Write a program to implement the First Come First Serve scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

```
//FCFS

#include<iostream>
using namespace std;

int n;
float avgCt, avgWt, avgTt;

struct Process{
    char Pname[5];

    int arvlTime;
    int brstTime
    int cmpTime;
    int wtngTime
    int tatTime;

    struct Process *next;
};

int isEmpty(Process *front){
    if(front==NULL || n==0){
        return 1;
    }
    return 0;
}

struct Process *insert(Process *front, int i){

    struct Process *p = (struct Process*)malloc(sizeof(struct Process));

    cout<<"Enter the name of the Process "<<i<<", its Burst and Arrival Time
:
";
    cin>>p->Pname>>p->brstTime>>p->arvlTime;
1.
       p->next = NULL;

       if(front==NULL){
           front = p;
       }

       else if (front->arvlTime > p->arvlTime){
           p->next = front;
           front = p;
       }

       else{

           struct Process *tmp = front;
           while (tmp->next != NULL && tmp->next->arvlTime < p->arvlTime){
               tmp = tmp->next;
           }
```

```
            p->next = tmp->next;
            tmp->next = p;
        }

        return front;
    }

    void calculate(Process *front){
        if(isEmpty(front)){
            cout<<"\nNo processes in the ready Queue";
            return;
        }
        front->wtngTime=0;
        front->cmpTime=front->brstTime;

        //calculating completion time
        int prv = front->cmpTime;
        struct Process *tmp = front->next;
        while(tmp!=NULL){
            tmp->cmpTime = prv + tmp->brstTime;
            prv = tmp->cmpTime;
            tmp=tmp->next;
        }

        //calculating waiting time
        prv = front->cmpTime;
        tmp = front->next;
        while(tmp!=NULL){
2.              tmp->wtngTime = prv - tmp->arvlTime;
            prv = tmp->cmpTime;
            tmp=tmp->next;
        }

        //calculating turn arround time
        tmp = front;
        while(tmp!=NULL){
            tmp->tatTime = tmp->wtngTime + tmp->brstTime;
            tmp=tmp->next;
        }

        //calculating average time
        tmp = front;
        float s1=0, s2=0, s3=0;
        while(tmp!=NULL){
            s1 = s1 + tmp->cmpTime;
            s2 = s2 + tmp->wtngTime;
            s3 = s3 + tmp->tatTime;
            tmp=tmp->next;
        }

        avgCt = s1/n;
        avgWt = s2/n;
        avgTt = s3/n;
    }

    void display(Process *front){
        if(isEmpty(front)){
            cout<<"\nNo processes in the ready Queue";
            return;
        }
```

```
    cout<<"\n\nDisplaying the table :- ";

    struct Process *tmp = front;

    cout<<"\n\n+--------------+-----------+--------------+---------------
--+--
------------+-----------------+---------------+";
    cout<<"\n|  Process name |  Burst Time |  Arrival Time |  Completion Time
|
Waiting Time |  TurnAround Time |  Response Time |";
    cout<<"\n+--------------+-----------+--------------+-----------------
+----
----------+-----------------+---------------+";

    while(tmp!=NULL){
        printf("\n|       %s       |     %2d      |       %2d      |
%2d
  |       %2d      |       %2d          |       %2d       |"
3.                      ,tmp->Pname, tmp->brstTime, tmp->arvlTime, tmp-
>cmpTime, tmp-
>wtngTime, tmp->tatTime, tmp->wtngTime);
    cout<<"\n+--------------+-----------+--------------+-----------------
+----
----------+-----------------+---------------+";
        tmp=tmp->next;
    }

    cout<<"\n\n";
    printf("\nAverage Completion time : %.2fns", avgCt);
    printf("\nAverage Waiting time : %.2fns", avgWt);
    printf("\nAverage TurnAround time : %.2fns", avgTt);
    printf("\nAverage Response time : %.2fns", avgWt);
}

void printGanttChart(Process *front){
    if(isEmpty(front)){
        cout<<"\nNo processes in the ready Queue";
        return;
    }

    cout<<"\n\nGantt Chart : ";

    struct Process *tmp = front;

    cout<<"\n\n+";
    while(tmp!=NULL){
        for(int i=0; i<2*tmp->brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
        tmp = tmp->next;
    }

    tmp = front;
    cout<<"\n|";
    while(tmp!=NULL){
        for(int i=0; i<tmp->brstTime-1; i++){
            cout<<" ";
        }
        cout<<tmp->Pname;
        for(int i=0; i<tmp->brstTime-1; i++){
            cout<<" ";
        }
```

```
        cout<<"| ";
        tmp = tmp->next;
    }
```

4.

```
    tmp = front;
    cout<<"\n+";
    while(tmp!=NULL){
        for(int i=0; i<2*tmp->brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
        tmp = tmp->next;
    }

    tmp = front;
    cout<<"\n0";
    while(tmp!=NULL){
        for(int i=0; i<2*tmp->brstTime-1; i++){
            cout<<" ";
        }
        // cout<<tmp->cmpTime;
        printf("%2d", tmp->cmpTime);
        tmp = tmp->next;
    }
    cout<<"\n\n";
}

int main(){
    cout<<"\nName : Mohd Adil";
    cout<<"\nRoll No : 20BCS042";

    cout<<"\nEnter the number of process";
    cin>>n;

    struct Process *front = NULL;

    for(int i=1; i<=n; i++){
        front = insert(front,i);
    }
```

```
        calculate(front);
        display(front);
        printGanttChart(front);
    return 0;
    }
```

OUTPUT:

```
Roll No : 20BCS042
Enter the number of process 5
Enter the name of the Process 1, its Burst and Arrival Time : P1 6 2
Enter the name of the Process 2, its Burst and Arrival Time : P2 2 5
Enter the name of the Process 3, its Burst and Arrival Time : P3 8 1
Enter the name of the Process 4, its Burst and Arrival Time : P4 3 0
Enter the name of the Process 5, its Burst and Arrival Time : P5 4 4

Displaying the table :-
```

```
Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      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       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+


Average Completion time : 15.00ns
Average Waiting time : 8.00ns
Average TurnAround time : 12.60ns
Average Response time : 8.00ns

Gantt Chart :

+------+----------------+------------+--------+----+
|  P4  |       P3       |     P1     |   P5   | P2 |
+------+----------------+------------+--------+----+
0      3                11           17       21   23
```

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.

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

struct Process
{
    char Pname[3];
    int memory;
    bool allocated = false;
};
struct Block
{
    int size;
    bool used = false;
    int rem;
    struct Process processAllocated;
};

int main()
{
    cout << "No. of block : ";
    int n;
    cin >> n;
    vector<Block> blocks;
    cout << "Enter Size of the " << n << " Blocks: ";
    for (int i = 0; i < n; i++)
    {
        Block tempBlock;
        cin >> tempBlock.size;
        tempBlock.rem = tempBlock.size;
        blocks.push_back(tempBlock);
```

```
    }
    cout << "No. of Process : ";
    int m;
    cin >> m;
    vector<Process> Processes;
    cout << "Enter Name and size of the Processes: ";
    for (int i = 0; i < m; i++)
    {
        Process tempProcess;
        cin >> tempProcess.Pname;
        cin >> tempProcess.memory;
        Processes.push_back(tempProcess);
    }
    // memory allocation
    for (int i = 0; i < m; i++)
    {
        bool exist = false;
        int index, min = INT16_MAX;
        for (int j = 0; j < n; j++)
        {
            if (Processes[i].memory <= blocks[j].rem &&
blocks[j].used == false && blocks[j].rem < min)
            {
                min = blocks[j].rem;
                exist = true;
                index = j;
            }
        }
        if (exist)
        {
            Processes[i].allocated = true;
            blocks[index].used = true;
            blocks[index].rem = blocks[index].size -
Processes[i].memory;
            blocks[index].processAllocated = Processes[i];
        }
```

```
    }
    cout << "\tBlock Number\tSize\tProcess Allocated\tInternal
Fragmentation" << endl;
    for (int i = 0; i < n; i++)
    {
        if (blocks[i].used == true)
        {
            cout << "\t\t" << i + 1 << "\t" << blocks[i].size <<
"\t\t" << blocks[i].processAllocated.Pname << "\t\t\t" <<
blocks[i].rem << endl;
        }
        else
        {
            cout << "\t\t" << i + 1 << "\t" << blocks[i].size <<
"\t\t"
                 << "---"
                 << "\t\t\t"
                 << "---" << endl;
        }
    }
    bool flag = true;
    int remProcessesSize = 0;
    for (int i = 0; i < m; i++)
    {
        if (Processes[i].allocated == false)
        {
            flag = false;
            remProcessesSize += Processes[i].memory;
        }
    }

    int IF = 0, EF = 0;
    for (int i = 0; i < n; i++)
    {
        if (blocks[i].used == true)
        {
```

```
                IF += blocks[i].rem;
            }
            else
            {
                if (flag == false)
                {
                    EF += blocks[i].rem;
                }
            }
        }
        if (EF <= remProcessesSize)
        {
            EF = 0;
        }
        cout << "Total Internal Fragmentation = " << IF << endl;
        cout << "Total External Fragmentation = " << EF << endl;
        return 0;
    }
```

OUTPUT:

```
No. of block : 5
Enter Size of the 5 Blocks: 200 100 300 400 500
No. of Process : 4
Enter Name and size of the Processes: p1 150 p2 300 p3 450 p4 50
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             p1                      50
                2       100             p4                      50
                3       300             p2                      0
                4       400             ---                     ---
                5       500             p3                      50
Total Internal Fragmentation = 150
Total External Fragmentation = 0
PS C:\Users\aadil\Desktop\CSE\OS Lab>
```