4      |       P5      |
0               4               5               8               13              15

------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------
Exited..

------------------------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

Program - 7

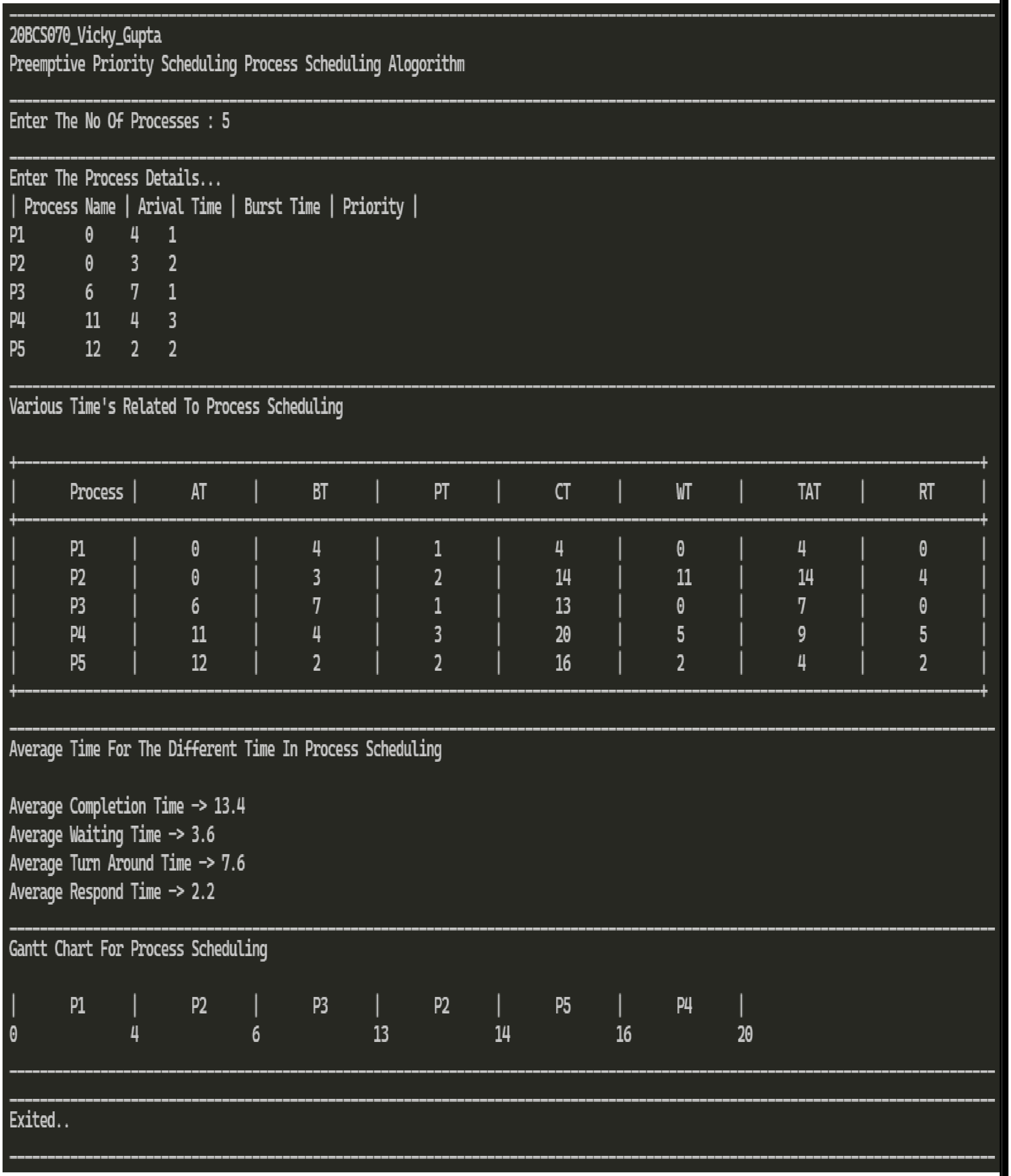
Code :-

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int PT;
    int WT;
    int CT;
    int RT;
    int TAT;
};

bool mycomp(Process P1, Process P2)
```

```
{
    if (P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }
    else if (P1.PT != P2.PT)
    {
        return P1.PT < P2.PT;
    }
    else
    {
        int num1 = stoi(P1.P_Name.substr(1));
        int num2 = stoi(P2.P_Name.substr(1));
        return num1 < num2;
    }
}

struct myCompPT
{
    bool operator()(Process &p1, Process const &p2)
    {
        if (p1.PT != p2.PT)
            return p1.PT > p2.PT;
        else
        {
            int num1 = stoi(p1.P_Name.substr(1));
            int num2 = stoi(p2.P_Name.substr(1));
            return num1 > num2;
        }
    }
};

void Print_Bars()
{
    for (int i = 0; i < 130; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In Process
Scheduling\n\n";

    cout << "Average Completion Time -> " << Av_CT << "\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT << "\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(vector<pair<string, pair<int, int>>>
&All_Interval)
{
    int size = All_Interval.size();
    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << "|\t\t|  ";
    }
    else
    {
        cout << "|\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << "\t|\t";
        }
        cout << All_Interval[i].first << "\t|\t";
    }
    cout << "\n";

    if (All_Interval[0].second.first != 0)
    {
        cout << " 0\t";
        cout << All_Interval[0].second.first << "\t";
    }
    else
    {
        cout << All_Interval[0].second.first << "\t\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << All_Interval[i].second.first << "\t\t";
        }
        cout << All_Interval[i].second.second << "\t\t";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process Scheduling\n\n";
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
    cout <<
"|\tProcess\t|\tAT\t|\tBT\t|\tPT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT
 |\n";
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "|\t" << P_Array[i].P_Name
             << "\t|\t" << P_Array[i].AT
             << "\t|\t" << P_Array[i].BT
             << "\t|\t" << P_Array[i].PT
             << "\t|\t" << P_Array[i].CT
             << "\t|\t" << P_Array[i].WT
             << "\t|\t" << P_Array[i].TAT
             << "\t|\t" << P_Array[i].RT << "\t|\n";
    }
    cout << "+----------------------------------------------------
------------------------------------------------------------------
---------+\n";
}

void Timing(vector<pair<string, pair<int, int>>> &All_Interval,
Process P_Array[], int T_Process)
{
    int size = All_Interval.size();
    for (int i = 0; i < T_Process; i++)
    {
        for (int j = size - 1; j >= 0; j--)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].CT = All_Interval[j].second.second;
                break;
            }
        }
        P_Array[i].TAT = P_Array[i].CT - P_Array[i].AT;
        P_Array[i].WT = P_Array[i].TAT - P_Array[i].BT;
        for (int j = 0; j < size; j++)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].RT = All_Interval[j].second.first -
P_Array[i].AT;
                break;
            }
        }
    }
    Print_Bars();
    Chart(P_Array, T_Process);
    Print_Bars();
    Average_Time(P_Array, T_Process);
    Print_Bars();
    GanttChart(All_Interval);
    Print_Bars();
}

vector<pair<string, pair<int, int>>> Time_Intervals(vector<string>
&timeArray)
{
    vector<pair<string, pair<int, int>>> processTimeInterval;
    for (int i = 0; i < timeArray.size(); i++)
    {
        int end = timeArray.size();
        for (int j = i + 1; j < timeArray.size(); j++)
        {
            if (timeArray[i] != timeArray[j])
            {
                end = j;
                break;
            }
        }
        processTimeInterval.push_back({timeArray[i], {i, end}});
        i = end - 1;
    }
    return processTimeInterval;
}

void AddTimeToArray(Process process, vector<string> &timeArray,
int timer, int BT)
{
    for (int i = timer; i < timer + BT; i++)
    {
        timeArray.push_back(process.P_Name);
    }
}

void Preemptive_Priority_Scheduling(Process P_Array[], int
T_Process)
{
    sort(P_Array, P_Array + T_Process, mycomp);
    priority_queue<Process, vector<Process>, myCompPT> pque;
    int processIterator = 0;
    vector<string> timeArray;
    pque.push(P_Array[0]);
    int timer = P_Array[processIterator].AT;
    if (timer != 0)
    {
        Process pnull;
        pnull.P_Name = "--";
        AddTimeToArray(pnull, timeArray, 0, timer);
    }
    processIterator++;
    while (!pque.empty() || processIterator < T_Process)
    {
        if (!pque.empty())
        {
            Process processCpuAllocated = pque.top();
            pque.pop();
            AddTimeToArray(processCpuAllocated, timeArray, timer,
1);
            timer += 1;
            processCpuAllocated.BT--;
            if (processCpuAllocated.BT != 0)
            {
                pque.push(processCpuAllocated);
            }
        }
        else
        {
            timeArray.push_back("--");
            timer++;
        }
        while (processIterator < T_Process && timer >=
P_Array[processIterator].AT)
        {
            pque.push(P_Array[processIterator++]);
        }
    }
    vector<pair<string, pair<int, int>>> Intervals =
Time_Intervals(timeArray);
    Timing(Intervals, P_Array, T_Process);
}

int main()
{
    system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Preemptive Priority Scheduling Process Scheduling
Alogorithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time | Priority
|\n";
```

```
    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
        cin >> P_Array[i].PT;
    }

    Preemptive_Priority_Scheduling(P_Array, T_Process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
--------------------------------------------------------------------------------------------------
20BCS070_Vicky_Gupta
Preemptive Priority Scheduling Process Scheduling Alogorithm

--------------------------------------------------------------------------------------------------
Enter The No Of Processes : 5

--------------------------------------------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time | Priority |
P1       0    4   1
P2       0    3   2
P3       6    7   1
P4       11   4   3
P5       12   2   2

--------------------------------------------------------------------------------------------------
Various Time's Related To Process Scheduling


+------------------------------------------------------------------------------------------------+
|      Process |      AT      |      BT      |      PT      |      CT      |      WT      |      TAT     |      RT      |
+------------------------------------------------------------------------------------------------+
|      P1      |      0       |      4       |      1       |      4       |      0       |      4       |      0       |
|      P2      |      0       |      3       |      2       |      14      |      11      |      14      |      4       |
|      P3      |      6       |      7       |      1       |      13      |      0       |      7       |      0       |
|      P4      |      11      |      4       |      3       |      20      |      5       |      9       |      5       |
|      P5      |      12      |      2       |      2       |      16      |      2       |      4       |      2       |
+------------------------------------------------------------------------------------------------+

--------------------------------------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 13.4
Average Waiting Time -> 3.6
Average Turn Around Time -> 7.6
Average Respond Time -> 2.2

--------------------------------------------------------------------------------------------------
Gantt Chart For Process Scheduling

|      P1      |      P2      |      P3      |      P2      |      P5      |      P4      |
0              4              6              13             14             16             20

--------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------
Exited..

--------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

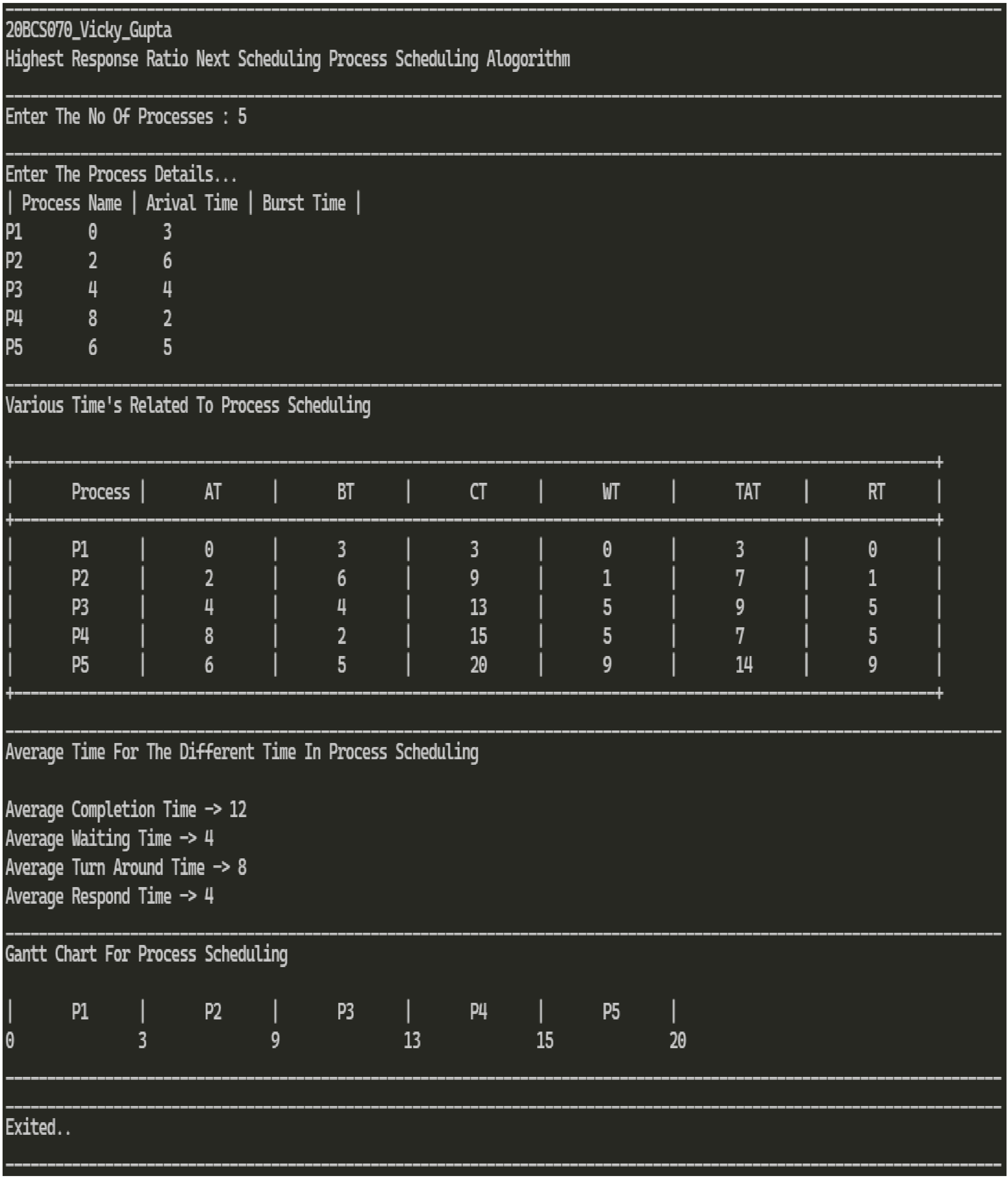
Program - 8

Code :-

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

struct Process
{
    string P_Name;
    int AT;
    int BT;
    int WT;
    int CT;
    int RT;
    int TAT;
};
int timer = 0;

bool mycomp(Process P1, Process P2)
```

```
{
    if (timer < max(P1.AT, P2.AT) && P1.AT != P2.AT)
    {
        return P1.AT < P2.AT;
    }

    double rr1 = ((timer - P1.AT) + P1.BT) / (double)P1.BT;
    double rr2 = ((timer - P2.AT) + P2.BT) / (double)P2.BT;
    if (rr1 != rr2)
    {
        return rr1 > rr2;
    }
    int num1 = stoi(P1.P_Name.substr(1));
    int num2 = stoi(P2.P_Name.substr(1));
    return num1 < num2;
}

void Print_Bars()
{
    for (int i = 0; i < 120; i++)
        cout << "_";
    cout << "\n";
}

void Average_Time(Process P_Array[], int T_Process)
{
    double Av_CT = 0, Av_RT = 0, Av_WT = 0, Av_TAT = 0;
    for (int i = 0; i < T_Process; i++)
    {
        Av_CT += P_Array[i].CT;
        Av_RT += P_Array[i].RT;
        Av_TAT += P_Array[i].TAT;
        Av_WT += P_Array[i].WT;
    }
    Av_WT /= T_Process;
    Av_TAT /= T_Process;
    Av_RT /= T_Process;
    Av_CT /= T_Process;
    cout << "Average Time For The Different Time In Process 
Scheduling\n\n";

    cout << "Average Completion Time -> " << Av_CT << "\n";
    cout << "Average Waiting Time -> " << Av_WT << "\n";
    cout << "Average Turn Around Time -> " << Av_TAT << "\n";
    cout << "Average Respond Time -> " << Av_RT << "\n";
}

void GanttChart(vector<pair<string, pair<int, int>>>
&All_Interval)
{
    int size = All_Interval.size();
    cout << "Gantt Chart For Process Scheduling\n";
    cout << "\n";
    if (All_Interval[0].second.first != 0)
    {
        cout << "|\t\t|  ";
    }
    else
    {
        cout << "|\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << "\t|\t";
        }
        cout << All_Interval[i].first << "\t|\t";
    }
    cout << "\n";

    if (All_Interval[0].second.first != 0)
    {
        cout << " 0\t";
        cout << All_Interval[0].second.first << "\t";
    }
    else
    {
        cout << All_Interval[0].second.first << "\t\t";
    }

    for (int i = 0; i < size; i++)
    {
        if (i != 0 && All_Interval[i - 1].second.second <
All_Interval[i].second.first)
        {
            cout << All_Interval[i].second.first << "\t\t";
        }
        cout << All_Interval[i].second.second << "\t\t";
    }
    cout << "\n";
}

void Chart(Process P_Array[], int T_Process)
{
    cout << "Various Time's Related To Process Scheduling\n\n";
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
    cout << 
"|\tProcess\t|\tAT\t|\tBT\t|\tCT\t|\tWT\t|\tTAT\t|\tRT      |\n";
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
    for (int i = 0; i < T_Process; i++)
    {
        cout << "|\t" << P_Array[i].P_Name
             << "\t|\t" << P_Array[i].AT
             << "\t|\t" << P_Array[i].BT
             << "\t|\t" << P_Array[i].CT
             << "\t|\t" << P_Array[i].WT
             << "\t|\t" << P_Array[i].TAT
             << "\t|\t" << P_Array[i].RT << "\t|\n";
    }
    cout << "+----------------------------------------------------
-----------------------------------------------------------+\n";
}

void Timing(vector<pair<string, pair<int, int>>> &All_Interval, 
Process P_Array[], int T_Process)
{
    int size = All_Interval.size();
    for (int i = 0; i < T_Process; i++)
    {
        for (int j = size - 1; j >= 0; j--)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].CT = All_Interval[j].second.second;
                break;
            }
        }
        P_Array[i].TAT = P_Array[i].CT - P_Array[i].AT;
        P_Array[i].WT = P_Array[i].TAT - P_Array[i].BT;
        for (int j = 0; j < size; j++)
        {
            if (P_Array[i].P_Name == All_Interval[j].first)
            {
                P_Array[i].RT = All_Interval[j].second.first -
P_Array[i].AT;
                break;
            }
        }
    }
    Print_Bars();
    Chart(P_Array, T_Process);
    Print_Bars();
    Average_Time(P_Array, T_Process);
    Print_Bars();
    GanttChart(All_Interval);
    Print_Bars();
}

vector<pair<string, pair<int, int>>> Time_Intervals(vector<string>
&timeArray)
{
    vector<pair<string, pair<int, int>>> processTimeInterval;
    for (int i = 0; i < timeArray.size(); i++)
    {
        int end = timeArray.size();
        for (int j = i + 1; j < timeArray.size(); j++)
        {
            if (timeArray[i] != timeArray[j])
            {
                end = j;
                break;
            }
        }
        processTimeInterval.push_back({timeArray[i], {i, end}});
        i = end - 1;
    }
    return processTimeInterval;
}

void AddTimeToArray(Process process, vector<string> &timeArray,
int timer, int BT)
{
    for (int i = timer; i < timer + BT; i++)
    {
        timeArray.push_back(process.P_Name);
    }
}

void HRRN(Process P_Array[], int T_Process)
{
    vector<Process> New_P_Array(P_Array, P_Array + T_Process);
    sort(New_P_Array.begin(), New_P_Array.end(), mycomp);
    vector<string> timeArray;
    timer = New_P_Array[0].AT;
    if (timer != 0)
    {
        Process pnull;
        pnull.P_Name = "--";
        AddTimeToArray(pnull, timeArray, 0, timer);
    }
    while (!New_P_Array.empty())
    {

        Process processCpuAllocated = New_P_Array[0];
        New_P_Array.erase(New_P_Array.begin());
        while (timer < processCpuAllocated.AT)
        {
            timeArray.push_back("--");
            timer++;
        }
        AddTimeToArray(processCpuAllocated, timeArray, timer,
processCpuAllocated.BT);
        timer += processCpuAllocated.BT;
        sort(New_P_Array.begin(), New_P_Array.end(), mycomp);
    }
    vector<pair<string, pair<int, int>>> Intervals =
Time_Intervals(timeArray);
    Timing(Intervals, P_Array, T_Process);
}

int main()
{
    // system("cls");
    Print_Bars();
    cout << "20BCS070_Vicky_Gupta\n";
    cout << "Highest Response Ratio Next Scheduling Process
Scheduling Alogorithm\n";
    Print_Bars();
    int T_Process;
    cout << "Enter The No Of Processes : ";
    cin >> T_Process;
    fflush(stdin);
    Process P_Array[T_Process];
    Print_Bars();
    cout << "Enter The Process Details...\n";
    cout << "| Process Name | Arival Time | Burst Time |\n";

    for (int i = 0; i < T_Process; i++)
    {
        cin >> P_Array[i].P_Name;
        cin >> P_Array[i].AT;
        cin >> P_Array[i].BT;
    }

    HRRN(P_Array, T_Process);
    Print_Bars();
    cout << "Exited..\n";
    Print_Bars();
    return 0;
}
```

Output :-



```
------------------------------------------------------------------------------------------------------------------------
20BCS070_Vicky_Gupta
Highest Response Ratio Next Scheduling Process Scheduling Alogorithm

------------------------------------------------------------------------------------------------------------------------
Enter The No Of Processes : 5

------------------------------------------------------------------------------------------------------------------------
Enter The Process Details...
| Process Name | Arival Time | Burst Time |
P1       0       3
P2       2       6
P3       4       4
P4       8       2
P5       6       5

------------------------------------------------------------------------------------------------------------------------
Various Time's Related To Process Scheduling


+---------------------------------------------------------------------------------------------------------------+
|       Process |       AT      |       BT      |       CT      |       WT      |       TAT     |       RT      |
+---------------------------------------------------------------------------------------------------------------+
|       P1      |       0       |       3       |       3       |       0       |       3       |       0       |
|       P2      |       2       |       6       |       9       |       1       |       7       |       1       |
|       P3      |       4       |       4       |       13      |       5       |       9       |       5       |
|       P4      |       8       |       2       |       15      |       5       |       7       |       5       |
|       P5      |       6       |       5       |       20      |       9       |       14      |       9       |
+---------------------------------------------------------------------------------------------------------------+

------------------------------------------------------------------------------------------------------------------------
Average Time For The Different Time In Process Scheduling

Average Completion Time -> 12
Average Waiting Time -> 4
Average Turn Around Time -> 8
Average Respond Time -> 4

------------------------------------------------------------------------------------------------------------------------
Gantt Chart For Process Scheduling

|       P1      |       P2      |       P3      |       P4      |       P5      |
0               3               9               13              15              20

------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------
Exited..

------------------------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

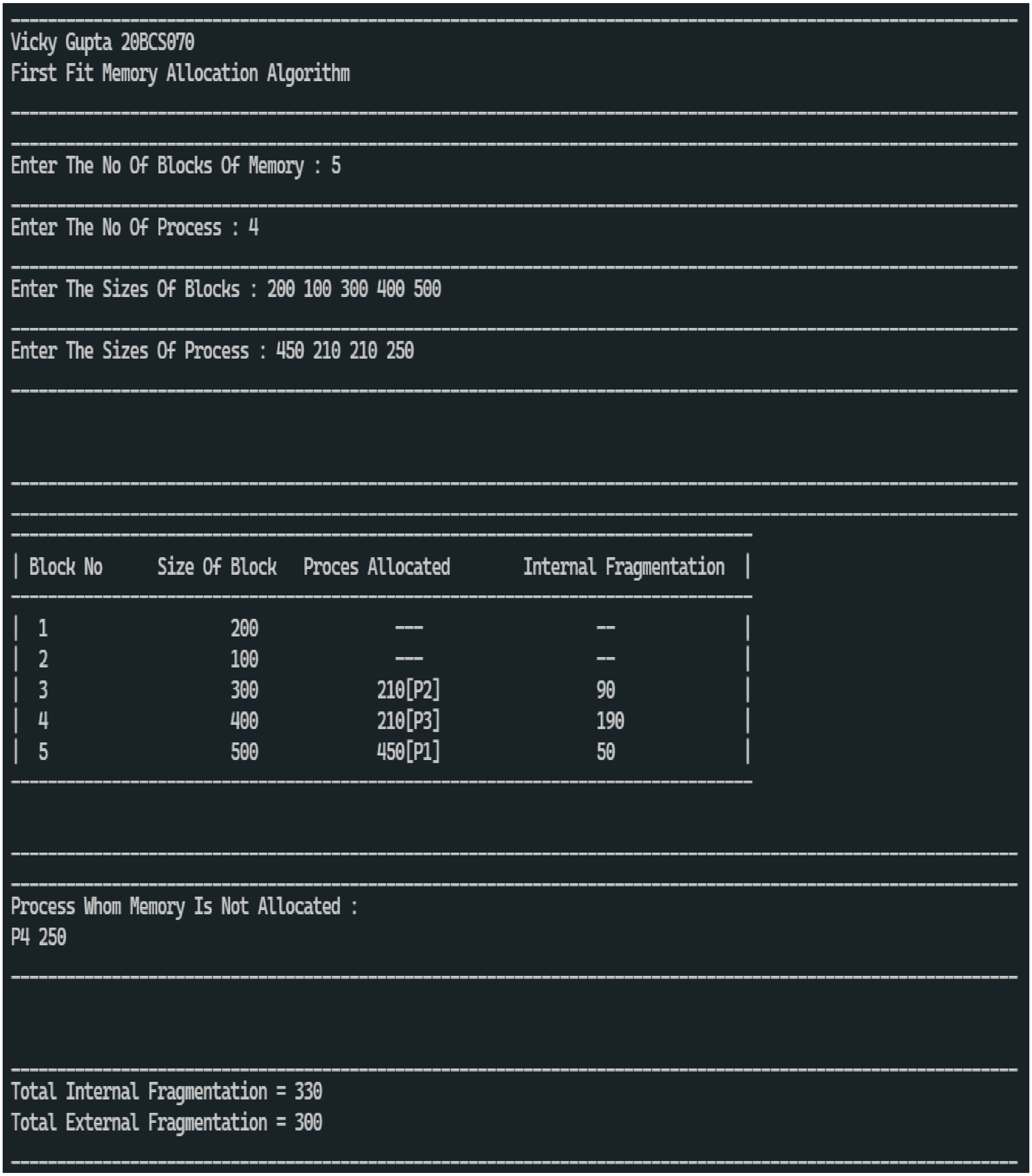
Program - 9

Code :-

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

typedef long long ll;

struct memoryBlocks
{
    bool isAllocated;
    int blockSize;
    int processSize;
    int internalFrag;
    string processName;
};

void printLines()
{
    for (int i = 0; i < 110; i++)
    {
        cout << "_";
    }
    cout << "\n";
}

void Display(vector<memoryBlocks> &memBlocks, int noOfBlocks, int
internalFrag, int externalFrag, vector<pair<int, string>>
&leftProcess)
{
    cout << "-----------------------------------------------------
----------------------------\n";
    cout << "| Block No\t"
         << "Size Of Block\t"
         << "Proces Allocated\t"
         << "Internal Fragmentation  |\n";
    cout << "-----------------------------------------------------
----------------------------\n";
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        if (memBlocks[bindx].isAllocated == false)
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << "  ---  "
                 << "\t\t\t"
                 << "--"
                 << "\t\t|\n";

        else
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << memBlocks[bindx].processSize << "[" <<
memBlocks[bindx].processName << "]"
                 << "\t\t\t" << memBlocks[bindx].internalFrag <<
"\t\t|\n";
    }
    cout << "-----------------------------------------------------
----------------------------\n";

    cout << "\n";
    printLines();
    printLines();
    if (!leftProcess.empty())
    {
        cout << "Process Whom Memory Is Not Allocated : \n";
        for (int lindx = 0; lindx < leftProcess.size(); lindx++)
        {
            cout << leftProcess[lindx].second << " " <<
leftProcess[lindx].first << "\n";
        }
    }

    printLines();
    cout << "\n\n";
    printLines();
    cout << "Total Internal Fragmentation = " << internalFrag <<
"\n";
    cout << "Total External Fragmentation = " << externalFrag <<
"\n";
    printLines();
}

void First_Fit(vector<memoryBlocks> &memBlocks, int noOfBlocks,
vector<pair<int, string>> &processSizes, int noOfProcess)
{
    vector<pair<int, string>> leftProcess;
    for (int pindx = 0; pindx < noOfProcess; pindx++)
    {
        bool isProcessMemAllocated = false;
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true ||
memBlocks[bindx].blockSize < processSizes[pindx].first)
                continue;

            isProcessMemAllocated = true;

            memBlocks[bindx].isAllocated = true;
            memBlocks[bindx].processName =
processSizes[pindx].second;
            memBlocks[bindx].processSize =
processSizes[pindx].first;
            memBlocks[bindx].internalFrag =
memBlocks[bindx].blockSize - processSizes[pindx].first;
            break;
        }
        if (isProcessMemAllocated == false)
        {
            leftProcess.push_back(processSizes[pindx]);
        }
    }
```

```cpp
    int externalFrag = 0, internalFrag = 0;
    if (leftProcess.empty() == false)
    {
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true)
                continue;
            externalFrag += memBlocks[bindx].blockSize;
        }
    }
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        internalFrag += memBlocks[bindx].internalFrag;
    }
    Display(memBlocks, noOfBlocks, internalFrag, externalFrag,
leftProcess);
}

int main()
{
    system("cls");

    printLines();
    cout << "Vicky Gupta 20BCS070\n";
    cout << "First Fit Memory Allocation Algorithm\n";
    printLines();
    printLines();

    int noOfBlocks;
    cout << "Enter The No Of Blocks Of Memory : ";
    cin >> noOfBlocks;

    printLines();
    int noOfProcess;
    cout << "Enter The No Of Process : ";
    cin >> noOfProcess;

    printLines();
    vector<memoryBlocks> memBlocks(noOfBlocks);
    cout << "Enter The Sizes Of Blocks : ";
    for (int i = 0; i < noOfBlocks; i++)
    {
        cin >> memBlocks[i].blockSize;
        memBlocks[i].isAllocated = false;
        memBlocks[i].processSize = 0;
        memBlocks[i].processName = "";
        memBlocks[i].internalFrag = 0;
    }

    printLines();
    vector<pair<int, string>> processSizes(noOfProcess);
    cout << "Enter The Sizes Of Process : ";
    for (int i = 0; i < noOfProcess; i++)
    {
        cin >> processSizes[i].first;
        processSizes[i].second = "P";
        processSizes[i].second += to_string(i + 1);
    }
    printLines();

    cout << "\n\n";
    printLines();
    printLines();
    First_Fit(memBlocks, noOfBlocks, processSizes, noOfProcess);
    return 0;
}
```

Output :-



```
--------------------------------------------------------------------------------------------------------------
Vicky Gupta 20BCS070
First Fit Memory Allocation Algorithm

--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
Enter The No Of Blocks Of Memory : 5

--------------------------------------------------------------------------------------------------------------
Enter The No Of Process : 4

--------------------------------------------------------------------------------------------------------------
Enter The Sizes Of Blocks : 200 100 300 400 500

--------------------------------------------------------------------------------------------------------------
Enter The Sizes Of Process : 450 210 210 250

--------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
| Block No      Size Of Block   Proces Allocated        Internal Fragmentation  |
---------------------------------------------------------------------------------

|  1                 200             ---                  --                   |
|  2                 100             ---                  --                   |
|  3                 300           210[P2]                90                   |
|  4                 400           210[P3]                190                  |
|  5                 500           450[P1]                50                   |
---------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
Process Whom Memory Is Not Allocated :
P4 250

--------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------
Total Internal Fragmentation = 330
Total External Fragmentation = 300

--------------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

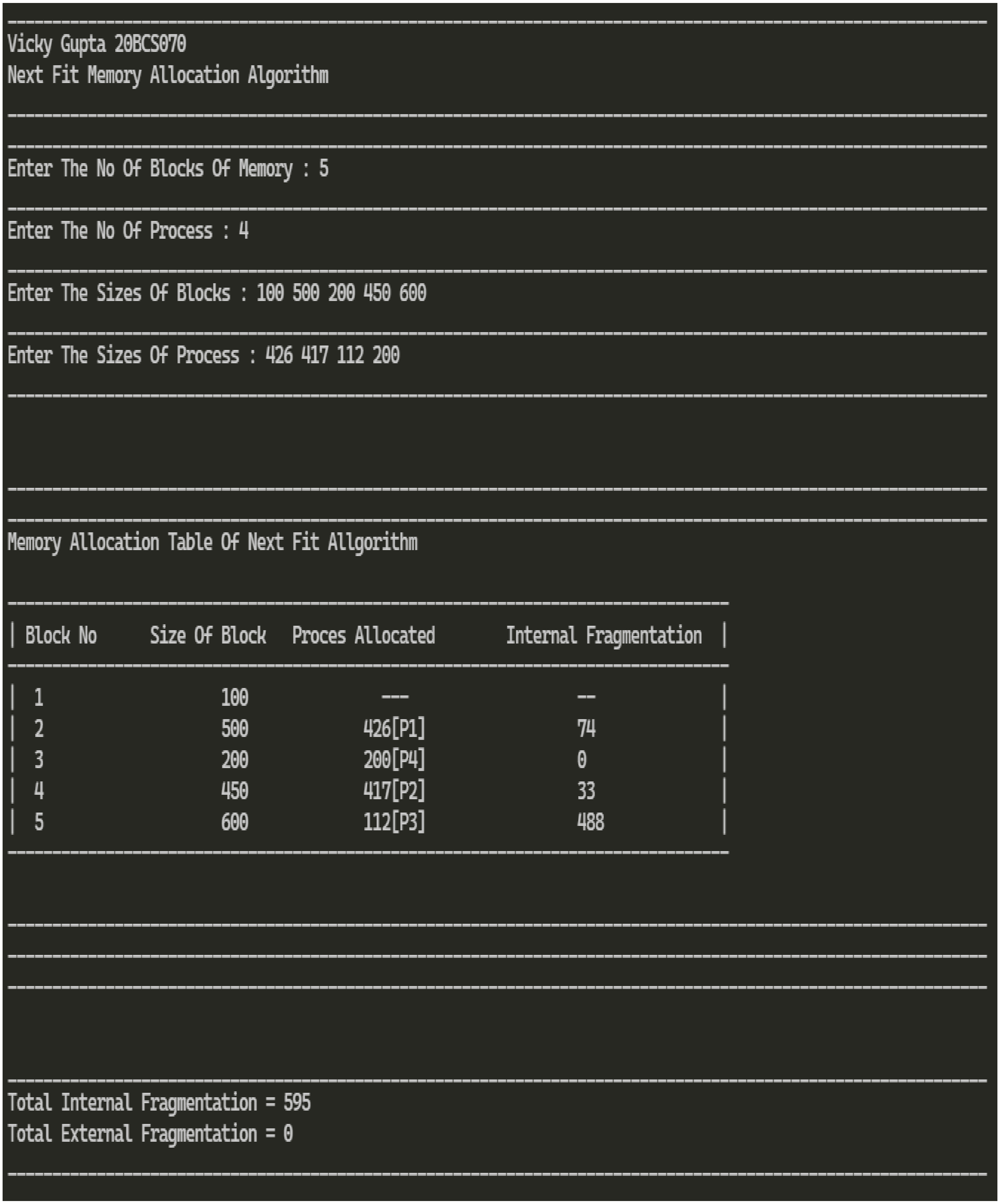
Program - 10

Code :-

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

typedef long long ll;

struct memoryBlocks
{
    bool isAllocated;
    int blockSize;
    int processSize;
    int internalFrag;
    string processName;
};

void printLines()
{
    for (int i = 0; i < 110; i++)
    {
        cout << "_";
    }
    cout << "\n";
}

void Display(vector<memoryBlocks> &memBlocks, int noOfBlocks, int
internalFrag, int externalFrag, vector<pair<int, string>>
&leftProcess)
{
    cout << "Memory Allocation Table Of Next Fit Allgorithm\n"
         << "\n";
    cout << "-----------------------------------------------------
----------------------------\n";
    cout << "| Block No\t"
         << "Size Of Block\t"
         << "Proces Allocated\t"
         << "Internal Fragmentation  |\n";
    cout << "-----------------------------------------------------
----------------------------\n";
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        if (memBlocks[bindx].isAllocated == false)
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << "  ---  "
                 << "\t\t\t"
                 << "--"
                 << "\t\t|\n";

        else
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << memBlocks[bindx].processSize << "[" <<
memBlocks[bindx].processName << "]"
                 << "\t\t\t" << memBlocks[bindx].internalFrag <<
"\t\t|\n";
    }
    cout << "-----------------------------------------------------
----------------------------\n";

    cout << "\n";
    printLines();
    printLines();
    if (!leftProcess.empty())
    {
        cout << "Process Whom Memory Is Not Allocated : \n";
        for (int lindx = 0; lindx < leftProcess.size(); lindx++)
        {
            cout << leftProcess[lindx].second << " " <<
leftProcess[lindx].first << "\n";
        }
    }

    printLines();
    cout << "\n\n";
    printLines();
    cout << "Total Internal Fragmentation = " << internalFrag <<
"\n";
    cout << "Total External Fragmentation = " << externalFrag <<
"\n";
    printLines();
}

void Next_Fit(vector<memoryBlocks> &memBlocks, int noOfBlocks,
vector<pair<int, string>> &processSizes, int noOfProcess)
{
    vector<pair<int, string>> leftProcess;
    int memIter = -1;
    for (int pindx = 0; pindx < noOfProcess; pindx++)
    {
        bool isProcessMemAllocated = false;
        int bindx = memIter;
        bindx++;
        for (bindx; bindx != memIter; bindx = (bindx + 1) %
noOfBlocks)
        {
            if (memBlocks[bindx].isAllocated == true ||
memBlocks[bindx].blockSize < processSizes[pindx].first)
                continue;

            isProcessMemAllocated = true;

            memBlocks[bindx].isAllocated = true;
            memBlocks[bindx].processName =
processSizes[pindx].second;
            memBlocks[bindx].processSize =
processSizes[pindx].first;
            memBlocks[bindx].internalFrag =
memBlocks[bindx].blockSize - processSizes[pindx].first;
            break;
```

```cpp
        }
        memIter = bindx;
        if (isProcessMemAllocated == false)
        {
            leftProcess.push_back(processSizes[pindx]);
        }
    }
    int externalFrag = 0, internalFrag = 0;
    if (leftProcess.empty() == false)
    {
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true)
                continue;
            externalFrag += memBlocks[bindx].blockSize;
        }
    }
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        internalFrag += memBlocks[bindx].internalFrag;
    }
    Display(memBlocks, noOfBlocks, internalFrag, externalFrag,
leftProcess);
}

int main()
{
    system("cls");

    printLines();
    cout << "Vicky Gupta 20BCS070\n";
    cout << "Next Fit Memory Allocation Algorithm\n";
    printLines();
    printLines();

    int noOfBlocks;
    cout << "Enter The No Of Blocks Of Memory : ";
    cin >> noOfBlocks;

    printLines();
    int noOfProcess;
    cout << "Enter The No Of Process : ";
    cin >> noOfProcess;

    printLines();
    vector<memoryBlocks> memBlocks(noOfBlocks);
    cout << "Enter The Sizes Of Blocks : ";
    for (int i = 0; i < noOfBlocks; i++)
    {
        cin >> memBlocks[i].blockSize;
        memBlocks[i].isAllocated = false;
        memBlocks[i].processSize = 0;
        memBlocks[i].processName = "";
        memBlocks[i].internalFrag = 0;
    }

    printLines();
    vector<pair<int, string>> processSizes(noOfProcess);
    cout << "Enter The Sizes Of Process : ";
    for (int i = 0; i < noOfProcess; i++)
    {
        cin >> processSizes[i].first;
        processSizes[i].second = "P";
        processSizes[i].second += to_string(i + 1);
    }
    printLines();

    cout << "\n\n";
    printLines();
    printLines();
    Next_Fit(memBlocks, noOfBlocks, processSizes, noOfProcess);
    return 0;
}
```

Output :-



```
-----------------------------------------------------------------------------------------------------------
Vicky Gupta 20BCS070
Next Fit Memory Allocation Algorithm

-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
Enter The No Of Blocks Of Memory : 5

-----------------------------------------------------------------------------------------------------------
Enter The No Of Process : 4

-----------------------------------------------------------------------------------------------------------
Enter The Sizes Of Blocks : 100 500 200 450 600

-----------------------------------------------------------------------------------------------------------
Enter The Sizes Of Process : 426 417 112 200

-----------------------------------------------------------------------------------------------------------



-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
Memory Allocation Table Of Next Fit Allgorithm


-----------------------------------------------------------------------------
| Block No      Size Of Block   Proces Allocated        Internal Fragmentation  |
-----------------------------------------------------------------------------

|  1                100             ---                 --                  |
|  2                500           426[P1]               74                  |
|  3                200           200[P4]               0                   |
|  4                450           417[P2]               33                  |
|  5                600           112[P3]               488                 |
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------



-----------------------------------------------------------------------------------------------------------
Total Internal Fragmentation = 595
Total External Fragmentation = 0

-----------------------------------------------------------------------------------------------------------
```

Operating System Lab

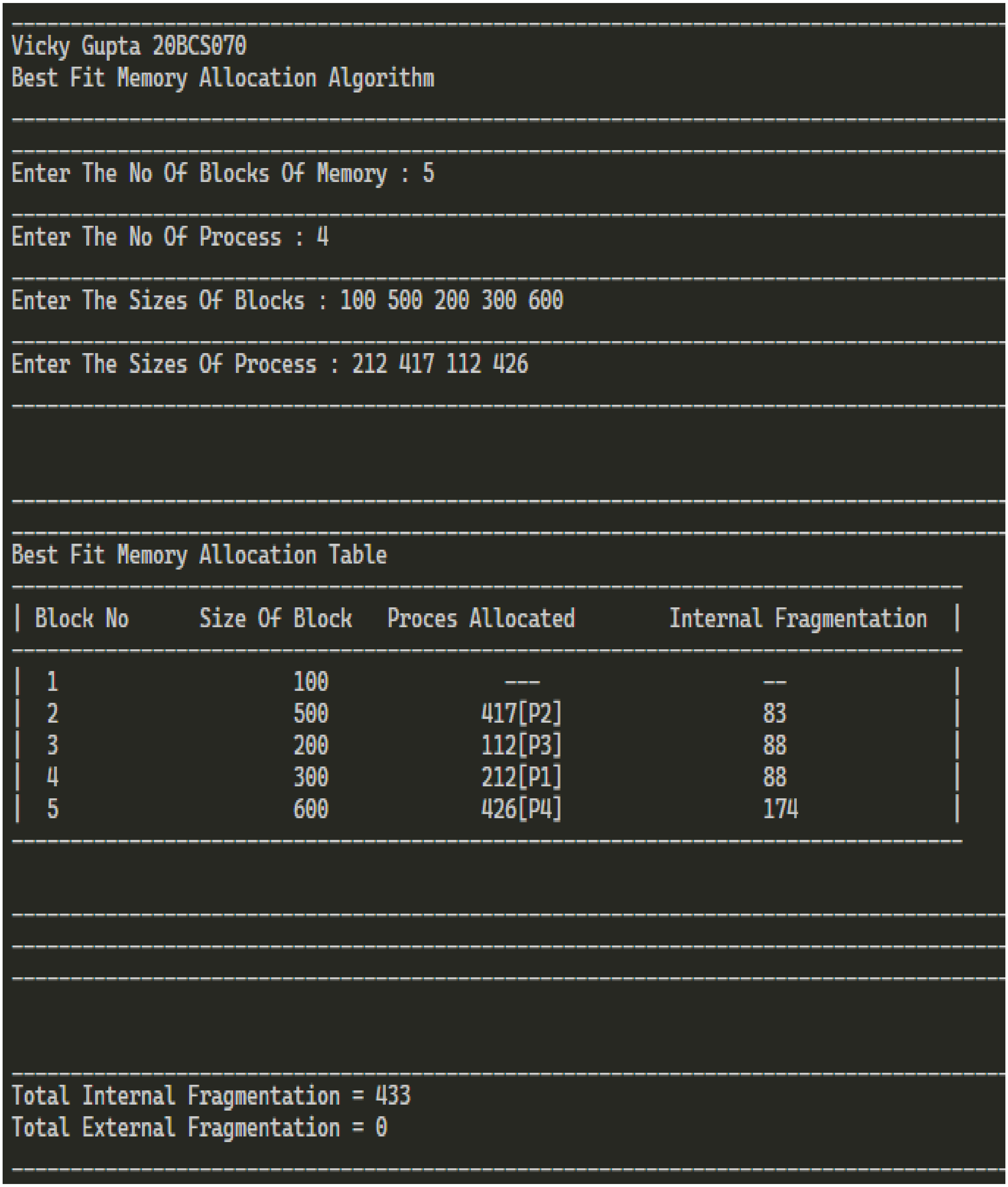
CEN-493

Program - 11

Code :-

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

typedef long long ll;

struct memoryBlocks
{
    bool isAllocated;
    int blockSize;
    int processSize;
    int internalFrag;
    string processName;
};

void printLines()
{
    for (int i = 0; i < 110; i++)
    {
        cout << "_";
    }
    cout << "\n";
}

void Display(vector<memoryBlocks> &memBlocks, int noOfBlocks, int
internalFrag, int externalFrag, vector<pair<int, string>>
&leftProcess)
{
    cout << "Best Fit Memory Allocation Table \n";
    cout << "-----------------------------------------------------
----------------------------\n";
    cout << "| Block No\t"
         << "Size Of Block\t"
         << "Proces Allocated\t"
         << "Internal Fragmentation  |\n";
    cout << "-----------------------------------------------------
----------------------------\n";
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        if (memBlocks[bindx].isAllocated == false)
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << "  ---  "
                 << "\t\t\t"
                 << "--"
                 << "\t\t|\n";

        else
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << memBlocks[bindx].processSize << "[" <<
memBlocks[bindx].processName << "]"
                 << "\t\t\t" << memBlocks[bindx].internalFrag <<
"\t\t|\n";
    }
    cout << "-----------------------------------------------------
----------------------------\n";

    cout << "\n";
    printLines();
    printLines();
    if (!leftProcess.empty())
    {
        cout << "Process Whom Memory Is Not Allocated : \n";
        for (int lindx = 0; lindx < leftProcess.size(); lindx++)
        {
            cout << leftProcess[lindx].second << " " <<
leftProcess[lindx].first << "\n";
        }
    }

    printLines();
    cout << "\n\n";
    printLines();
    cout << "Total Internal Fragmentation = " << internalFrag <<
"\n";
    cout << "Total External Fragmentation = " << externalFrag <<
"\n";
    printLines();
}

void Best_Fit(vector<memoryBlocks> &memBlocks, int noOfBlocks,
vector<pair<int, string>> &processSizes, int noOfProcess)
{
    vector<pair<int, string>> leftProcess;
    for (int pindx = 0; pindx < noOfProcess; pindx++)
    {
        bool isProcessMemAllocated = false;
        int emptyBlock = 0, bestBlockSize = 1e9;
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true ||
memBlocks[bindx].blockSize < processSizes[pindx].first)
                continue;

            isProcessMemAllocated = true;
            if (bestBlockSize > memBlocks[bindx].blockSize)
            {
                emptyBlock = bindx;
                bestBlockSize = memBlocks[bindx].blockSize;
            }
        }
        if (isProcessMemAllocated == false)
        {
            leftProcess.push_back(processSizes[pindx]);
        }
        else
        {
            memBlocks[emptyBlock].isAllocated = true;
            memBlocks[emptyBlock].processName =
processSizes[pindx].second;
            memBlocks[emptyBlock].processSize =
processSizes[pindx].first;
            memBlocks[emptyBlock].internalFrag =
memBlocks[emptyBlock].blockSize - processSizes[pindx].first;
        }
    }
    int externalFrag = 0, internalFrag = 0;
    if (leftProcess.empty() == false)
    {
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true)
                continue;
            externalFrag += memBlocks[bindx].blockSize;
        }
    }
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        internalFrag += memBlocks[bindx].internalFrag;
    }
    Display(memBlocks, noOfBlocks, internalFrag, externalFrag,
leftProcess);
}

int main()
{
    system("cls");

    printLines();
    cout << "Vicky Gupta 20BCS070\n";
    cout << "Best Fit Memory Allocation Algorithm\n";
    printLines();
    printLines();

    int noOfBlocks;
    cout << "Enter The No Of Blocks Of Memory : ";
    cin >> noOfBlocks;

    printLines();
    int noOfProcess;
    cout << "Enter The No Of Process : ";
    cin >> noOfProcess;
    printLines();
    vector<memoryBlocks> memBlocks(noOfBlocks);
    cout << "Enter The Sizes Of Blocks : ";
    for (int i = 0; i < noOfBlocks; i++)
    {
        cin >> memBlocks[i].blockSize;
        memBlocks[i].isAllocated = false;
        memBlocks[i].processSize = 0;
        memBlocks[i].processName = "";
        memBlocks[i].internalFrag = 0;
    }

    printLines();
    vector<pair<int, string>> processSizes(noOfProcess);
    cout << "Enter The Sizes Of Process : ";
    for (int i = 0; i < noOfProcess; i++)
    {
        cin >> processSizes[i].first;
        processSizes[i].second = "P";
        processSizes[i].second += to_string(i + 1);
    }
    printLines();

    cout << "\n\n";
    printLines();
    printLines();
    Best_Fit(memBlocks, noOfBlocks, processSizes, noOfProcess);
    return 0;
}
```

Output :-



```
-------------------------------------------------------------------------------------
Vicky Gupta 20BCS070
Best Fit Memory Allocation Algorithm

-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
Enter The No Of Blocks Of Memory : 5

-------------------------------------------------------------------------------------
Enter The No Of Process : 4

-------------------------------------------------------------------------------------
Enter The Sizes Of Blocks : 100 500 200 300 600

-------------------------------------------------------------------------------------
Enter The Sizes Of Process : 212 417 112 426

-------------------------------------------------------------------------------------



-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
Best Fit Memory Allocation Table
---------------------------------------------------------------------------------
| Block No      Size Of Block   Proces Allocated        Internal Fragmentation  |
---------------------------------------------------------------------------------

|  1                   100              ---                     --             |
|  2                   500            417[P2]                   83             |
|  3                   200            112[P3]                   88             |
|  4                   300            212[P1]                   88             |
|  5                   600            426[P4]                   174            |
---------------------------------------------------------------------------------


-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------



-------------------------------------------------------------------------------------
Total Internal Fragmentation = 433
Total External Fragmentation = 0

-------------------------------------------------------------------------------------
```

Operating System Lab

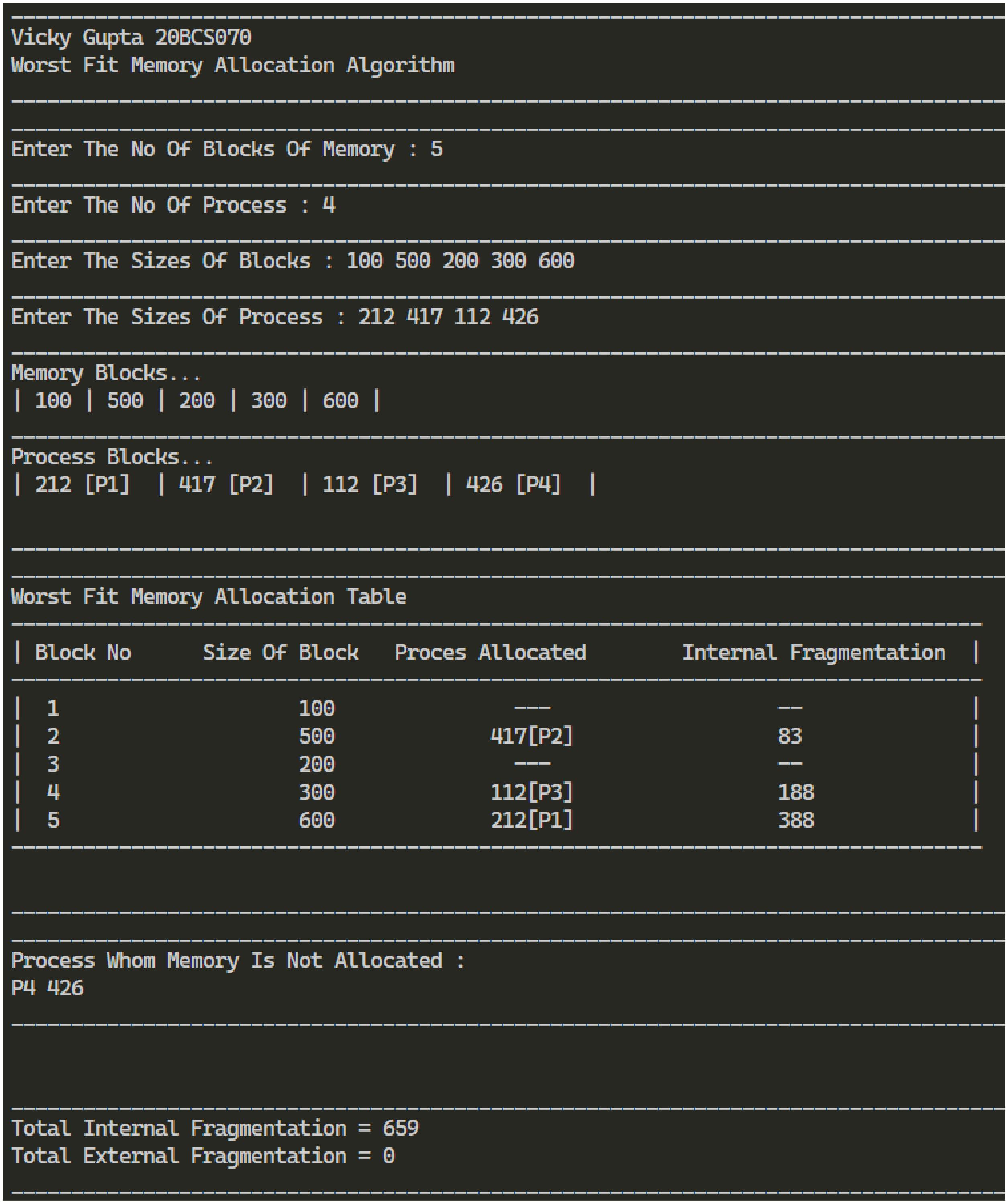
CEN-493

Program - 12

Code :-

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

typedef long long ll;

struct memoryBlocks
{
    bool isAllocated;
    int blockSize;
    int processSize;
    int internalFrag;
    string processName;
};

void printLines()
{
    for (int i = 0; i < 110; i++)
    {
        cout << "_";
    }
    cout << "\n";
}

void Display(vector<memoryBlocks> &memBlocks, int noOfBlocks, int
internalFrag, int externalFrag, vector<pair<int, string>>
&leftProcess)
{
    cout << "Worst Fit Memory Allocation Table \n";
    cout << "-----------------------------------------------------
----------------------------\n";
    cout << "| Block No\t"
         << "Size Of Block\t"
         << "Proces Allocated\t"
         << "Internal Fragmentation  |\n";
    cout << "-----------------------------------------------------
----------------------------\n";
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        if (memBlocks[bindx].isAllocated == false)
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << "  ---  "
                 << "\t\t\t"
                 << "--"
                 << "\t\t|\n";

        else
            cout << "|  " << bindx + 1 << "\t\t\t" <<
memBlocks[bindx].blockSize << "\t\t"
                 << memBlocks[bindx].processSize << "[" <<
memBlocks[bindx].processName << "]"
                 << "\t\t\t" << memBlocks[bindx].internalFrag <<
"\t\t|\n";
    }
    cout << "-----------------------------------------------------
----------------------------\n";

    cout << "\n";
    printLines();
    printLines();
    if (!leftProcess.empty())
    {
        cout << "Process Whom Memory Is Not Allocated : \n";
        for (int lindx = 0; lindx < leftProcess.size(); lindx++)
        {
            cout << leftProcess[lindx].second << " " <<
leftProcess[lindx].first << "\n";
        }
    }

    printLines();
    cout << "\n\n";
    printLines();
    cout << "Total Internal Fragmentation = " << internalFrag <<
"\n";
    cout << "Total External Fragmentation = " << externalFrag <<
"\n";
    printLines();
}

void Worst_Fit(vector<memoryBlocks> &memBlocks, int noOfBlocks,
vector<pair<int, string>> &processSizes, int noOfProcess)
{
    vector<pair<int, string>> leftProcess;
    for (int pindx = 0; pindx < noOfProcess; pindx++)
    {
        bool isProcessMemAllocated = false;
        int emptyBlock = 0, largestBlockSize = 0;
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true ||
memBlocks[bindx].blockSize < processSizes[pindx].first)
                continue;

            isProcessMemAllocated = true;
            if (largestBlockSize < memBlocks[bindx].blockSize)
            {
                emptyBlock = bindx;
                largestBlockSize = memBlocks[bindx].blockSize;
            }
        }
        if (isProcessMemAllocated == false)
        {
            leftProcess.push_back(processSizes[pindx]);
        }
        else
        {
            memBlocks[emptyBlock].isAllocated = true;
            memBlocks[emptyBlock].processName =
processSizes[pindx].second;
            memBlocks[emptyBlock].processSize =
processSizes[pindx].first;
            memBlocks[emptyBlock].internalFrag =
memBlocks[emptyBlock].blockSize - processSizes[pindx].first;
        }
    }
    int externalFrag = 0, internalFrag = 0;
    if (leftProcess.empty() == false)
    {
        for (int bindx = 0; bindx < noOfBlocks; bindx++)
        {
            if (memBlocks[bindx].isAllocated == true)
                continue;
            externalFrag += memBlocks[bindx].blockSize;
        }
        int leftProcessSize = 0;
        bool isExternFrag = 0;
        for (int iter = 0; iter < leftProcess.size(); iter++)
        {
            if (leftProcess[iter].first < externalFrag)
            {
                isExternFrag = 1;
                break;
            }
        }
        if (isExternFrag == 0)
        {
            externalFrag = 0;
        }
    }
    for (int bindx = 0; bindx < noOfBlocks; bindx++)
    {
        internalFrag += memBlocks[bindx].internalFrag;
    }
    Display(memBlocks, noOfBlocks, internalFrag, externalFrag,
leftProcess);
}

int main()
{
    system("cls");

    printLines();
    cout << "Vicky Gupta 20BCS070\n";
    cout << "Worst Fit Memory Allocation Algorithm\n";
    printLines();
    printLines();

    int noOfBlocks;
    cout << "Enter The No Of Blocks Of Memory : ";
    cin >> noOfBlocks;

    printLines();
    int noOfProcess;
    cout << "Enter The No Of Process : ";
    cin >> noOfProcess;

    printLines();
    vector<memoryBlocks> memBlocks(noOfBlocks);
    cout << "Enter The Sizes Of Blocks : ";
    for (int i = 0; i < noOfBlocks; i++)
    {
        cin >> memBlocks[i].blockSize;
        memBlocks[i].isAllocated = false;
        memBlocks[i].processSize = 0;
        memBlocks[i].processName = "";
        memBlocks[i].internalFrag = 0;
    }

    printLines();
    vector<pair<int, string>> processSizes(noOfProcess);
    cout << "Enter The Sizes Of Process : ";
    for (int i = 0; i < noOfProcess; i++)
    {
        cin >> processSizes[i].first;
        processSizes[i].second = "P";
        processSizes[i].second += to_string(i + 1);
    }
    printLines();
    cout << "Memory Blocks...\n";
    cout << "| ";
    for (int i = 0; i < noOfBlocks; i++)
    {
        cout << memBlocks[i].blockSize << " | ";
    }
    cout << "\n";
    printLines();
    cout << "Process Blocks...\n";
    cout << "| ";
    for (int i = 0; i < noOfProcess; i++)
    {
        cout << processSizes[i].first << " [" << 
processSizes[i].second << "]  | ";
    }
    cout << "\n\n";
    printLines();
    printLines();
    Worst_Fit(memBlocks, noOfBlocks, processSizes, noOfProcess);
    return 0;
}
```

Output :-



```
-----------------------------------------------------------------------------------
Vicky Gupta 20BCS070
Worst Fit Memory Allocation Algorithm

-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
Enter The No Of Blocks Of Memory : 5

-----------------------------------------------------------------------------------
Enter The No Of Process : 4

-----------------------------------------------------------------------------------
Enter The Sizes Of Blocks : 100 500 200 300 600

-----------------------------------------------------------------------------------
Enter The Sizes Of Process : 212 417 112 426

-----------------------------------------------------------------------------------
Memory Blocks...
| 100 | 500 | 200 | 300 | 600 |

-----------------------------------------------------------------------------------
Process Blocks...
| 212 [P1]  | 417 [P2]  | 112 [P3]  | 426 [P4]  |


-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
Worst Fit Memory Allocation Table
---------------------------------------------------------------------------------
| Block No      Size Of Block   Proces Allocated        Internal Fragmentation  |
---------------------------------------------------------------------------------
|  1                    100               ----                  --              |
|  2                    500             417[P2]                 83              |
|  3                    200               ----                  --              |
|  4                    300             112[P3]                 188             |
|  5                    600             212[P1]                 388             |
---------------------------------------------------------------------------------


-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
Process Whom Memory Is Not Allocated :
P4 426

-----------------------------------------------------------------------------------



-----------------------------------------------------------------------------------
Total Internal Fragmentation = 659
Total External Fragmentation = 0

-----------------------------------------------------------------------------------
```

Operating System Lab

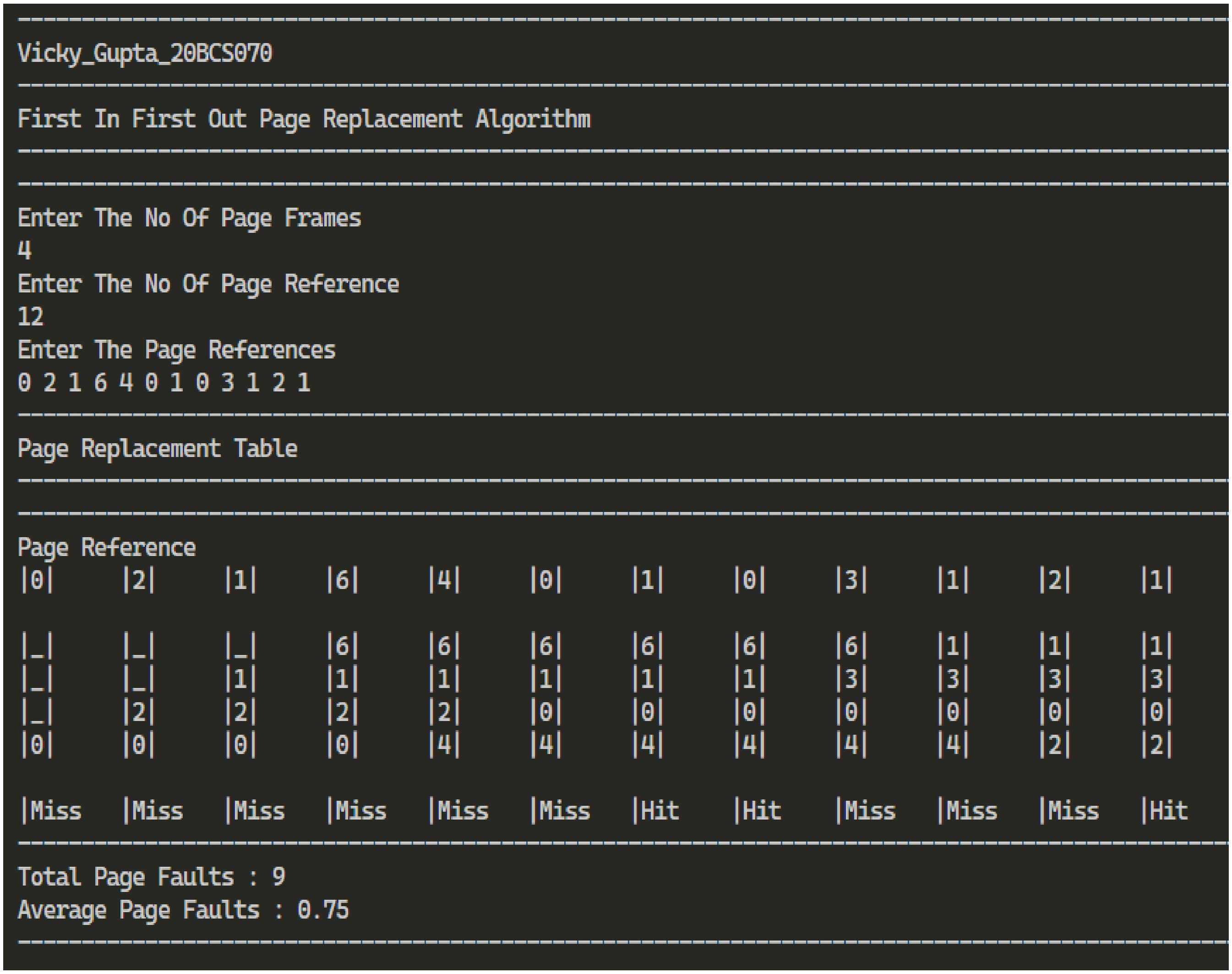
CEN-493

Program - 13

Code :-

```
#include <iostream>
#include <math.h>
#include <stack>
#include <algorithm>
using namespace std;

struct Fifo_State
{
    vector<int> state;
    bool isFault;
    int top;
};

void printLines()
{
    for (int i = 0; i < 120; i++)
    {
        cout << "-";
    }
    cout << "\n";
}

void Print(vector<Fifo_State> &allStates, int pageFaults)
{
    printLines();
    cout << "Page Replacement Table\n";
    printLines();
    printLines();
    cout << "Page Reference\n";
    for (int state = 0; state < allStates.size(); state++)
    {
        cout << "|" << allStates[state].top << "|\t";
    }
    cout << "\n\n";

    for (int state = allStates[0].state.size() - 1; state >= 0;
state--)
    {
        for (int i = 0; i < allStates.size(); i++)
        {
            if (allStates[i].state[state] == -1)
                cout << "|"
                     << "_"
                     << "|\t";
            else
                cout << "|" << allStates[i].state[state] << "|\t";
        }
        cout << "\n";
    }
    cout << "\n";
    for (int state = 0; state < allStates.size(); state++)
    {
        if (allStates[state].isFault)
        {
            cout << "|"
                 << "Miss"
                 << "\t";
        }
        else
        {
            cout << "|"
                 << "Hit"
                 << "\t";
        }
    }
    cout << "\n";
    printLines();
    cout << "Total Page Faults : " << pageFaults << "\n";
    double averagePageFaults = pageFaults /
(double)allStates.size();
    cout << "Average Page Faults : " << averagePageFaults << "\n";
    printLines();
}

void Page_Replacement_Fifo(int noOfPageFrames, vector<int>
&pageReferences)
{
    vector<Fifo_State> allStates;
    vector<int> frame(noOfPageFrames, -1);
    int firstIndex = 0, pageFaults = 0, prIndex, top = 0;

    for (prIndex = 0; top != noOfPageFrames; prIndex++)
    {
        bool isFind = 0;
        for (int fIndex = 0; fIndex < top; fIndex++)
        {
            if (frame[fIndex] == pageReferences[prIndex])
            {
                isFind = true;
                break;
            }
        }
        Fifo_State newState;
        if (isFind)
        {
            newState.isFault = 0;
        }
        else
        {
            frame[top] = pageReferences[prIndex];
            newState.isFault = 1;
            pageFaults++;
            top++;
        }
        newState.top = pageReferences[prIndex];
        newState.state = frame;
        allStates.push_back(newState);
    }
    for (prIndex; prIndex < pageReferences.size(); prIndex++)
    {
        bool isFind = 0;
        for (int fIndex = 0; fIndex < noOfPageFrames; fIndex++)
        {
            if (frame[fIndex] == pageReferences[prIndex])
            {
                isFind = true;
                break;
            }
        }
        if (isFind)
        {
            Fifo_State newState;
            newState.isFault = 0;
            newState.top = pageReferences[prIndex];
            newState.state = frame;
            allStates.push_back(newState);
        }
        else
        {
            Fifo_State newState;
            newState.isFault = 1;
            pageFaults++;
            newState.top = pageReferences[prIndex];
            for (int fIndex = 0; fIndex <= noOfPageFrames;
fIndex++)
            {
                if (frame[fIndex] == pageReferences[firstIndex])
                {
                    firstIndex++;
                    frame[fIndex] = pageReferences[prIndex];
                    break;
                }
            }
            newState.state = frame;
            allStates.push_back(newState);
        }
    }
    Print(allStates, pageFaults);
}

int main()
{
    system("cls");
    printLines();
    cout << "Vicky_Gupta_20BCS070\n";
    printLines();
    cout << "First In First Out Page Replacement Algorithm\n";
    printLines();
    printLines();
    int noOfPageFrames;

    cout << "Enter The No Of Page Frames \n";
    cin >> noOfPageFrames;

    int noOfPageReference;
    cout << "Enter The No Of Page Reference\n";
    cin >> noOfPageReference;

    vector<int> pageReferences(noOfPageReference);
    cout << "Enter The Page References\n";
    for (int i = 0; i < noOfPageReference; i++)
    {
        cin >> pageReferences[i];
    }

    Page_Replacement_Fifo(noOfPageFrames, pageReferences);
    return 0;
}
```

Output :-



```
-----------------------------------------------------------------------------------------------
Vicky_Gupta_20BCS070
-----------------------------------------------------------------------------------------------
First In First Out Page Replacement Algorithm
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
Enter The No Of Page Frames
4
Enter The No Of Page Reference
12
Enter The Page References
0 2 1 6 4 0 1 0 3 1 2 1
-----------------------------------------------------------------------------------------------
Page Replacement Table
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
Page Reference
|0|     |2|     |1|     |6|     |4|     |0|     |1|     |0|     |3|     |1|     |2|     |1|

|_|     |_|     |_|     |6|     |6|     |6|     |6|     |6|     |6|     |1|     |1|     |1|
|_|     |_|     |1|     |1|     |1|     |1|     |1|     |1|     |3|     |3|     |3|     |3|
|_|     |2|     |2|     |2|     |2|     |0|     |0|     |0|     |0|     |0|     |0|     |0|
|0|     |0|     |0|     |0|     |4|     |4|     |4|     |4|     |4|     |4|     |2|     |2|

|Miss   |Miss   |Miss   |Miss   |Miss   |Miss   |Hit    |Hit    |Miss   |Miss   |Miss   |Hit
-----------------------------------------------------------------------------------------------
Total Page Faults : 9
Average Page Faults : 0.75
-----------------------------------------------------------------------------------------------
```

Operating System Lab

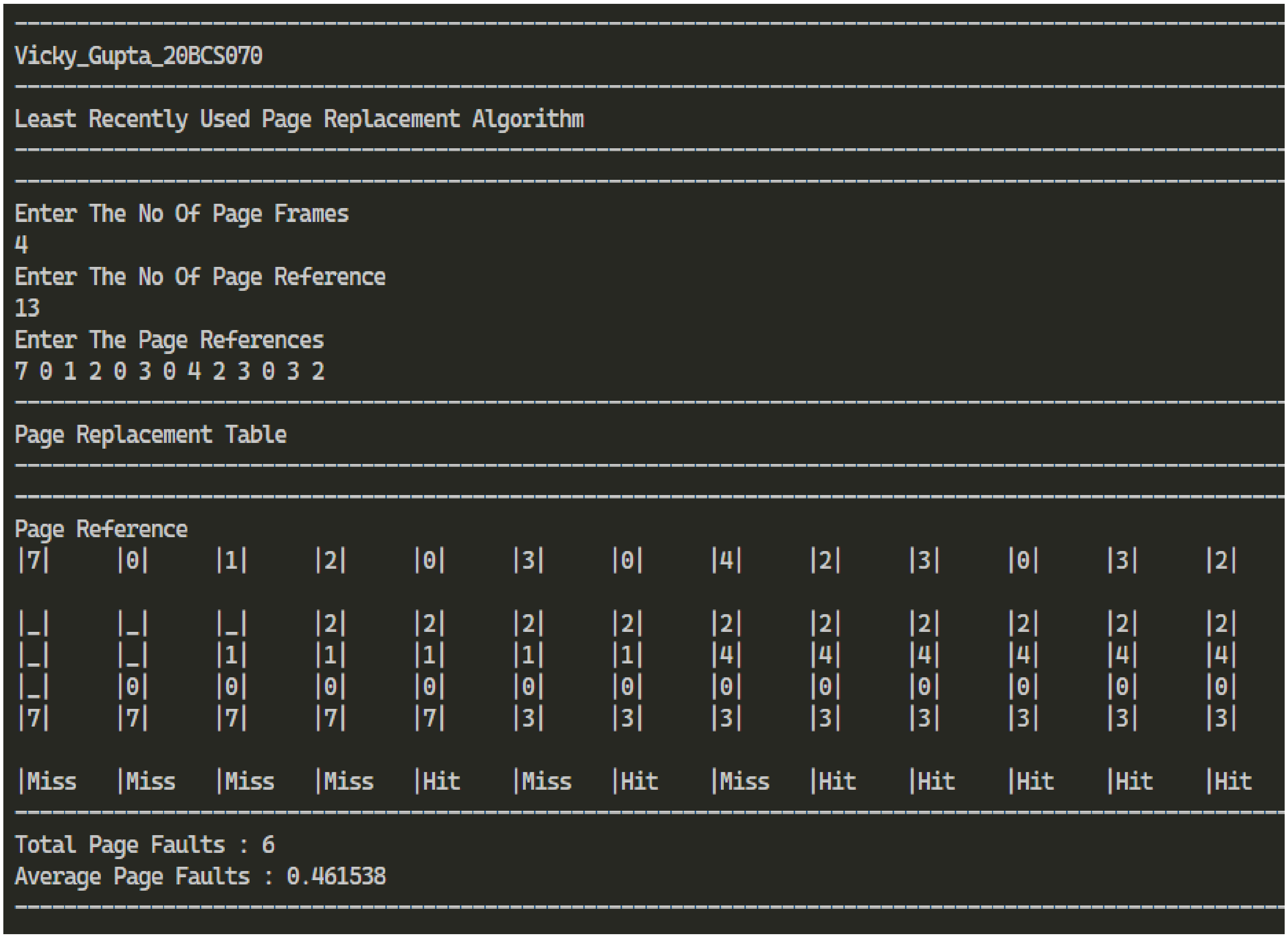
CEN-493

Program - 14

Code :-

```
#include <iostream>
#include <math.h>
#include <stack>
#include <algorithm>
using namespace std;

struct LRU_State
{
    vector<int> state;
    bool isFault;
    int top;
};

void printLines()
{
    for (int i = 0; i < 100; i++)
    {
        cout << "-";
    }
    cout << "\n";
}

void Print(vector<LRU_State> &allStates, int pageFaults)
{
    printLines();
    cout << "Page Replacement Table\n";
    printLines();
    printLines();
    cout << "Page Reference\n";
    for (int state = 0; state < allStates.size(); state++)
    {
        cout << "|" << allStates[state].top << "|\t";
    }
    cout << "\n\n";

    for (int state = allStates[0].state.size() - 1; state >= 0;
state--)
    {
        for (int i = 0; i < allStates.size(); i++)
        {
            if (allStates[i].state[state] == -1)
                cout << "|"
                     << "_"
                     << "|\t";
            else
                cout << "|" << allStates[i].state[state] << "|\t";
        }
        cout << "\n";
    }
    cout << "\n";
    for (int state = 0; state < allStates.size(); state++)
    {
        if (allStates[state].isFault)
        {
            cout << "|"
                 << "Miss"
                 << "\t";
        }
        else
        {
            cout << "|"
                 << "Hit"
                 << "\t";
        }
    }
    cout << "\n";
    printLines();
    cout << "Total Page Faults : " << pageFaults << "\n";
    double averagePageFaults = pageFaults /
(double)allStates.size();
    cout << "Average Page Faults : " << averagePageFaults << "\n";
    printLines();
}

void rotate(vector<int> &arr, int x, int y)
{
    int first = arr[x];
    for (int i = x; i < y; i++)
    {
        arr[i] = arr[i + 1];
    }
    arr[y] = first;
}

void Page_Replacement_LRU(int noOfPageFrames, vector<int>
&pageReferences)
{
    vector<LRU_State> allStates;
    vector<int> frame(noOfPageFrames, -1), lru(noOfPageFrames, -
1);
    int pageFaults = 0, top = 0, prIndex = 0;

    for (prIndex = 0; top != noOfPageFrames; prIndex++)
    {
        bool isFind = false;
        for (int fIndex = 0; fIndex < top; fIndex++)
        {
            if (frame[fIndex] == pageReferences[prIndex])
            {
                isFind = true;
                break;
            }
        }
        LRU_State newState;
        if (isFind)
        {
            for (int lruIndex = 0; lruIndex < top; lruIndex++)
            {
                if (lru[lruIndex] == pageReferences[prIndex])
                {
                    rotate(lru, lruIndex, top - 1);
                    break;
                }
            }
            newState.isFault = 0;
        }
        else
        {
            frame[top] = pageReferences[prIndex];
            lru[top] = pageReferences[prIndex];
            newState.isFault = 1;
            pageFaults++;
            top++;
        }
        newState.top = pageReferences[prIndex];
        newState.state = frame;
        allStates.push_back(newState);
    }

    for (prIndex; prIndex < pageReferences.size(); prIndex++)
    {
        bool isFind = 0;
        for (int fIndex = 0; fIndex < noOfPageFrames; fIndex++)
        {
            if (frame[fIndex] == pageReferences[prIndex])
            {
                isFind = true;
                break;
            }
        }
        if (isFind)
        {
            for (int lruIndex = 0; lruIndex < noOfPageFrames;
lruIndex++)
            {
                if (lru[lruIndex] == pageReferences[prIndex])
                {
                    rotate(lru, lruIndex, noOfPageFrames - 1);
                    break;
                }
            }
            LRU_State newState;
            newState.isFault = 0;
            newState.top = pageReferences[prIndex];
            newState.state = frame;
            allStates.push_back(newState);
        }
        else
        {
            LRU_State newState;
            newState.isFault = 1;
            pageFaults++;
            newState.top = pageReferences[prIndex];
            int leastUsed = lru[0];
            for (int fIndex = 0; fIndex <= noOfPageFrames;
fIndex++)
            {
                if (frame[fIndex] == leastUsed)
                {
                    frame[fIndex] = pageReferences[prIndex];
                    lru[0] = pageReferences[prIndex];
                    break;
                }
            }
            rotate(lru, 0, noOfPageFrames - 1);
            newState.state = frame;
            allStates.push_back(newState);
        }
    }
    Print(allStates, pageFaults);
}

int main()
{
    system("cls");
    printLines();
    cout << "Vicky_Gupta_20BCS070\n";
    printLines();
    cout << "Least Recently Used Page Replacement Algorithm\n";
    printLines();
    printLines();
    int noOfPageFrames;

    cout << "Enter The No Of Page Frames \n";
    cin >> noOfPageFrames;

    int noOfPageReference;
    cout << "Enter The No Of Page Reference\n";
    cin >> noOfPageReference;
    vector<int> pageReferences(noOfPageReference);
    cout << "Enter The Page References\n";
    for (int i = 0; i < noOfPageReference; i++)
    {
        cin >> pageReferences[i];
    }
    Page_Replacement_LRU(noOfPageFrames, pageReferences);
    return 0;
}
```

Output :-



```
------------------------------------------------------------------------------------------------------
Vicky_Gupta_20BCS070
------------------------------------------------------------------------------------------------------
Least Recently Used Page Replacement Algorithm
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
Enter The No Of Page Frames
4
Enter The No Of Page Reference
13
Enter The Page References
7 0 1 2 0 3 0 4 2 3 0 3 2
------------------------------------------------------------------------------------------------------
Page Replacement Table
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
Page Reference
|7|     |0|     |1|     |2|     |0|     |3|     |0|     |4|     |2|     |3|     |0|     |3|     |2|

|_|     |_|     |_|     |2|     |2|     |2|     |2|     |2|     |2|     |2|     |2|     |2|     |2|
|_|     |_|     |1|     |1|     |1|     |1|     |1|     |4|     |4|     |4|     |4|     |4|     |4|
|_|     |0|     |0|     |0|     |0|     |0|     |0|     |0|     |0|     |0|     |0|     |0|     |0|
|7|     |7|     |7|     |7|     |7|     |3|     |3|     |3|     |3|     |3|     |3|     |3|     |3|

|Miss   |Miss   |Miss   |Miss   |Hit    |Miss   |Hit    |Miss   |Hit    |Hit    |Hit    |Hit    |Hit
------------------------------------------------------------------------------------------------------
Total Page Faults : 6
Average Page Faults : 0.461538
------------------------------------------------------------------------------------------------------
```

Operating System Lab

CEN-493

Program - 15

Code :-

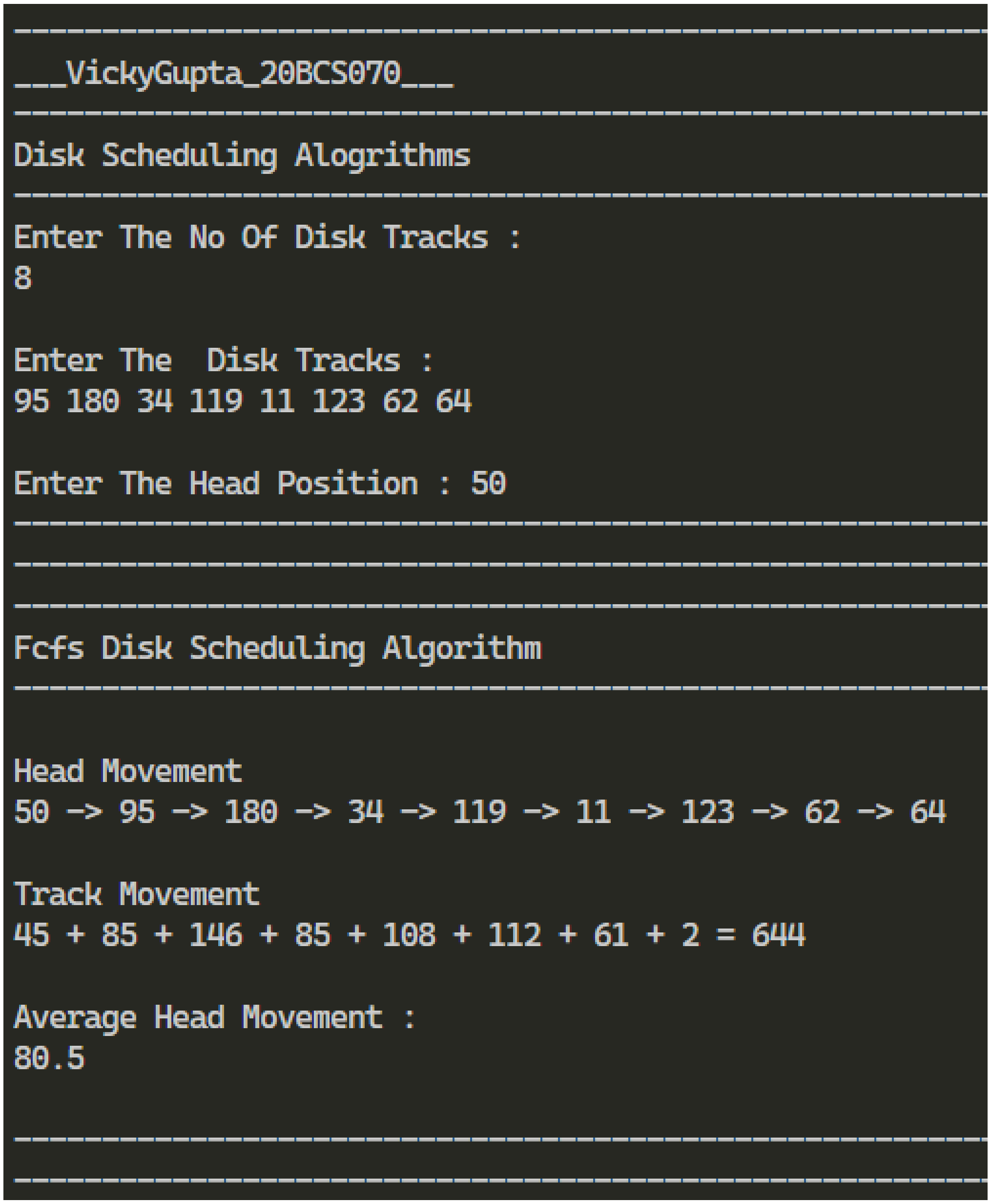
```
#include <iostream>
#include <math.h>
#include <vector>
#include <algorithm>
using namespace std;

void printLines()
{
    for (int i = 0; i < 120; i++)
    {
        cout << "-";
    }
    cout << "\n";
}
```

```
void printTheInfo(string info, int noOfDiskTracks,
vector<int> trackMovement, vector<int> headMovement)
{
    printLines();
    cout << info << "\n";
    printLines();
    int totalTrackMovement = 0;
    cout << "\nHead Movement\n";
    for (int i = 0; i < headMovement.size(); i++)
    {
        if (headMovement.size() - 1 == i)
            cout << headMovement[i] << " ";
        else
            cout << headMovement[i] << " -> ";
    }
    cout << "\n";

    cout << "\nTrack Movement\n";
    for (int i = 0; i < noOfDiskTracks; i++)
    {
        totalTrackMovement += trackMovement[i];
        if (i == noOfDiskTracks - 1)
            cout << trackMovement[i];
        else
            cout << trackMovement[i] << " + ";
    }
    cout << " = " << totalTrackMovement << "\n";
    float avgHeadMovement = (totalTrackMovement /
(float)noOfDiskTracks);
    cout << "\nAverage Head Movement : \n";
    cout << avgHeadMovement << "\n\n";
}

void fcfsDiskScheduling(int noOfDiskTracks, vector<int>
diskTracks, int headPosition)
{
    vector<int> headMovement, trackMovement;
    int prevHeadPosition = headPosition;
    headMovement.push_back(prevHeadPosition);
    for (int track = 0; track < noOfDiskTracks; track++)
    {
        headMovement.push_back(diskTracks[track]);
        trackMovement.push_back(abs(diskTracks[track] -
prevHeadPosition));
        prevHeadPosition = diskTracks[track];
    }

    printTheInfo("Fcfs Disk Scheduling Algorithm",
noOfDiskTracks, trackMovement, headMovement);
}

void sstfDiskScheduling(int noOfDiskTracks, vector<int>
diskTracks, int headPosition)
{
    vector<int> headMovement, trackMovement;
    int prevHeadPosition = headPosition;
    headMovement.push_back(prevHeadPosition);
    while (!diskTracks.empty())
    {
        int shortestSeekTime = 1e9, shortestSeekTimeIndex
= 0;
        for (int i = 0; i < diskTracks.size(); i++)
        {
            if (shortestSeekTime > abs(diskTracks[i] -
prevHeadPosition))
            {
                shortestSeekTime = abs(diskTracks[i] -
prevHeadPosition);
                shortestSeekTimeIndex = i;
            }
        }

        headMovement.push_back(diskTracks[shortestSeekTim
eIndex]);
        trackMovement.push_back(abs(diskTracks[shortestSe
ekTimeIndex] - prevHeadPosition));
        prevHeadPosition =
diskTracks[shortestSeekTimeIndex];
```

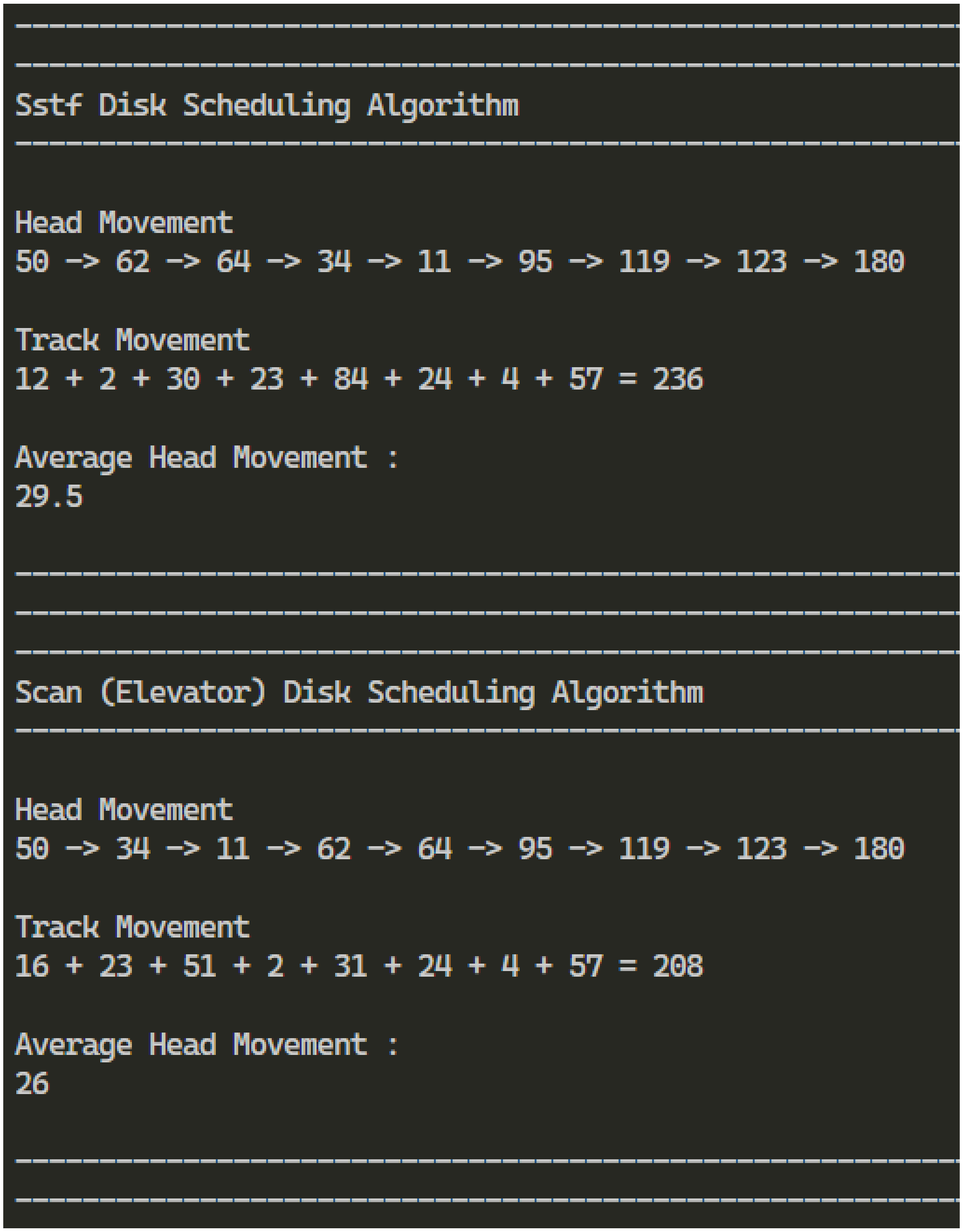
```
        diskTracks.erase(diskTracks.begin() + 
shortestSeekTimeIndex);
    }

    printTheInfo("Sstf Disk Scheduling Algorithm", 
noOfDiskTracks, trackMovement, headMovement);
}

void scanDiskScheduling(int noOfDiskTracks, vector<int> 
diskTracks, int headPosition)
{
    vector<int> headMovement, trackMovement;
    int prevHeadPosition = headPosition;
    headMovement.push_back(prevHeadPosition);
    sort(diskTracks.begin(), diskTracks.end());

    int strtTrack = lower_bound(diskTracks.begin(), 
diskTracks.end(), prevHeadPosition) - diskTracks.begin();
    if (diskTracks[strtTrack] > prevHeadPosition)
        strtTrack--;

    for (int track = strtTrack; track >= 0; track--)
    {
        headMovement.push_back(diskTracks[track]);
        trackMovement.push_back(abs(diskTracks[track] - 
prevHeadPosition));
        prevHeadPosition = diskTracks[track];
    }
    for (int track = strtTrack + 1; track < 
noOfDiskTracks; track++)
    {
        headMovement.push_back(diskTracks[track]);
        trackMovement.push_back(abs(diskTracks[track] - 
prevHeadPosition));
        prevHeadPosition = diskTracks[track];
    }
    printTheInfo("Scan (Elevator) Disk Scheduling 
Algorithm", noOfDiskTracks, trackMovement, headMovement);
}
```

```
int main()
{
    system("cls");

    printLines();
    cout << "___VickyGupta_20BCS070___\n";
    printLines();

    cout << "Disk Scheduling Alogrithms\n";
    printLines();

    int noOfDiskTracks;
    cout << "Enter The No Of Disk Tracks : \n";
    cin >> noOfDiskTracks;

    vector<int> diskTrack(noOfDiskTracks);
    cout << "\nEnter The  Disk Tracks :\n";
    for (int i = 0; i < noOfDiskTracks; i++)
    {
        cin >> diskTrack[i];
    }

    int headPosition;
    cout << "\nEnter The Head Position : ";
    cin >> headPosition;

    printLines();

    printLines();
    fcfsDiskScheduling(noOfDiskTracks, diskTrack,
headPosition);
    printLines();
    printLines();
    sstfDiskScheduling(noOfDiskTracks, diskTrack,
headPosition);
    printLines();
    printLines();
    scanDiskScheduling(noOfDiskTracks, diskTrack, 
headPosition);
    printLines();
    printLines();

    return 0;
}
```

Output :-



```
-------------------------------------------------------
___VickyGupta_20BCS070___
-------------------------------------------------------
Disk Scheduling Alogrithms
-------------------------------------------------------
Enter The No Of Disk Tracks :
8

Enter The  Disk Tracks :
95 180 34 119 11 123 62 64

Enter The Head Position : 50
-------------------------------------------------------
-------------------------------------------------------
-------------------------------------------------------
Fcfs Disk Scheduling Algorithm
-------------------------------------------------------


Head Movement
50 -> 95 -> 180 -> 34 -> 119 -> 11 -> 123 -> 62 -> 64

Track Movement
45 + 85 + 146 + 85 + 108 + 112 + 61 + 2 = 644

Average Head Movement :
80.5


-------------------------------------------------------
-------------------------------------------------------
```



```
--------------------------------------------------------
--------------------------------------------------------
Sstf Disk Scheduling Algorithm
--------------------------------------------------------


Head Movement
50 -> 62 -> 64 -> 34 -> 11 -> 95 -> 119 -> 123 -> 180

Track Movement
12 + 2 + 30 + 23 + 84 + 24 + 4 + 57 = 236

Average Head Movement :
29.5


--------------------------------------------------------
--------------------------------------------------------
--------------------------------------------------------
Scan (Elevator) Disk Scheduling Algorithm
--------------------------------------------------------


Head Movement
50 -> 34 -> 11 -> 62 -> 64 -> 95 -> 119 -> 123 -> 180

Track Movement
16 + 23 + 51 + 2 + 31 + 24 + 4 + 57 = 208

Average Head Movement :
26


--------------------------------------------------------
--------------------------------------------------------
```