NAME: MOHD ADIL

ROLL NO.: 20BCS042

BRANCH COMPUTER ENGINEERING

SUBJECT: OPERATING SYSTEM LAB (CEN-493)

SUBJECT TEACHERS:

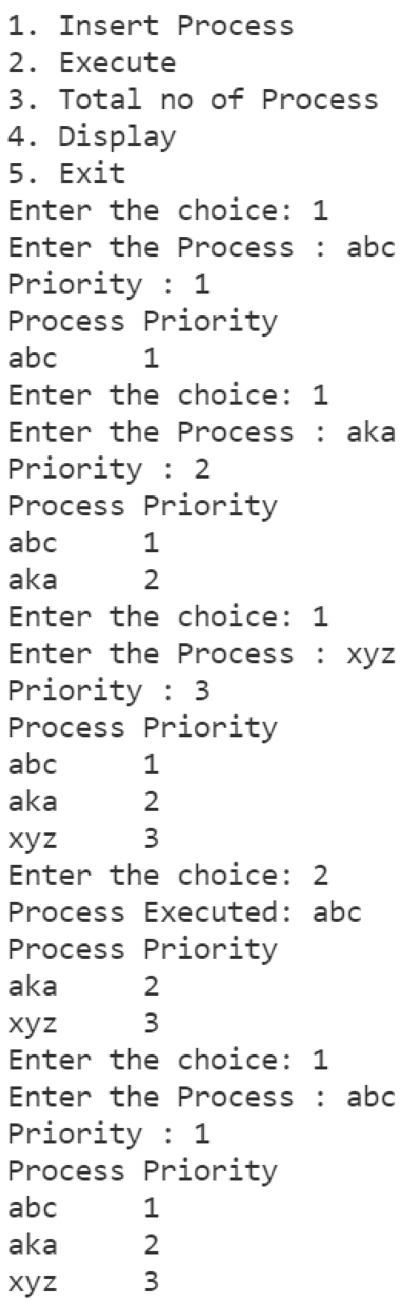
DR. SHAHZAD ALAM MR. JAWAHAR LAL

INDEX

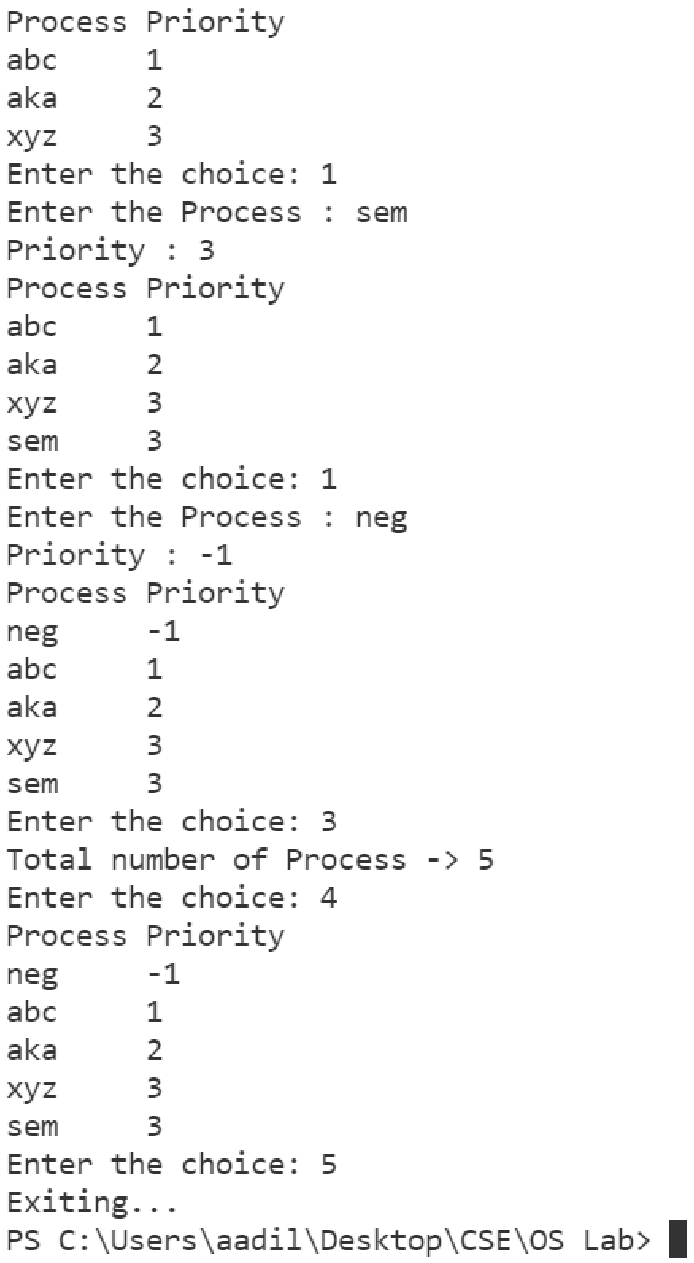
9	Write a program to implement the First fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	*24-Mar-2022*
10	Write a program 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*

```
#include <stdio.h>
#include <stdlib.h>

struct PQueue
{
    char n[4];
    int pr;
    struct PQueue *next;
} *front = NULL;

int count = 0;

void Insert()
{
    struct PQueue *temp = malloc(sizeof(struct PQueue));
    if (temp == NULL)
        printf("Heap Overflow\n");
    else
    {
        printf("Enter the Process : ");
        scanf("%s", temp->n);
        printf("Priority : ");
        scanf("%d", &temp->pr);
        temp->next = NULL;
        if (front == NULL || temp->pr < front->pr)
        {
            temp->next = front;
            front = temp;
        }
        else
        {
            struct PQueue *p = front;
            while (p->next != NULL && p->next->pr <= temp->pr)
                p = p->next;
            temp->next = p->next;
            p->next = temp;
        }
        count++;
    }
}
void Execute()
{
    if (front == NULL)
        printf("Queue Underflow\n");
    else
    {
        struct PQueue *temp = front;
        front = front->next;
        printf("Process Executed: %s\n", temp->n);
        free(temp);
        count--;
    }
}
void Display()
{
    if (front == NULL)
        printf("Queue is Empty\n");
    else
    {
        struct PQueue *temp = front;
        printf("Process\tPriority\n");
        while (temp != NULL)
        {
            printf("%s\t%d\n", temp->n, temp->pr);
            temp = temp->next;
        }
    }
}

int main()
{
    int choice;
    printf("\n1. Insert Process\n2. Execute\n3. Total no of Process\n4. 
Display\n5. Exit\n");
    while (1)
    {
        printf("Enter the choice: ");
        scanf("%d", &choice);
        getchar();
        switch (choice)
        {
        case 1:
            Insert();
            Display();
            break;
        case 2:
            Execute();
            Display();
            break;
        case 3:
            printf("Total number of Process -> %d\n", count);
            break;
        case 4:
            Display();
            break;
        case 5:
            printf("Exiting...");
            exit(0);
            break;
        }
    }
    return 0;
}
```

OUTPUT:



```
1. Insert Process
2. Execute
3. Total no of Process
4. Display
5. Exit
Enter the choice: 1
Enter the Process : abc
Priority : 1
Process Priority
abc     1
Enter the choice: 1
Enter the Process : aka
Priority : 2
Process Priority
abc     1
aka     2
Enter the choice: 1
Enter the Process : xyz
Priority : 3
Process Priority
abc     1
aka     2
xyz     3
Enter the choice: 2
Process Executed: abc
Process Priority
aka     2
xyz     3
Enter the choice: 1
Enter the Process : abc
Priority : 1
Process Priority
abc     1
aka     2
xyz     3
```



```
Process Priority
abc     1
aka     2
xyz     3
Enter the choice: 1
Enter the Process : sem
Priority : 3
Process Priority
abc     1
aka     2
xyz     3
sem     3
Enter the choice: 1
Enter the Process : neg
Priority : -1
Process Priority
neg     -1
abc     1
aka     2
xyz     3
sem     3
Enter the choice: 3
Total number of Process -> 5
Enter the choice: 4
Process Priority
neg     -1
abc     1
aka     2
xyz     3
sem     3
Enter the choice: 5
Exiting...
PS C:\Users\aadil\Desktop\CSE\OS Lab> █
```

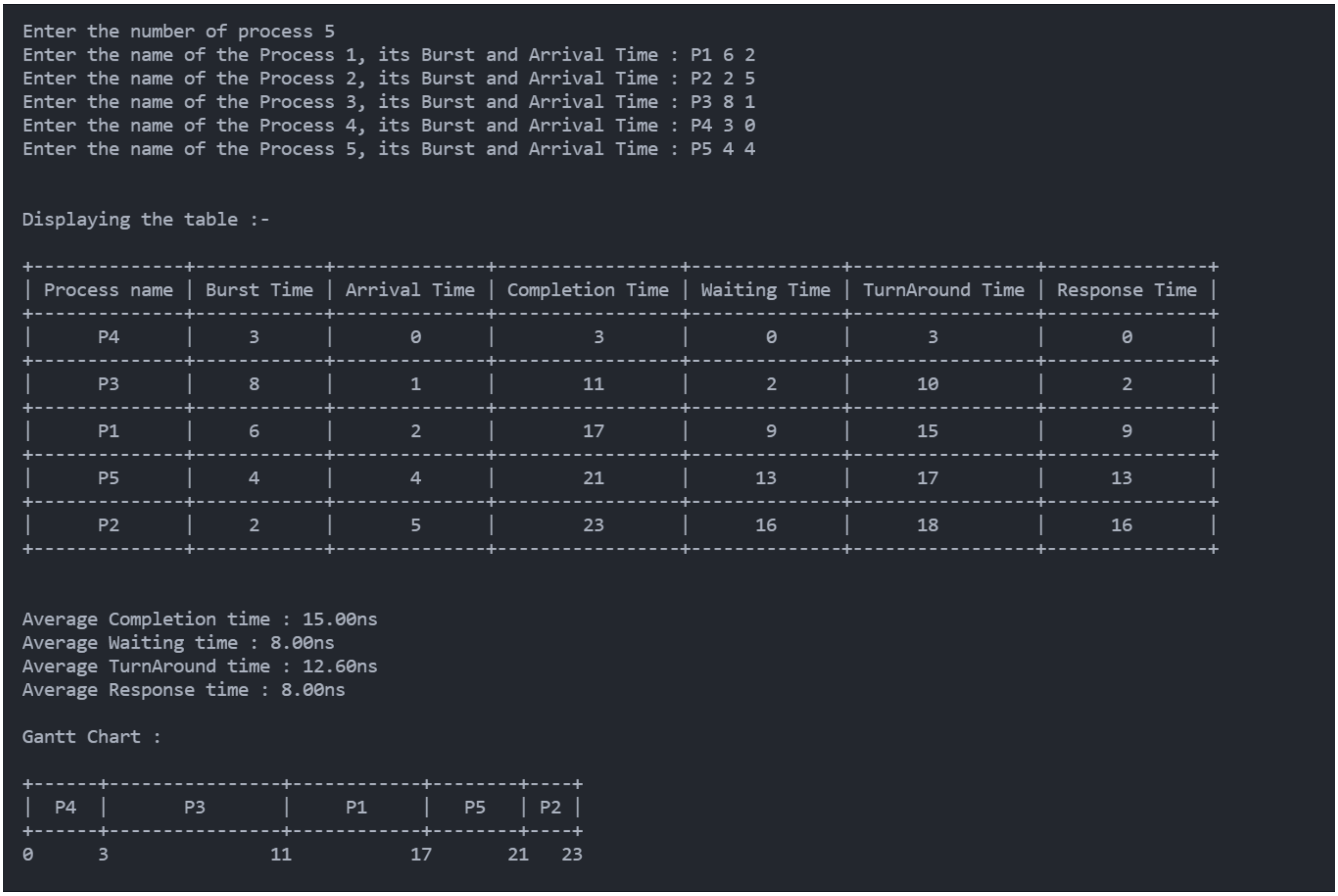
Name: Mohd Adil
Roll No: 20BCS042

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 Printing 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;
    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){
        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       |"
```

```cpp
               ,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;
    }
    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;
}

// 5 P1 6 2 P2 2 5 P3 8 1 P4 3 0 P5 4 4
```

Output:



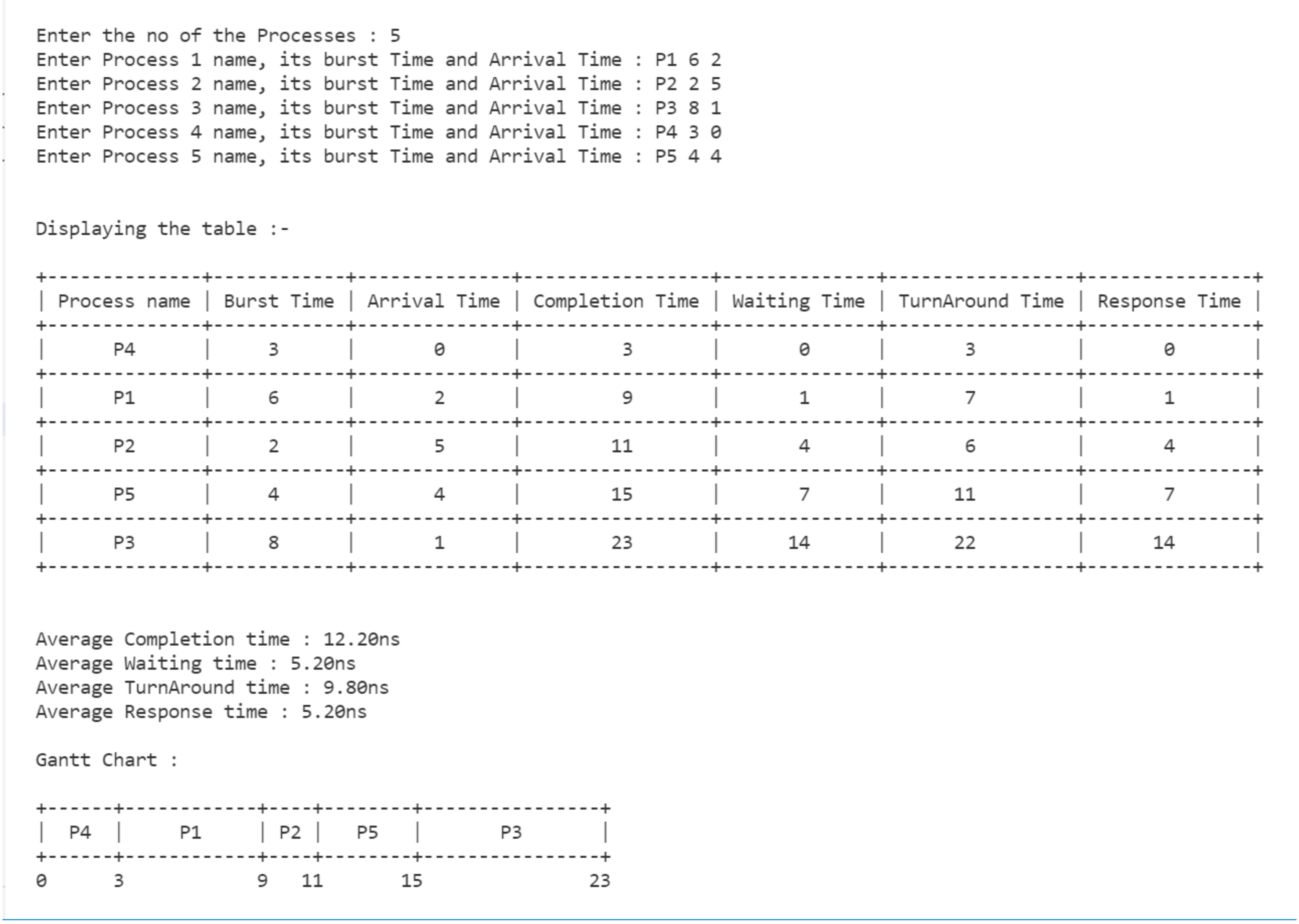
```
Enter the number of process 5
Enter the name of the Process 1, its Burst and Arrival Time : P1 6 2
Enter the name of the Process 2, its Burst and Arrival Time : P2 2 5
Enter the name of the Process 3, its Burst and Arrival Time : P3 8 1
Enter the name of the Process 4, its Burst and Arrival Time : P4 3 0
Enter the name of the Process 5, its Burst and Arrival Time : P5 4 4


Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| 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
```

PROGRAM:3 -> SHORTEST JOB FIRST NON-PREEMPTIVE SCHEDULING ALGORITHM

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

struct Process{
    char Pname[3];
    int arvlTime;
    int brstTime;
    int cmplTime;
    int wtngTime;
    int tartTime;
    int respTime;
};

struct priorityQ{
    Process pr;
    priorityQ *next;
};

priorityQ *push(priorityQ *front, Process pSample, char b){
    priorityQ *node = new priorityQ;
    node->pr = pSample;
    node->next=NULL;

//means push according to burst Time
    if(b=='b'){
        if(front==NULL){
            front=node;
        }
        else if(front->pr.brstTime > pSample.brstTime){
            node->next=front;
            front=node;
        }
        else{
            priorityQ *tmp=front;
            while (tmp->next!=NULL && tmp->next->pr.brstTime <
pSample.brstTime){
                tmp=tmp->next;
            }

            node->next=tmp->next;
            tmp->next=node;
        }
    }

//otherwise push accoring to arrival time
    else{
        if(front==NULL){
            front=node;
        }
        else if(front->pr.arvlTime > pSample.arvlTime){
            node->next=front;
            front=node;
        }
        else{
            priorityQ *tmp=front;
            while (tmp->next!=NULL && tmp->next->pr.arvlTime <
pSample.arvlTime){
                tmp=tmp->next;
            }

            node->next=tmp->next;
            tmp->next=node;
        }
    }

    return front;
}

priorityQ *pop(priorityQ *front){
    front=front->next;
    return front;
}

Process top(priorityQ *front){
    return front->pr;
}

bool empty(priorityQ *front){
    return (front==NULL);
```

```
}

//ans vector
vector<Process> v;
int n;
float avgc, avgw, avgt;

void SJF(priorityQ *pq1){
    int cmpt = top(pq1).brstTime;
    v.push_back(top(pq1));
    pq1 = pop(pq1);

    priorityQ *pq2=NULL;

    while(!empty(pq1)){
        while(!empty(pq1) && top(pq1).arvlTime < cmpt){
            pq2 = push(pq2, top(pq1), 'b');
            pq1 = pop(pq1);
        }

        cmpt += top(pq2).brstTime;
        v.push_back(top(pq2));
        pq2 = pop(pq2);
    }

    while(!empty(pq2)){
        v.push_back(top(pq2));
        pq2 = pop(pq2);
    }
}

void calculateTimes(){
    v.front().wtngTime=0;
    v.front().cmplTime = v.front().brstTime;
    float sumc=0, sumw=0, sumt=0;

    //calculating completion time
    int prv = v.front().cmplTime;
    sumc += prv;
    for(int i=1; i<n; i++){
        v[i].cmplTime = prv + v[i].brstTime;
        prv = v[i].cmplTime;
        sumc += v[i].cmplTime;
    }

    //calculating waiting time
    prv = v.front().cmplTime;
    for(int i=1; i<n; i++){
        v[i].wtngTime = prv - v[i].arvlTime;
        prv = v[i].cmplTime;
        sumw += v[i].wtngTime;
    }

    //calculating turn around time
    for(int i=0; i<n; i++){
        v[i].tartTime = v[i].brstTime + v[i].wtngTime;
        sumt += v[i].tartTime;
    }

    //calculating avg(s) time
    avgc = sumc/n;
    avgw = sumw/n;
    avgt = sumt/n;
}

void display(){
    cout<<"\n\nDisplaying the table :- ";

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

    for(auto i:v){
        printf("\n|      %s      |    %2d      |      %2d      |
 %2d       |      %2d      |      %2d         |      %2d       |"
               ,i.Pname, i.brstTime, i.arvlTime, i.cmplTime,
i.wtngTime, i.tartTime, i.wtngTime);
    cout<<"\n+--------------+------------+--------------+-------------
----+--------------+-----------------+---------------+";
    }
    cout<<"\n\n";
    printf("\nAverage Completion time : %.2fns", avgc);
    printf("\nAverage Waiting time : %.2fns", avgw);
    printf("\nAverage TurnAround time : %.2fns", avgt);
    printf("\nAverage Response time : %.2fns", avgw);
}

void printGantt(){

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

    cout<<"\n\n+";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
    }

    cout<<"\n|";
    for(auto p:v){
        for(int i=0; i<p.brstTime-1; i++){
            cout<<" ";
        }
        cout<<p.Pname;
        for(int i=0; i<p.brstTime-1; i++){
            cout<<" ";
        }
        cout<<"|";
    }

    cout<<"\n+";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
    }

    cout<<"\n0";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime-1; i++){
            cout<<" ";
        }
        printf("%2d", p.cmplTime);
    }
    cout<<"\n\n";
}

int main(){
    priorityQ *pq1=NULL;

    cout<<"Enter the no of the Processes : ";
    cin>>n;

    for(int i=0; i<n; i++){
        struct Process p;
        cout<<"Enter Process "<<i+1<<" name, its burst Time and
Arrival Time : ";
        cin>>p.Pname>>p.brstTime>>p.arvlTime;
        pq1 = push(pq1, p, 'a');
    }

    //sort according to arrival time + burst Time :
    SJF(pq1);
    calculateTimes();


    display();
    printGantt();
    return 0;
}
```

OUTPUT:



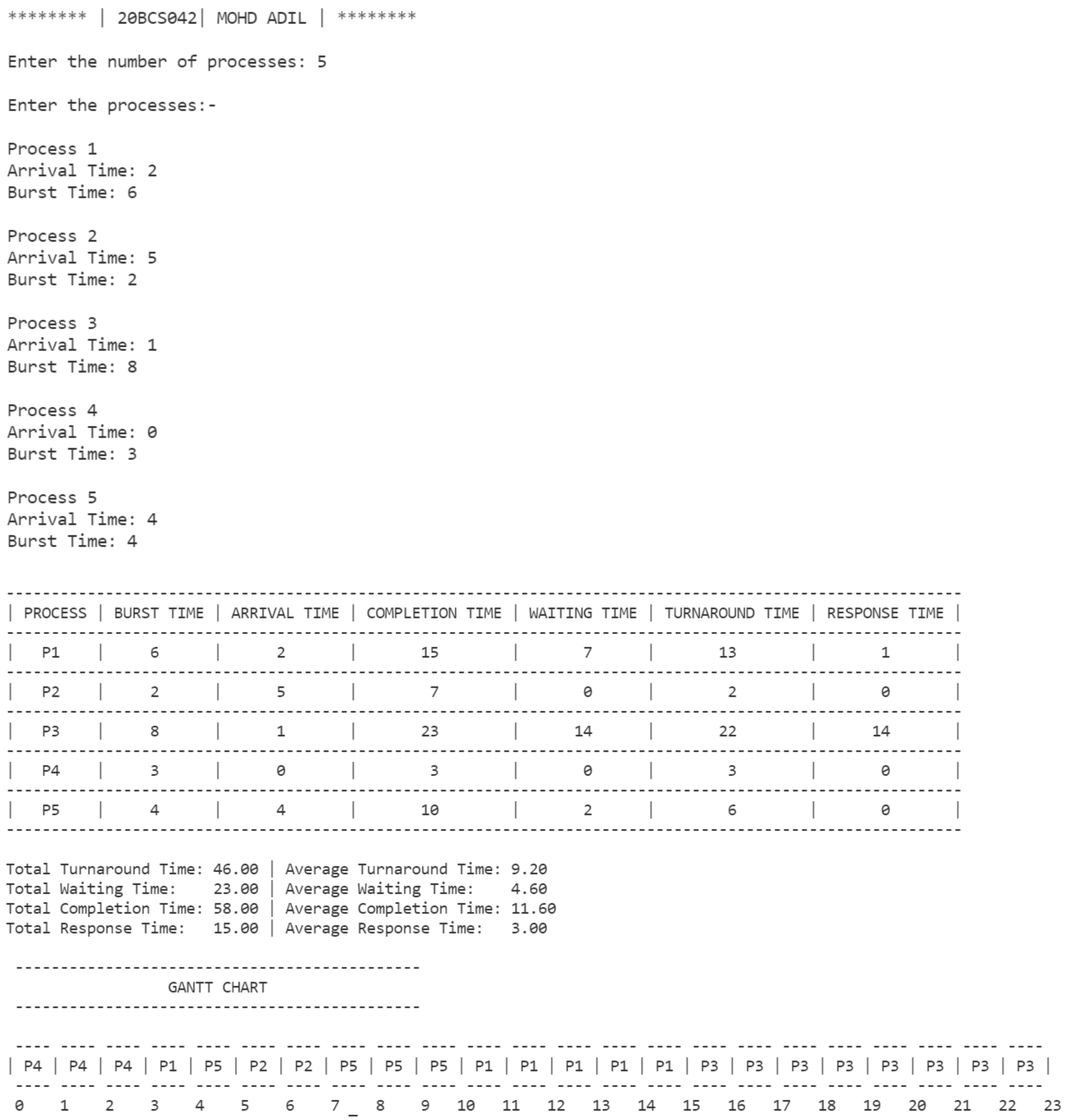
```
Enter the no of the Processes : 5
Enter Process 1 name, its burst Time and Arrival Time : P1 6 2
Enter Process 2 name, its burst Time and Arrival Time : P2 2 5
Enter Process 3 name, its burst Time and Arrival Time : P3 8 1
Enter Process 4 name, its burst Time and Arrival Time : P4 3 0
Enter Process 5 name, its burst Time and Arrival Time : P5 4 4


Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P4      |     3      |       0      |         3       |       0      |       3         |       0       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P1      |     6      |       2      |         9       |       1      |       7         |       1       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P2      |     2      |       5      |        11       |       4      |       6         |       4       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P5      |     4      |       4      |        15       |       7      |      11         |       7       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P3      |     8      |       1      |        23       |      14      |      22         |      14       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+


Average Completion time : 12.20ns
Average Waiting time : 5.20ns
Average TurnAround time : 9.80ns
Average Response time : 5.20ns

Gantt Chart :

+------+------------+----+--------+----------------+
|  P4  |     P1     | P2 |   P5   |       P3       |
+------+------------+----+--------+----------------+
0      3            9   11       15               23
```

20BCS042 | MOHD ADIL

PROGRAM 4: SRTF

```c
#include <stdio.h>

struct process
{
    int pid;
    int burst_time;
    int arrival_time;
    int waiting_time;
    int completion_time;
    int turnaround_time;
    int response_time;
    int start_time;
    int is_completed;
} pro[100];

int process_at_time[100];

void print_table(int num);
void timeCalculation(int burst_remaining[], int n);
void sortCompletion(int num);
void print_gantt(int n);

int main()
{
    printf("\n******** | 20BCS042| MOHD ADIL | ********\n");
    int n;
    int burst_remaining[100];

    printf("\nEnter the number of processes: ");
    scanf("%d", &n);

    printf("\nEnter the processes:-\n");
    for (int i = 0; i < n; i++)
    {
        printf("\nProcess %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &pro[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &pro[i].burst_time);
        pro[i].pid = i + 1;
        burst_remaining[i] = pro[i].burst_time;
    }

    timeCalculation(burst_remaining, n);
    sortCompletion(n);
    print_gantt(n);
}
void timeCalculation(int burst_remaining[], int n)
{
    float average_turnaround_time;
    float average_waiting_time;
    float average_completion_time;
    float average_response_time;

    float total_turnaround_time = 0;
    float total_waiting_time = 0;
    float total_completion_time = 0;
    float total_response_time = 0;
    float total_idle_time = 0;

    int current_time = 0;
    int completed_pro = 0;
    int prev = 0;

    while (completed_pro != n)
    {
        int shortest = -1;
        int min = 10000000;
        for (int i = 0; i < n; i++)
        {
            if (pro[i].arrival_time <= current_time &&
pro[i].is_completed == 0)
            {
                if (burst_remaining[i] < min)
                {
                    min = burst_remaining[i];
                    shortest = i;
                }
                else if (burst_remaining[i] == min)
                {
                    if (pro[i].arrival_time <
pro[shortest].arrival_time)
                    {
                        min = burst_remaining[i];
                        shortest = i;
                    }
                }
            }
        }

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

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

                total_turnaround_time +=
pro[shortest].turnaround_time;
                total_waiting_time += pro[shortest].waiting_time;
                total_response_time += pro[shortest].response_time;
                total_completion_time +=
pro[shortest].completion_time;

                pro[shortest].is_completed = 1;
                completed_pro++;
            }
            process_at_time[current_time - 1] = shortest + 1;
        }
        else
        {
            current_time++;
        }
    }

    average_waiting_time = total_waiting_time / n;
    average_response_time = total_response_time / n;
    average_turnaround_time = total_turnaround_time / n;
    average_completion_time = total_completion_time / n;

    print_table(n);

    printf("\nTotal Turnaround Time: %0.2f | Average Turnaround Time:
%0.2f", total_turnaround_time, average_turnaround_time);
    printf("\nTotal Waiting Time:    %0.2f | Average Waiting
Time:    %0.2f", total_waiting_time, average_waiting_time);
    printf("\nTotal Completion Time: %0.2f | Average Completion Time:
%0.2f", total_completion_time, average_completion_time);
    printf("\nTotal Response Time:   %0.2f | Average Response
Time:   %0.2f", total_response_time, average_response_time);
}
void sortCompletion(int num)
{
    struct process temp;
    for (int i = 0; i < num - 1; i++)
    {
        for (int j = 0; j < num - i - 1; j++)
        {
            if (pro[j].completion_time > pro[j + 1].completion_time)
            {
                temp = pro[j];
                pro[j] = pro[j + 1];
                pro[j + 1] = temp;
            }
        }
    }
}
void print_table(int num)
{
    printf("----------------------------------------------------------
------------------------------------------------\n");
    printf("| PROCESS | BURST TIME | ARRIVAL TIME | COMPLETION TIME |
WAITING TIME | TURNAROUND TIME | RESPONSE TIME |\n");
    printf("----------------------------------------------------------
------------------------------------------------\n");
    for (int i = 0; i < num; i++)
    {
        printf("|   P%d    |     %d      |      %d       |       %2d
     |      %2d      |       %2d        |      %2d       |\n",
pro[i].pid, pro[i].burst_time, pro[i].arrival_time,
pro[i].completion_time, pro[i].waiting_time, pro[i].turnaround_time,
pro[i].response_time);
        printf("------------------------------------------------------
----------------------------------------------------\n");
    }
}
void print_gantt(int n)
{
    printf("\n\n ---------------------------------------------\n");
    printf("                  GANTT CHART\n");
    printf(" ---------------------------------------------\n");
    printf("\n ");

    for (int i = 0; i < pro[n - 1].completion_time; i++)
    {
        printf("----");
        printf(" ");
    }
    printf("\n|");
    for (int i = 0; i < pro[n - 1].completion_time; i++)
    {
        printf(" P%d |", process_at_time[i]);
    }
    printf("\n ");
    for (int i = 0; i < pro[n - 1].completion_time; i++)
    {
        printf("----");
        printf(" ");
    }
    printf("\n");
    for (int i = 0; i <= pro[n - 1].completion_time; i++)
    {
        printf("%2d    ", i);
    }
}
//2 6 5 2 1 8 0 3 4 4
```

OUTPUT:



```
 ******** | 20BCS042| MOHD ADIL | ********

 Enter the number of processes: 5

 Enter the processes:-

 Process 1
 Arrival Time: 2
 Burst Time: 6

 Process 2
 Arrival Time: 5
 Burst Time: 2

 Process 3
 Arrival Time: 1
 Burst Time: 8

 Process 4
 Arrival Time: 0
 Burst Time: 3

 Process 5
 Arrival Time: 4
 Burst Time: 4

 ------------------------------------------------------------------------------------------------
 | PROCESS | BURST TIME | ARRIVAL TIME | COMPLETION TIME | WAITING TIME | TURNAROUND TIME | RESPONSE TIME |
 ------------------------------------------------------------------------------------------------
 |   P1    |     6      |      2       |       15        |      7       |       13        |       1       |
 ------------------------------------------------------------------------------------------------
 |   P2    |     2      |      5       |        7        |      0       |        2        |       0       |
 ------------------------------------------------------------------------------------------------
 |   P3    |     8      |      1       |       23        |     14       |       22        |      14       |
 ------------------------------------------------------------------------------------------------
 |   P4    |     3      |      0       |        3        |      0       |        3        |       0       |
 ------------------------------------------------------------------------------------------------
 |   P5    |     4      |      4       |       10        |      2       |        6        |       0       |
 ------------------------------------------------------------------------------------------------

 Total Turnaround Time: 46.00 | Average Turnaround Time: 9.20
 Total Waiting Time:    23.00 | Average Waiting Time:    4.60
 Total Completion Time: 58.00 | Average Completion Time: 11.60
 Total Response Time:   15.00 | Average Response Time:   3.00

  ----------------------------------------
                 GANTT CHART
  ----------------------------------------

  ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
 | P4 | P4 | P4 | P1 | P5 | P2 | P2 | P5 | P5 | P5 | P1 | P1 | P1 | P1 | P1 | P3 | P3 | P3 | P3 | P3 | P3 | P3 | P3 |
  ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  0    1    2    3    4    5    6    7 _  8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23
```

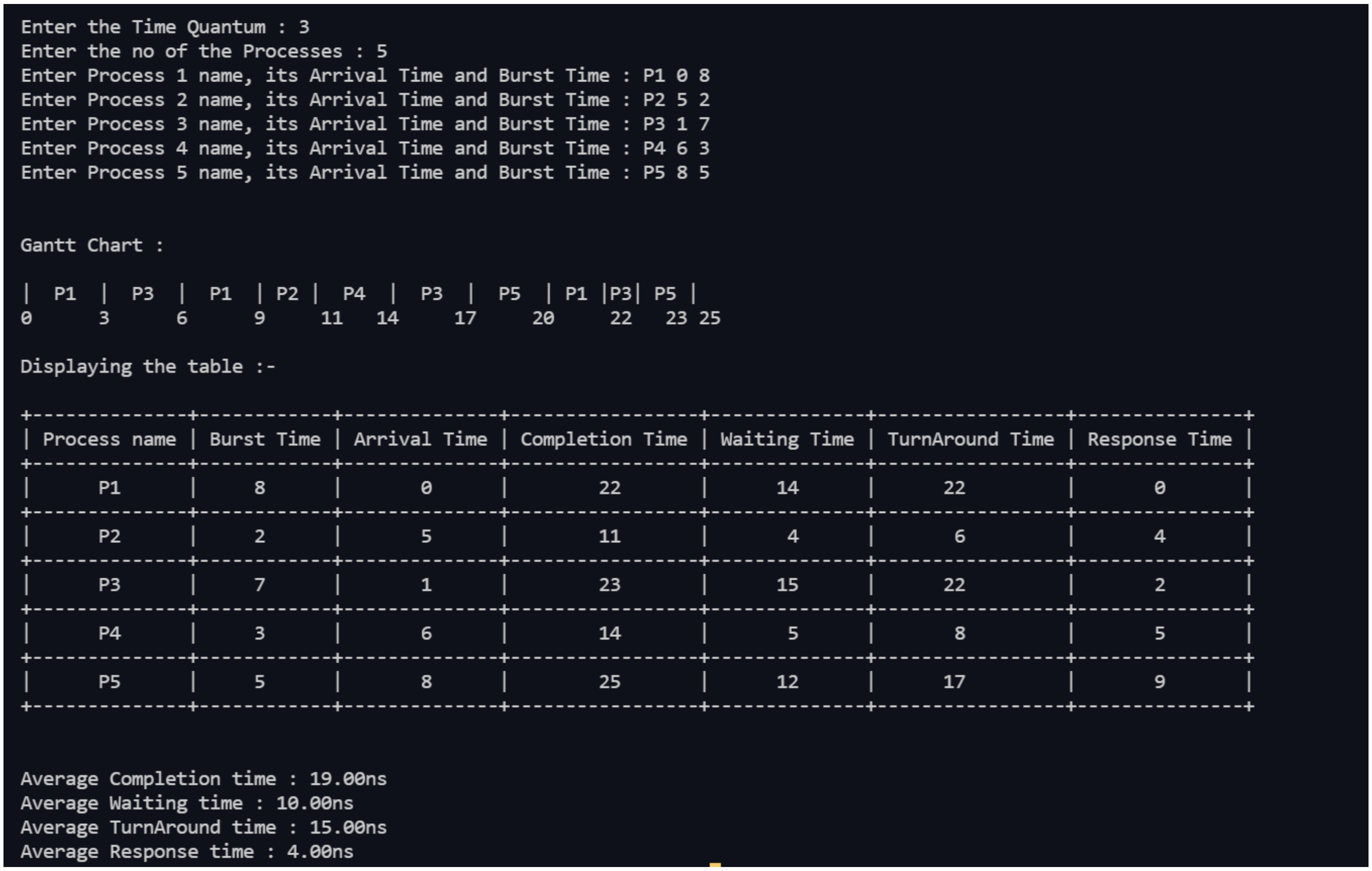
Name : Mohd Adil

Roll No : 20BCS042

A program to implement the Round Robin scheduling algorithm with time quantum =t and find the
average turnaround time, waiting time, completion time and response time for the overall process. Also Printing
Gantt chart for it.

```
#include<iostream>
#include<vector>
#include<queue>

using namespace std;

struct Process{
    char Pname[3];
    int id;
    int Times[6];
    vector<pair<int,int>> SEtime;
    //for calculations
    int b;
};

vector<Process> v;
vector<pair<int, int>> TimeSet;
int n;
float avgc, avgw, avgt, avgr;

void printProcess(string pname, int s, int e){
    TimeSet. push_back(make_pair (s, e));
    cout<<"|";
    int block = e-s;
    for(int i=0; i<e-s-1; i++) cout<<" ";
    cout<<pname;
    for(int i=0; i<e-s-1; i++) cout<<" ";
}

void printGantt(){
    cout<<"0";
    cout<<"     ";
    for(auto T:TimeSet){
        printf("%2d", T.second);
        for(int i=0; i<2*(T.second-T.first)-1; i++) cout<<" ";
    }
}

void RoundRobin(int TimeQuantum){
    queue<Process> pq;
    pq.push(v.front());

    int currentTime=0;
    bool visited[n] = {false};
    visited[0]=true;

    while(!pq.empty()){
        Process p = pq.front();
        pq.pop();
        int Pid = p.id;
        int tb = min(TimeQuantum, p.b);

        pair<int, int> pr;
        pr.first = currentTime;
        currentTime+=tb;
        pr.second = currentTime;
        v[Pid].SEtime.push_back(pr);
        v[Pid].b-=tb;
        // cout<<v[Pid].Pname<<" "<<v[Pid].b<<endl;
        printProcess(p.Pname, pr.first, pr.second);

        for(int i=0; i<v.size(); i++){
            if(v[i].Times[0]<=currentTime && v[i].b!=0 && !visited[v[i].id]){
                // cout<<v[i].Pname<<" ";
                pq.push(v[i]);
                visited[v[i].id]=true;
            }
        }
        if(v[Pid].b!=0){
            pq.push(v[Pid]);
        }
    }
    cout<<"|";
    cout<<endl;
    printGantt();
}

void calculateTimes(){

    float sumc=0, sumw=0, sumt=0, sumr=0;

    //calculating completion time
    for(auto &p : v){
        int sze = p.SEtime.size();
        p.Times[2] = p.SEtime[sze-1].second;
        sumc += p.Times[2];
    }

    //calculating turn around time
    // CT-AR
    for(auto &p : v){
        p.Times[4] = p.Times[2] - p.Times[0];
        sumt += p.Times[4];
    }


    //calculating waiting time
    // TAT-BT
    for(auto &p : v){
        p.Times[3] = p.Times[2] - p.Times[0] - p.Times[1];
        sumw += p.Times[3];
    }


    //calculating Response Time
    // First - AT
    for(auto &p : v){
        p.Times[5] = p.SEtime[0].first - p.Times[0];
        sumr += p.Times[5];
    }

    //calculating avg(s) time
    avgc = sumc/n;
    avgw = sumw/n;
    avgt = sumt/n;
    avgr = sumr/n;
}

void display(){
    cout<<"\n\nDisplaying the table :- ";

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

    for(auto i:v){
        printf("\n|      %s      |    %2d      |      %2d      |       %2d        |      %2d      |      %2d         |      %2d       |"
               ,i.Pname, i.Times[1], i.Times[0], i.Times[2], i.Times[3], i.Times[4], i.Times[5]);
    cout<<"\n+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+";
    }

    cout<<"\n\n";
    printf("\nAverage Completion time : %.2fns", avgc);
    printf("\nAverage Waiting time : %.2fns", avgw);
    printf("\nAverage TurnAround time : %.2fns", avgt);
    printf("\nAverage Response time : %.2fns", avgr);
}

int main(){
    int TimeQuantum;
    cout<<"Enter the Time Quantum : ";
    cin>>TimeQuantum;
    cout<<"Enter the no of the Processes : ";
    cin>>n;
    for(int i=0; i<n; i++){
        struct Process p;
        cout<<"Enter Process "<<i+1<<" name, its Arrival Time and Burst Time : ";
        cin>>p.Pname>>p.Times[0]>>p.Times[1];
        p.id=i;
        p.b = p.Times[1];
        v.push_back(p);
    }
    cout<<endl<<endl<<"Gantt Chart : "<<endl<<endl;
    RoundRobin(TimeQuantum);
    calculateTimes();
    display();
    return 0;
}
```

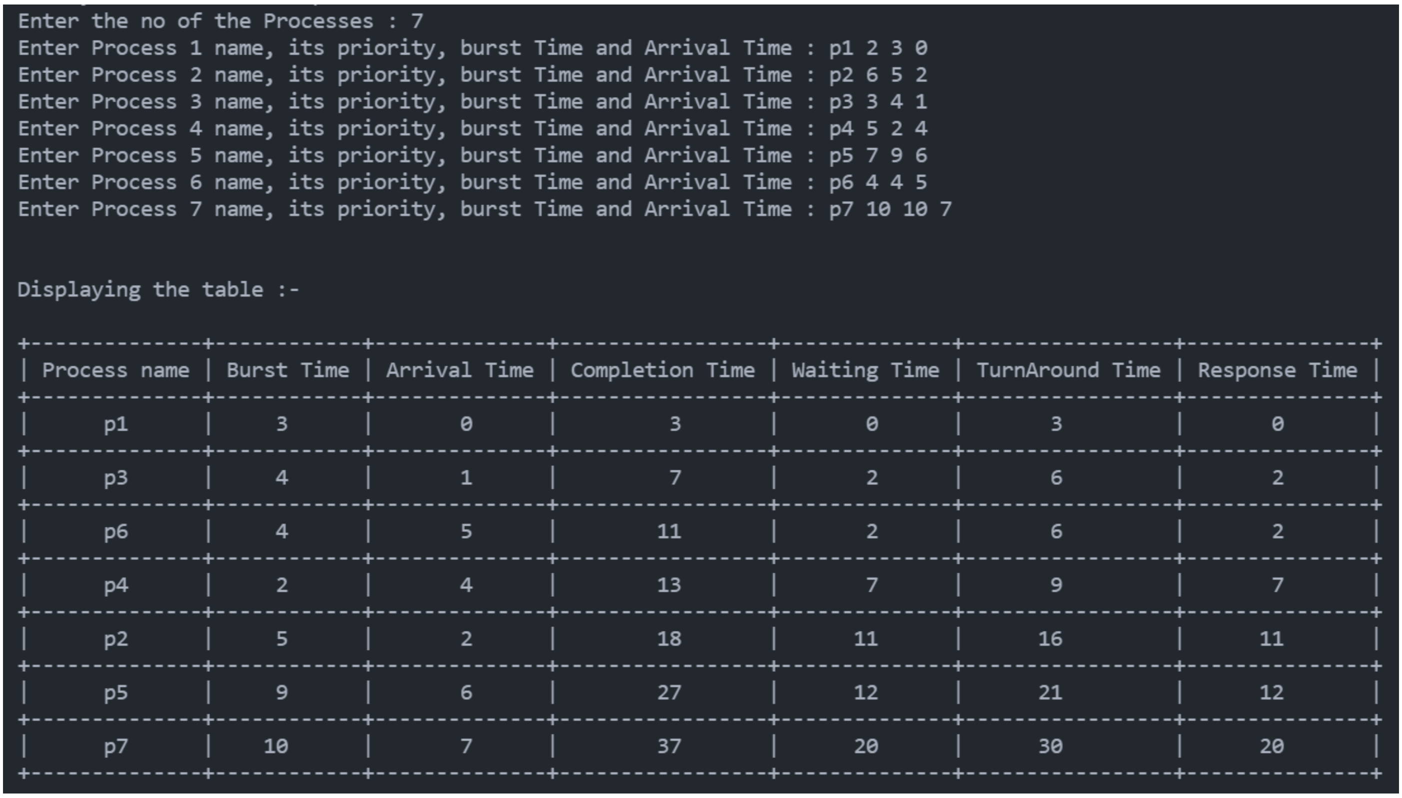
Output :



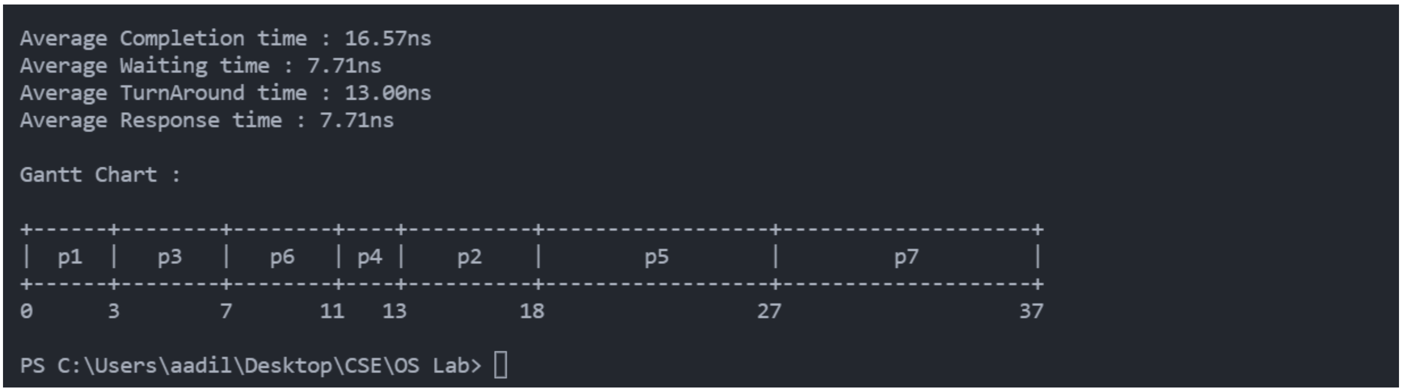
```
Enter the Time Quantum : 3
Enter the no of the Processes : 5
Enter Process 1 name, its Arrival Time and Burst Time : P1 0 8
Enter Process 2 name, its Arrival Time and Burst Time : P2 5 2
Enter Process 3 name, its Arrival Time and Burst Time : P3 1 7
Enter Process 4 name, its Arrival Time and Burst Time : P4 6 3
Enter Process 5 name, its Arrival Time and Burst Time : P5 8 5


Gantt Chart :

|  P1  |  P3  |  P1  | P2 |  P4  |  P3  |  P5  | P1 |P3| P5 |
0      3      6      9     11   14     17     20     22   23 25

Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P1      |     8      |       0      |        22       |      14      |      22         |       0       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P2      |     2      |       5      |        11       |       4      |       6         |       4       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P3      |     7      |       1      |        23       |      15      |      22         |       2       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P4      |     3      |       6      |        14       |       5      |       8         |       5       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      P5      |     5      |       8      |        25       |      12      |      17         |       9       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+


Average Completion time : 19.00ns
Average Waiting time : 10.00ns
Average TurnAround time : 15.00ns
Average Response time : 4.00ns
```

```
//20BCS042 Mohd Adil
//program 6: Non-premetive priority scheduling
#include<iostream>
#include<vector>
using namespace std;

struct Process{
    char Pname[3];
    int arvlTime;
    int brstTime;
    int priority;
    int cmplTime;
    int wtngTime;
    int tartTime;
    int respTime;
};

struct priorityQ{
    Process pr;
    priorityQ *next;
};

priorityQ *push(priorityQ *front, Process process, char c){
    priorityQ *node = new priorityQ;
    node->pr = process;
    node->next=NULL;

//push according to priority
    if(c=='p'){
        if(front==NULL){
            front=node;
        }
        else if(front->pr.priority > process.priority){
            node->next=front;
            front=node;
        }
        else{
            priorityQ *tmp=front;
            while (tmp->next!=NULL && tmp->next->pr.priority <
process.priority){
                tmp=tmp->next;
            }

            node->next=tmp->next;
            tmp->next=node;
        }
    }
//push according to arrival time
    else{
        if(front==NULL){
            front=node;
        }
        else if(front->pr.arvlTime > process.arvlTime){
            node->next=front;
            front=node;
        }
        else{
            priorityQ *tmp=front;
            while (tmp->next!=NULL && tmp->next->pr.arvlTime <
process.arvlTime){
                tmp=tmp->next;
            }

            node->next=tmp->next;
            tmp->next=node;
        }
    }

    return front;
}

priorityQ *pop(priorityQ *front){
    front=front->next;
    return front;
}

Process front(priorityQ *front){
    return front->pr;
}

bool empty(priorityQ *front){
    return (front==NULL);
}

//ans vector
vector<Process> v;
int n;
float avgc, avgw, avgt;

void PriorityScheduling(priorityQ *pq1){
    int cmpt = front(pq1).brstTime;
    v.push_back(front(pq1));
    pq1 = pop(pq1);

    priorityQ *pq2=NULL;

    while(!empty(pq1)){
        while(!empty(pq1) && front(pq1).arvlTime < cmpt){
            pq2 = push(pq2, front(pq1), 'p');
            pq1 = pop(pq1);
        }

        cmpt += front(pq2).brstTime;
        v.push_back(front(pq2));
        pq2 = pop(pq2);
    }

    while(!empty(pq2)){
        v.push_back(front(pq2));
        pq2 = pop(pq2);
    }
}

void calculateTimes(){
    v.front().wtngTime=0;
    v.front().cmplTime = v.front().brstTime;
    float sumc=0, sumw=0, sumt=0;

    //calculating completion time
    int prv = v.front().cmplTime;
    sumc += prv;
    for(int i=1; i<n; i++){
        v[i].cmplTime = prv + v[i].brstTime;
        prv = v[i].cmplTime;
        sumc += v[i].cmplTime;
    }

    //calculating waiting time
    prv = v.front().cmplTime;
    for(int i=1; i<n; i++){
        v[i].wtngTime = prv - v[i].arvlTime;
        prv = v[i].cmplTime;
        sumw += v[i].wtngTime;
    }

    //calculating turn around time
    for(int i=0; i<n; i++){
        v[i].tartTime = v[i].brstTime + v[i].wtngTime;
        sumt += v[i].tartTime;
    }

    //calculating avg(s) time
    avgc = sumc/n;
    avgw = sumw/n;
    avgt = sumt/n;
}

void display(){
    cout<<"\n\nDisplaying the table :- ";

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

    for(auto i:v){
        printf("\n|      %s      |    %2d      |      %2d      |        %2d
      |      %2d      |      %2d         |      %2d       |"
               ,i.Pname, i.brstTime, i.arvlTime, i.cmplTime, i.wtngTime,
i.tartTime, i.wtngTime);
    cout<<"\n+--------------+------------+--------------+-----------------+-
-------------+-----------------+---------------+";
    }

    cout<<"\n\n";
    printf("\nAverage Completion time : %.2fns", avgc);
    printf("\nAverage Waiting time : %.2fns", avgw);
    printf("\nAverage TurnAround time : %.2fns", avgt);
    printf("\nAverage Response time : %.2fns", avgw);
}

void printGantt(){

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

    cout<<"\n\n+";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
    }
    cout<<"\n|";
    for(auto p:v){
        for(int i=0; i<p.brstTime-1; i++){
            cout<<" ";
        }
        cout<<p.Pname;
        for(int i=0; i<p.brstTime-1; i++){
            cout<<" ";
        }
        cout<<"|";
    }

    cout<<"\n+";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime; i++){
            cout<<"-";
        }
        cout<<"+";
    }

    cout<<"\n0";
    for(auto p:v){
        for(int i=0; i<2*p.brstTime-1; i++){
            cout<<" ";
        }
        printf("%2d", p.cmplTime);
    }
    cout<<"\n\n";
}

int main(){
    priorityQ *pq1=NULL;

    cout<<"Enter the no of the Processes : ";
    cin>>n;

    for(int i=0; i<n; i++){
        struct Process p;
        cout<<"Enter Process "<<i+1<<" name, its priority, burst Time and
Arrival Time : ";
        cin>>p.Pname>>p.priority>>p.brstTime>>p.arvlTime;
        pq1 = push(pq1, p, 'a');//initially pushed according to arrival time
    }
    PriorityScheduling(pq1);
    calculateTimes();
    display();
    printGantt();
    return 0;
}

//sample input:-
// 7 p1 2 3 0 p2 6 5 2 p3 3 4 1 p4 5 2 4 p5 7 9 6 p6 4 4 5 p7 10 10 7
```

OUTPUT



```
Enter the no of the Processes : 7
Enter Process 1 name, its priority, burst Time and Arrival Time : p1 2 3 0
Enter Process 2 name, its priority, burst Time and Arrival Time : p2 6 5 2
Enter Process 3 name, its priority, burst Time and Arrival Time : p3 3 4 1
Enter Process 4 name, its priority, burst Time and Arrival Time : p4 5 2 4
Enter Process 5 name, its priority, burst Time and Arrival Time : p5 7 9 6
Enter Process 6 name, its priority, burst Time and Arrival Time : p6 4 4 5
Enter Process 7 name, its priority, burst Time and Arrival Time : p7 10 10 7


Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p1      |     3      |      0       |        3        |      0       |        3        |       0       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p3      |     4      |      1       |        7        |      2       |        6        |       2       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p6      |     4      |      5       |       11        |      2       |        6        |       2       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p4      |     2      |      4       |       13        |      7       |        9        |       7       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p2      |     5      |      2       |       18        |      11      |       16        |      11       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p5      |     9      |      6       |       27        |      12      |       21        |      12       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p7      |     10     |      7       |       37        |      20      |       30        |      20       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
```

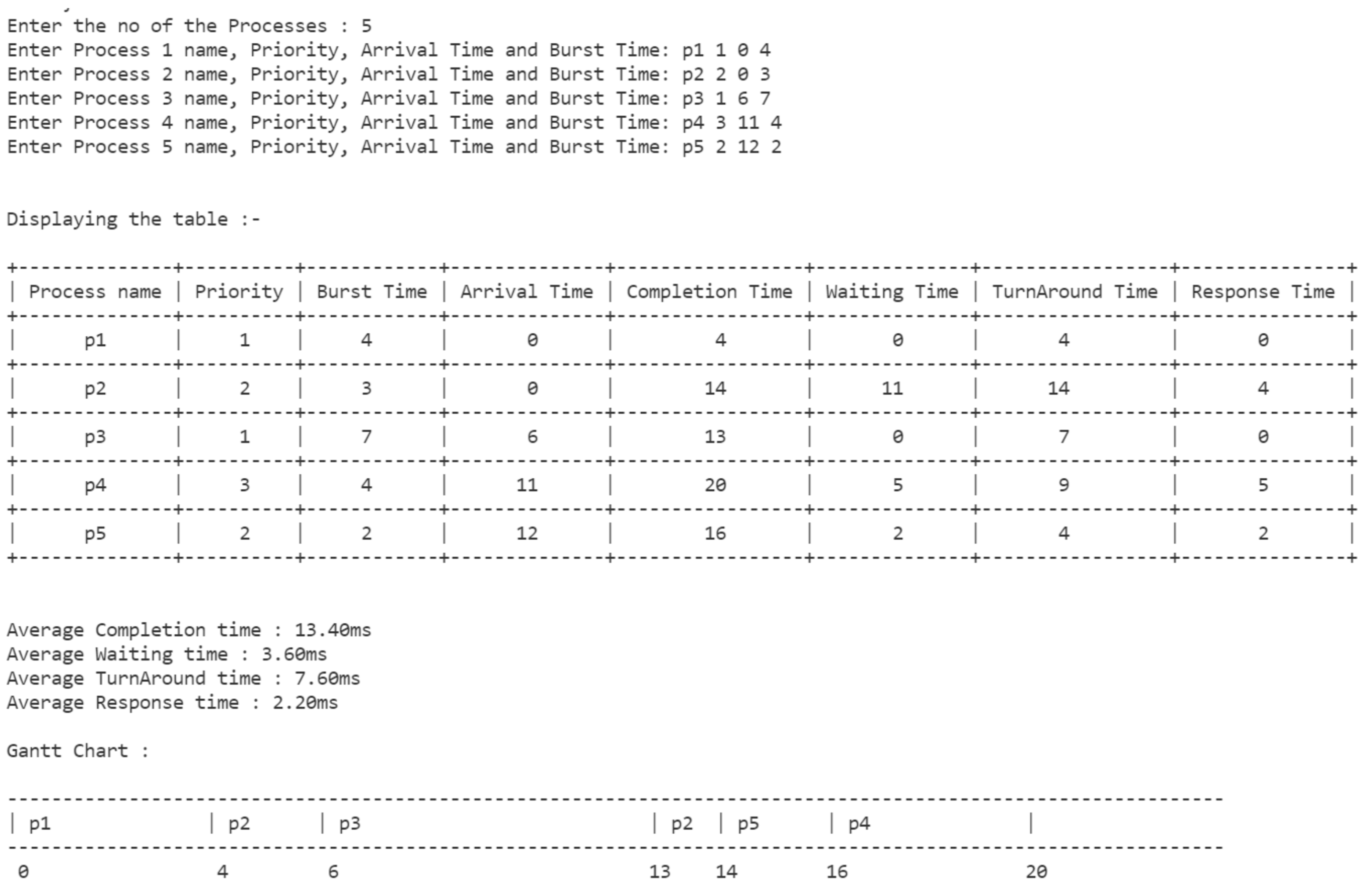


```
Average Completion time : 16.57ns
Average Waiting time : 7.71ns
Average TurnAround time : 13.00ns
Average Response time : 7.71ns

Gantt Chart :

+------+--------+--------+----+----------+------------------+--------------------+
|  p1  |   p3   |   p6   | p4 |    p2    |        p5        |         p7         |
+------+--------+--------+----+----------+------------------+--------------------+
0      3        7        11   13         18                 27                   37

PS C:\Users\aadil\Desktop\CSE\OS Lab> 
```

```
// Preemtive priority scheduling
// Handled edge cases + idle Time

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

struct Process
{
    char Pname[3];
    int prTY;
    int Times[6];
    pair<int, int> scope;
    int b;
};

struct Gantt
{
    int s;
    int e;
    string pname;
};

// ans vector
vector<Process> v;
vector<Gantt> vG;
vector<bool> visited;
int n, CurrTime = 0;
float avgc, avgw, avgt, avgr;

bool allVisited()
{
    // traverse the visited array and find a false, hence return it
    for (auto b : visited)
    {
        if (!b)
            return false;
    }
    return true;
}

void SRTF()
{
    while (!allVisited())
    {

        int min = 1000;
        int idx = -1;
        for (int i = 0; i < v.size(); i++)
        {
            if (v[i].Times[0] <= CurrTime && v[i].b != 0)
            {

                if (v[i].prTY < min)
                {
                    idx = i;
                    min = v[i].prTY;
                }
            }
        }
        if (idx != -1)
        {
            int t = CurrTime;
            v[idx].b--;
            if (v[idx].b == 0)
            {
                visited[idx] = true;
            }
            if (v[idx].scope.first == -1)
            {
                v[idx].scope.first = t;
            }
            v[idx].scope.second = t + 1;

            Gantt g;
            g.s = t;
            g.e = t + 1;
            g.pname = v[idx].Pname;
            vG.push_back(g);
        }
        CurrTime++;
    }
}

void calculateTimes()
{

    float sumc = 0, sumw = 0, sumt = 0, sumr = 0;

    // calculating completion time
    for (auto &p : v)
    {
        p.Times[2] = p.scope.second;
        sumc += p.Times[2];
    }

    // calculating turn around time
    //  CT-AR
    for (auto &p : v)
    {
        p.Times[4] = p.Times[2] - p.Times[0];
        sumt += p.Times[4];
    }

    // calculating waiting time
    //  TAT-BT
    for (auto &p : v)
    {
        p.Times[3] = p.Times[2] - p.Times[0] - p.Times[1];
        sumw += p.Times[3];
    }

    // calculating Response Time
    //  First - AT
    for (auto &p : v)
    {
        p.Times[5] = p.scope.first - p.Times[0];
        sumr += p.Times[5];
    }

    // calculating avg(s) time
    avgc = sumc / n;
    avgw = sumw / n;
    avgt = sumt / n;
    avgr = sumr / n;
}

void display()
{
    cout << "\n\nDisplaying the table :- ";

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

```
    cout << "\n+--------------+----------+------------+--------------+------
-----------+--------------+-----------------+---------------+";
    // cout<<"\n+--------------+------------+--------------+----------------
-+--------------+-----------------+---------------+";

    for (auto i : v)
    {
        printf("\n|      %s      |    %2d    |    %2d      |      %2d      |
       %2d       |      %2d      |      %2d         |      %2d       |",
i.Pname, i.prTY, i.Times[1], i.Times[0], i.Times[2], i.Times[3], i.Times[4],
i.Times[5]);
        // cout<<"\n+--------------+------------+--------------+------------
-----+--------------+-----------------+---------------+";
        cout << "\n+--------------+----------+------------+--------------+--
---------------+--------------+-----------------+---------------+";
    }

    cout << "\n\n";
    printf("\nAverage Completion time : %.2fms", avgc);
    printf("\nAverage Waiting time : %.2fms", avgw);
    printf("\nAverage TurnAround time : %.2fms", avgt);
    printf("\nAverage Response time : %.2fms", avgr);
}

void PrintGantt()
{
    cout << endl
         << endl
         << "Gantt Chart : " << endl
         << endl;
    cout << "---------------------------------------------------------------
-----------------------------------------------";
    cout << endl;

    vector<int> t;
    // vector<pair<int,int>> indices;
    string prv = "-1";
    for (int i = 0; i < CurrTime; i++)
    {
        string ch = "--";
        for (auto g : vG)
        {
            if (g.s == i)
            {
                ch = g.pname;
                break;
            }
        }

        if (prv != ch)
        {
            cout << "| " << ch << "  ";
            t.push_back(i);
        }
        else
            cout << "    ";
        prv = ch;
    }

    cout << "|" << endl;
    t.push_back(CurrTime);
    cout << "---------------------------------------------------------------
-----------------------------------------------";
    cout << endl;
    int prev = 0;
    for (int i = 0; i < t.size(); i++)
    {
        for (int j = 0; j < (t[i] - prev); j++)
        {
            cout << "    ";
        }
        // cout<<t[i];
        printf("%2d", t[i]);

        prev = t[i];
    }
}

int main()
{
    cout << "Enter the no of the Processes : ";
    cin >> n;

    for (int i = 0; i < n; i++)
    {
        struct Process p;
        cout << "Enter Process " << i + 1 << " name, Priority, Arrival Time
and Burst Time: ";
        cin >> p.Pname >> p.prTY >> p.Times[0] >> p.Times[1];
        p.b = p.Times[1];
        visited.push_back(false);
        p.scope.first = -1;
        p.scope.second = -1;
        v.push_back(p);
    }

    SRTF();
    calculateTimes();
    display();
    PrintGantt();
    return 0;
}
```

OUTPUT



```
Enter the no of the Processes : 5
Enter Process 1 name, Priority, Arrival Time and Burst Time: p1 1 0 4
Enter Process 2 name, Priority, Arrival Time and Burst Time: p2 2 0 3
Enter Process 3 name, Priority, Arrival Time and Burst Time: p3 1 6 7
Enter Process 4 name, Priority, Arrival Time and Burst Time: p4 3 11 4
Enter Process 5 name, Priority, Arrival Time and Burst Time: p5 2 12 2


Displaying the table :-

+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Priority | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p1      |    1     |     4      |      0       |        4        |      0       |        4        |       0       |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p2      |    2     |     3      |      0       |       14        |      11      |       14        |       4       |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p3      |    1     |     7      |      6       |       13        |      0       |        7        |       0       |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p4      |    3     |     4      |      11      |       20        |      5       |        9        |       5       |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p5      |    2     |     2      |      12      |       16        |      2       |        4        |       2       |
+--------------+----------+------------+--------------+-----------------+--------------+-----------------+---------------+


Average Completion time : 13.40ms
Average Waiting time : 3.60ms
Average TurnAround time : 7.60ms
Average Response time : 2.20ms

Gantt Chart :

--------------------------------------------------------------------------------------------------------------
| p1              | p2      | p3                          | p2  | p5      | p4              |
--------------------------------------------------------------------------------------------------------------
 0                 4         6                            13    14        16                20
```

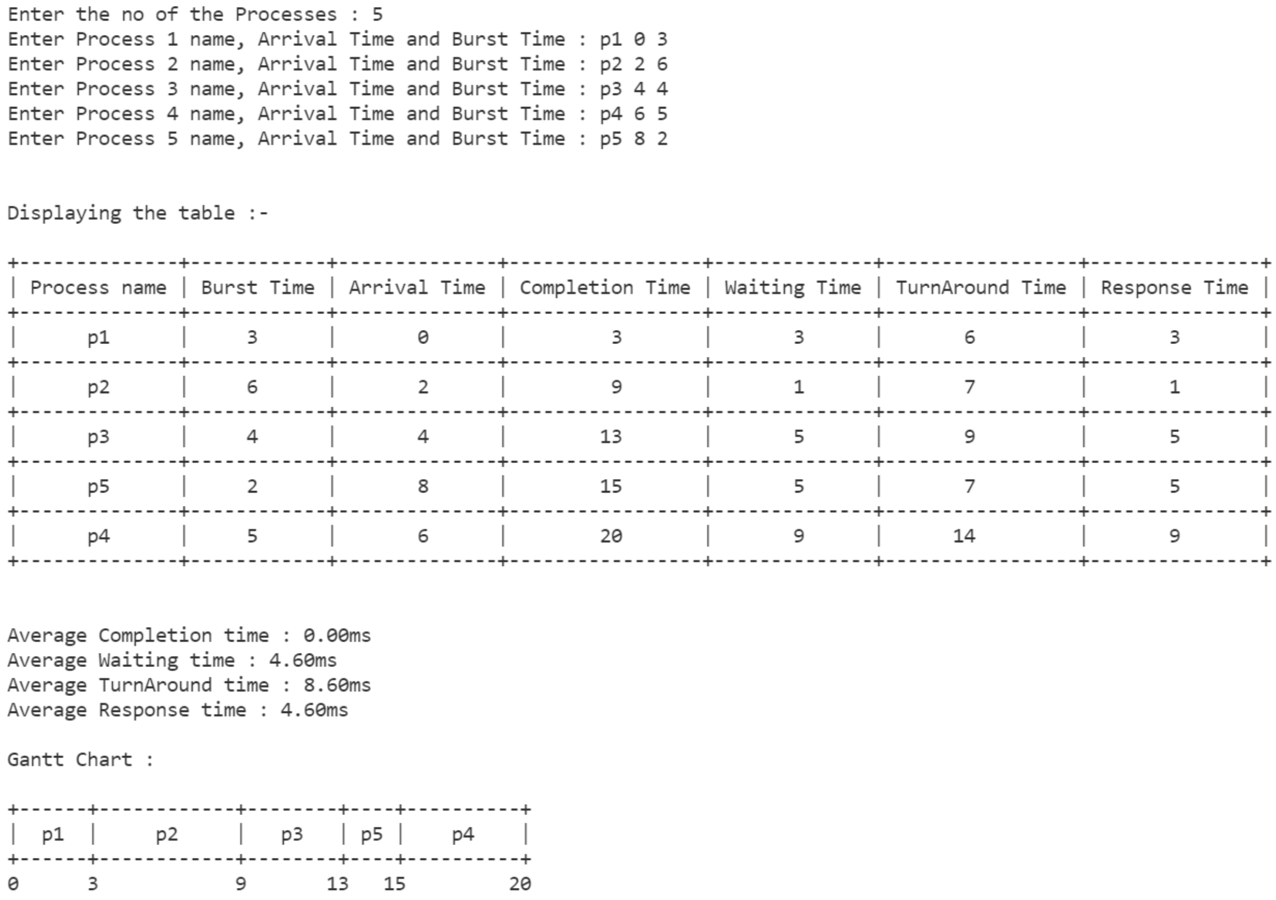
Thank you

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

struct Process{
    char Pname[3];
    int id;
    int Times[6];
};

int n;
float avgc, avgw, avgt;
vector<Process> input;
vector<Process> v;
vector<bool> visited;

bool completed(){
    for(auto b:visited) if(!b) return false;
    return true;
}

void HRRN(){
    int currentTime=0, mx=-1, idx=-1;
    float ResponseRatio;

    while(!completed()){
        for(auto p:input){
            if(!visited[p.id] && p.Times[0]<=currentTime){
                ResponseRatio = (1.00)*(currentTime - p.Times[0] +
p.Times[1])/p.Times[1];
                if(ResponseRatio>mx){
                    idx = p.id;
                    mx=ResponseRatio;
                }
            }
        }

        if(idx!=-1){
            visited[idx]=true;
            currentTime+=input[idx].Times[1];
            input[idx].Times[2]=currentTime;
            v.push_back(input[idx]);
```

```
                mx=-1;idx=-1;
            }
            else{
                currentTime++;
            }
        }
    }

    void calculateTimes(){
        v.front().Times[3]=0;
        float sumc=0, sumw=0, sumt=0;

        //calculating waiting time and Response Time
        int prv = v.front().Times[2];
        for(auto &p:v){
            p.Times[3] = prv - p.Times[0];
            p.Times[5] = p.Times[3];
            prv = p.Times[2];
            sumw += p.Times[3];
        }

        //calculating turn around time
        for(auto &p:v){
            p.Times[4] = p.Times[1] + p.Times[3];
            sumt += p.Times[4];
        }

        //calculating avg(s) time
        avgc = sumc/n;
        avgw = sumw/n;
        avgt = sumt/n;
    }

    void display(){
        cout<<"\n\nDisplaying the table :- ";

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

        for(auto i:v){
            printf("\n|      %s      |    %2d      |      %2d      |        %2d
      |      %2d      |      %2d         |      %2d       |"
```

```
               ,i.Pname, i.Times[1], i.Times[0], i.Times[2], i.Times[3],
i.Times[4], i.Times[5]);
    cout<<"\n+--------------+------------+--------------+-----------------+-
-------------+-----------------+---------------+";
    }

    cout<<"\n\n";
    printf("\nAverage Completion time : %.2fms", avgc);
    printf("\nAverage Waiting time : %.2fms", avgw);
    printf("\nAverage TurnAround time : %.2fms", avgt);
    printf("\nAverage Response time : %.2fms", avgw);
}

void printFree1(int x, int y, char a, char b){
    if(x==y) return;

    for(int i=0; i<2*(x-y); i++){
        cout<<a;
    }
    cout<<b;
}

void printFree2(int x, int y, int z){
    if((x-y)==z) return;

    // x-z to be printed
    int gap = x-y-z;
    for(int i=0; i<2*gap-1; i++){
        cout<<" ";
    }
    printf("%2d", x-z);
}

void printGantt(){

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

    // printing the upper part of Gantt Chart
    cout<<"\n\n+";
    int prv = 0;
    for(auto p:v){
        printFree1(p.Times[2]-prv, p.Times[1], '-', '+');
        for(int i=0; i<2*p.Times[1]; i++){
            cout<<"-";
        }
        cout<<"+";
        prv = p.Times[2];
    }

    // Printing the middle one
    cout<<"\n|";
    prv=0;
    for(auto p:v){
        printFree1(p.Times[2]-prv, p.Times[1], ' ', '|');
        for(int i=0; i<p.Times[1]-1; i++){
            cout<<" ";
        }
        cout<<p.Pname;
        for(int i=0; i<p.Times[1]-1; i++){
            cout<<" ";
        }
        cout<<"|";
        prv = p.Times[2];
    }

    // Printing the bottom one
    cout<<"\n+";
    prv = 0;
    for(auto p:v){
        printFree1(p.Times[2]-prv, p.Times[1], '-', '+');
        for(int i=0; i<2*p.Times[1]; i++){
            cout<<"-";
        }
        cout<<"+";
        prv = p.Times[2];
    }

    // Printing the indexes of times
    cout<<"\n0";
    prv=0;
    for(auto p:v){
        printFree2(p.Times[2], prv, p.Times[1]);
        for(int i=0; i<2*p.Times[1]-1; i++){
            cout<<" ";
        }
        printf("%2d", p.Times[2]);
        prv = p.Times[2];
    }
    cout<<"\n\n";
}

int main(){
    cout<<"Enter the no of the Processes : ";
    cin>>n;

    for(int i=0; i<n; i++){
        struct Process p;
        cout<<"Enter Process "<<i+1<<" name, Arrival Time and Burst Time :
";
        cin>>p.Pname>>p.Times[0]>>p.Times[1];
        p.id=i;
        visited.push_back(false);
        input.push_back(p);
    }

    HRRN();
    calculateTimes();
    display();
    printGantt();
    return 0;
}
```

OUTPUT



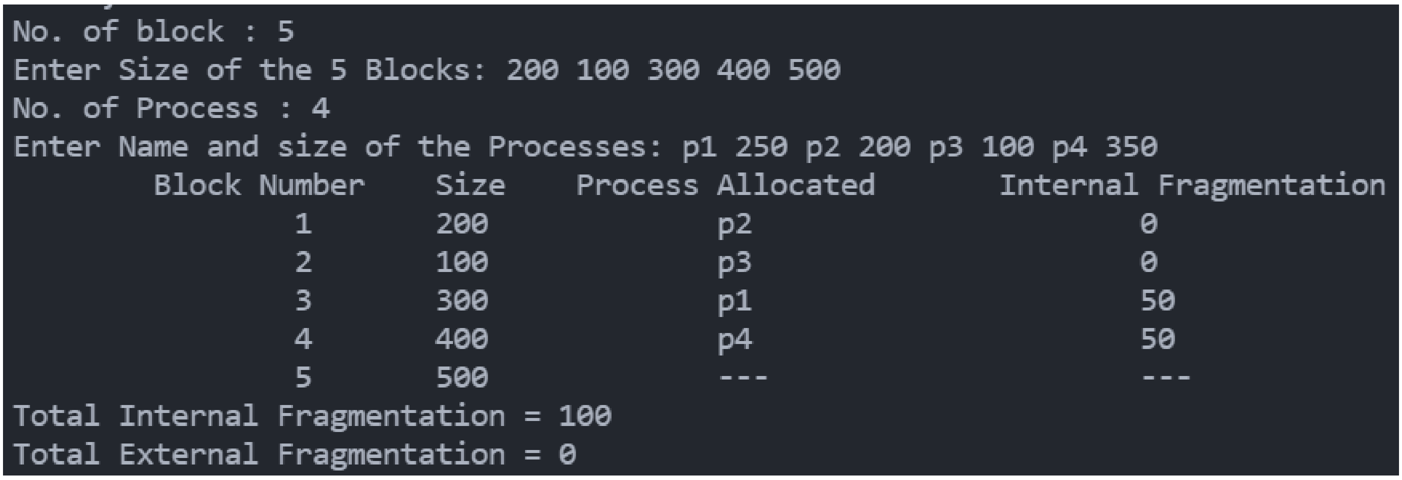
```
Enter the no of the Processes : 5
Enter Process 1 name, Arrival Time and Burst Time : p1 0 3
Enter Process 2 name, Arrival Time and Burst Time : p2 2 6
Enter Process 3 name, Arrival Time and Burst Time : p3 4 4
Enter Process 4 name, Arrival Time and Burst Time : p4 6 5
Enter Process 5 name, Arrival Time and Burst Time : p5 8 2


Displaying the table :-

+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
| Process name | Burst Time | Arrival Time | Completion Time | Waiting Time | TurnAround Time | Response Time |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p1      |      3     |       0      |        3        |      3       |        6        |       3       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p2      |      6     |       2      |        9        |      1       |        7        |       1       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p3      |      4     |       4      |       13        |      5       |        9        |       5       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p5      |      2     |       8      |       15        |      5       |        7        |       5       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+
|      p4      |      5     |       6      |       20        |      9       |       14        |       9       |
+--------------+------------+--------------+-----------------+--------------+-----------------+---------------+


Average Completion time : 0.00ms
Average Waiting time : 4.60ms
Average TurnAround time : 8.60ms
Average Response time : 4.60ms

Gantt Chart :

+------+------------+--------+----+----------+
|  p1  |     p2     |   p3   | p5 |    p4    |
+------+------------+--------+----+----------+
0      3            9        13   15         20
```

Thank you

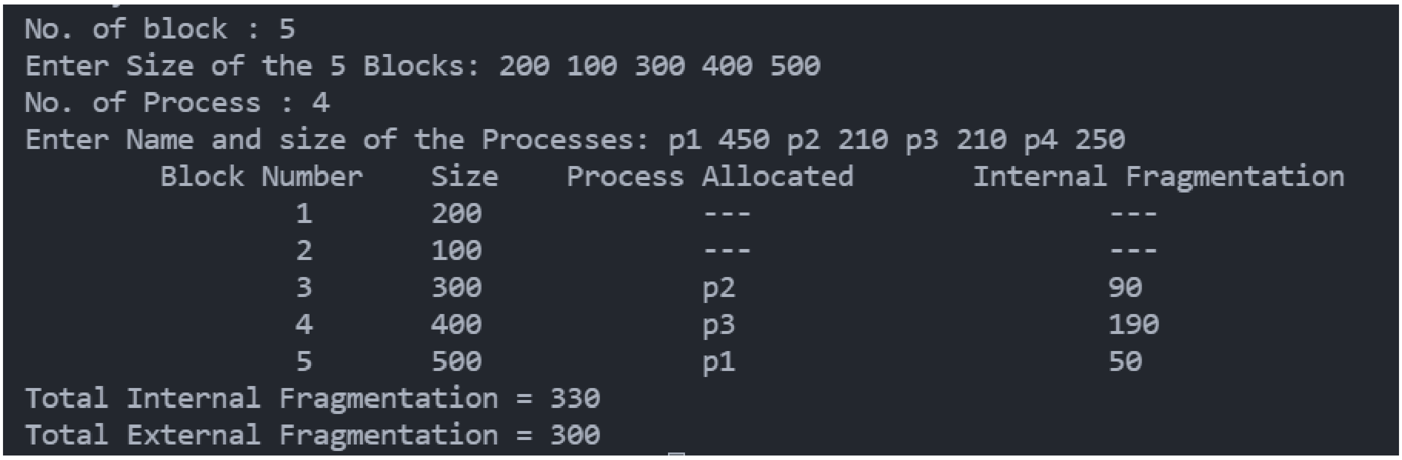
20BCS402 MOHD ADIL Program 9| First fit memory management algorithm.

```
#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++)
    {
        for (int j = 0; j < n; j++)
        {
            if (Processes[i].memory <= blocks[j].rem)
            {
                Processes[i].allocated = true;
                blocks[j].used = true;
                blocks[j].rem = blocks[j].size - Processes[i].memory;
                blocks[j].processAllocated = Processes[i];
                break;
            }
            else
            {
                continue;
            }
        }
    }
    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;
    for (int i = 0; i < m; i++)
    {
        if (Processes[i].allocated == false)
        {
            flag = false;
            break;
        }
        else
        {
            continue;
        }
    }

    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;
            }
        }
    }
    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 250 p2 200 p3 100 p4 350
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             p2                      0
                2       100             p3                      0
                3       300             p1                      50
                4       400             p4                      50
                5       500             ---                     ---
Total Internal Fragmentation = 100
Total External Fragmentation = 0
```



```
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 450 p2 210 p3 210 p4 250
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             ---                     ---
                2       100             ---                     ---
                3       300             p2                      90
                4       400             p3                      190
                5       500             p1                      50
Total Internal Fragmentation = 330
Total External Fragmentation = 300
```

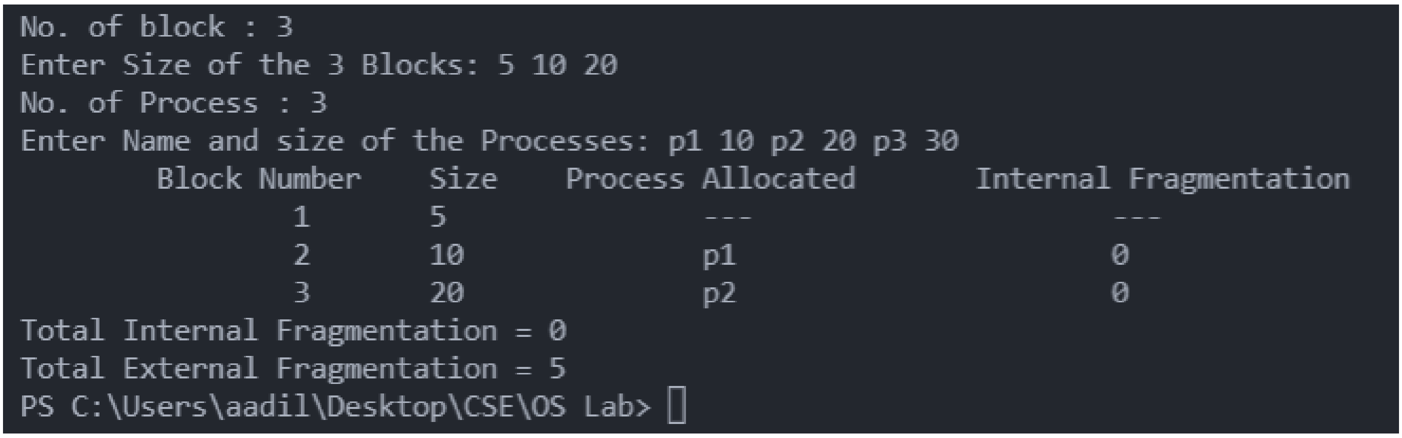
Thank you

20BCS402 MOHD ADIL Program 10| Next fit memory management algorithm.

```
#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
    int j = 0;
    for (int i = 0; i < m; i++)
    {
        int prv = j;
        do
        {
            if (Processes[i].memory <= blocks[j].rem && blocks[j].used ==
false)
            {
                Processes[i].allocated = true;
                blocks[j].used = true;
                blocks[j].rem = blocks[j].size - Processes[i].memory;
                blocks[j].processAllocated = Processes[i];
                break;
            }
            else
            {
                j = (j + 1) % n;
            }
        } while (j != prv);
    }
    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;
    for (int i = 0; i < m; i++)
    {
        if (Processes[i].allocated == false)
        {
            flag = false;
            break;
```

```cpp
        }
        else
        {
            continue;
        }
    }

    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;
            }
        }
    }
    cout << "Total Internal Fragmentation = " << IF << endl;
    cout << "Total External Fragmentation = " << EF << endl;
    return 0;
}
```

OUTPUT



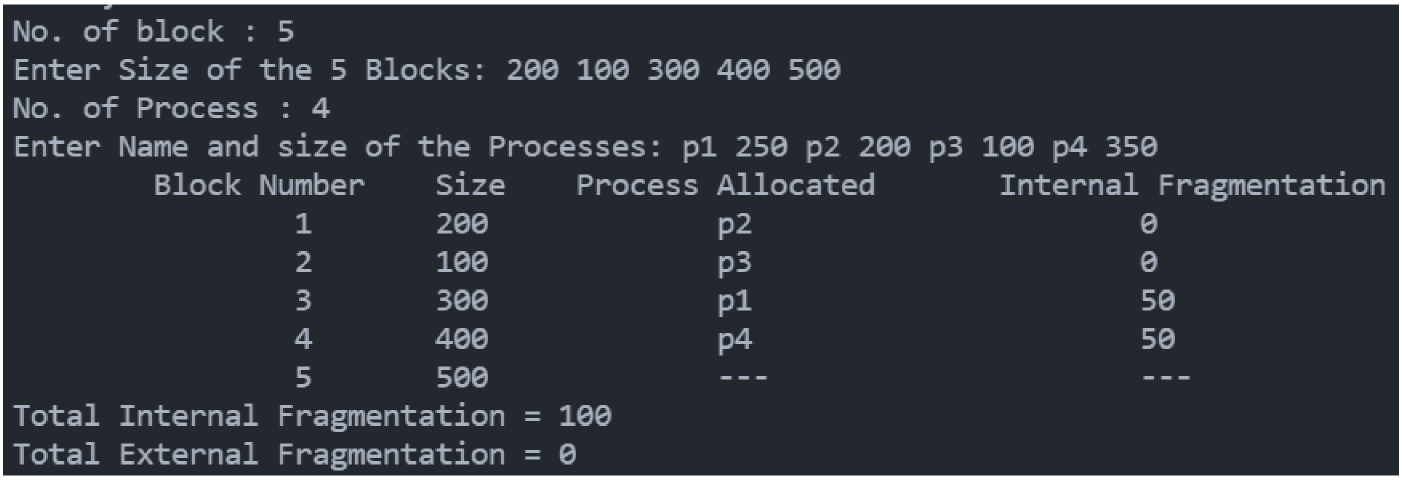
```
No. of block : 3
Enter Size of the 3 Blocks: 5 10 20
No. of Process : 3
Enter Name and size of the Processes: p1 10 p2 20 p3 30
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       5               ---                     ---
                2       10              p1                      0
                3       20              p2                      0
Total Internal Fragmentation = 0
Total External Fragmentation = 5
PS C:\Users\aadil\Desktop\CSE\OS Lab>
```

Thank you

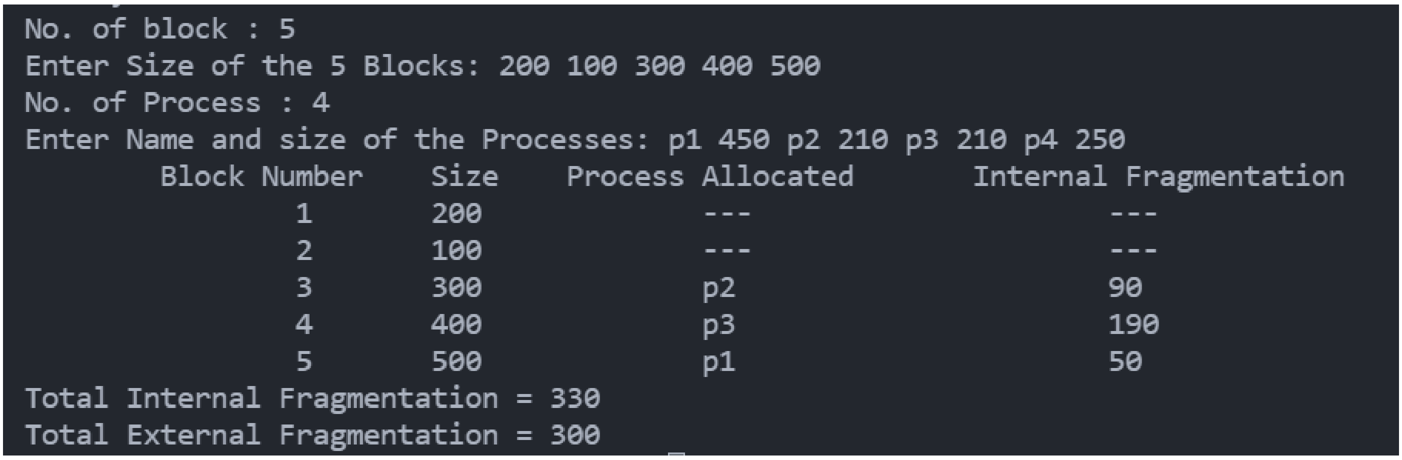
Program 11| Best fit memory management algorithm.

```cpp
#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
    int j = 0;
    for (int i = 0; i < m; i++)
    {
        int prv = j;
        do
        {
            if (Processes[i].memory <= blocks[j].rem && blocks[j].used ==
false)
            {
                Processes[i].allocated = true;
                blocks[j].used = true;
                blocks[j].rem = blocks[j].size - Processes[i].memory;
                blocks[j].processAllocated = Processes[i];
                break;
            }
            else
            {
                j = (j + 1) % n;
            }
        } while (j != prv);
    }
    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;
    for (int i = 0; i < m; i++)
    {
        if (Processes[i].allocated == false)
        {
            flag = false;
            break;
        }
        else
        {
            continue;
        }
    }

    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;
            }
        }
    }
    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 250 p2 200 p3 100 p4 350
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             p2                      0
                2       100             p3                      0
                3       300             p1                      50
                4       400             p4                      50
                5       500             ---                     ---
Total Internal Fragmentation = 100
Total External Fragmentation = 0
```



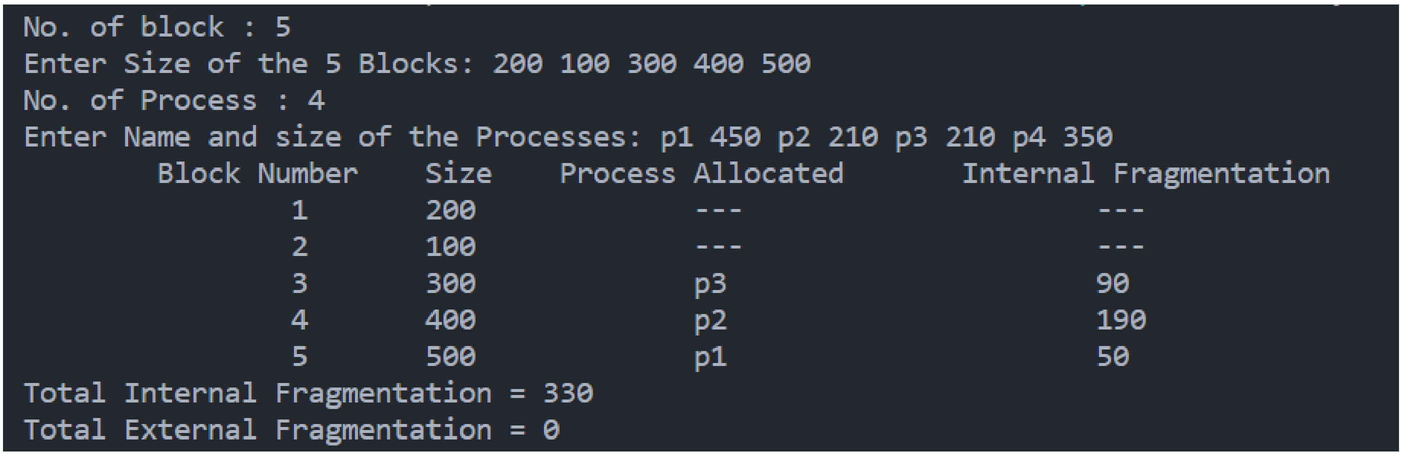
```
No. of block : 5
Enter Size of the 5 Blocks: 200 100 300 400 500
No. of Process : 4
Enter Name and size of the Processes: p1 450 p2 210 p3 210 p4 250
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             ---                     ---
                2       100             ---                     ---
                3       300             p2                      90
                4       400             p3                      190
                5       500             p1                      50
Total Internal Fragmentation = 330
Total External Fragmentation = 300
```

Thank you

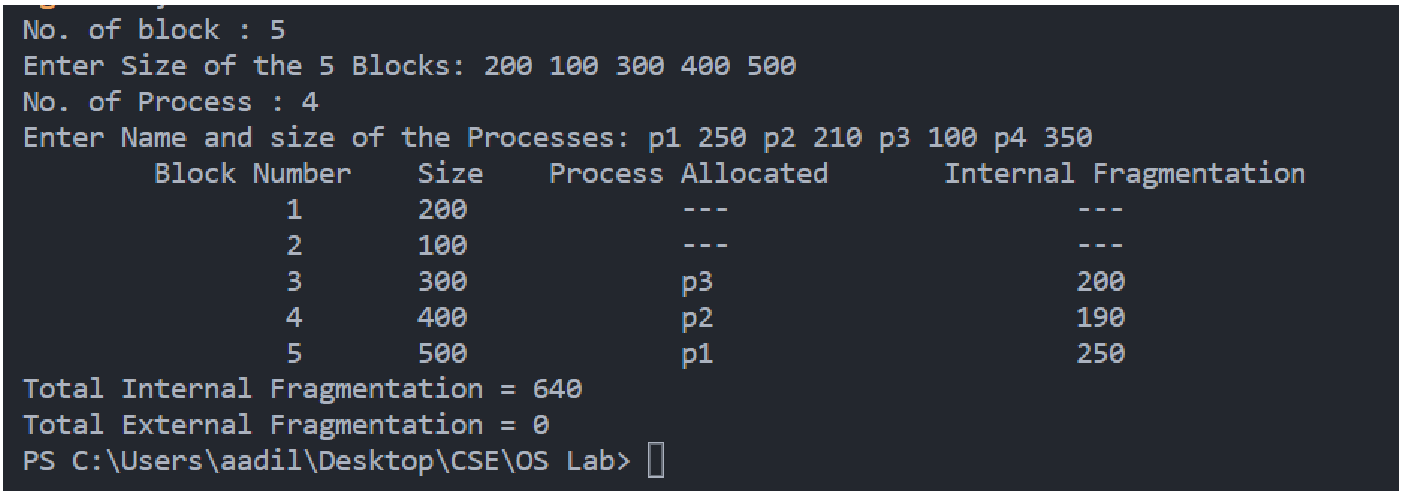
Program 12| Worst fit memory management algorithm.

```cpp
#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, max = INT16_MIN;
        for (int j = 0; j < n; j++)
        {
            if (Processes[i].memory <= blocks[j].rem && blocks[j].used ==
false && blocks[j].rem > max)
            {
                max = 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;
    for (int i = 0; i < m; i++)
    {
        if (Processes[i].allocated == false)
        {
            flag = false;
            break;
        }
        else
        {
            continue;
        }
    }

    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;
            }
        }
    }
    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 450 p2 210 p3 210 p4 350
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             ---                     ---
                2       100             ---                     ---
                3       300             p3                      90
                4       400             p2                      190
                5       500             p1                      50
Total Internal Fragmentation = 330
Total External Fragmentation = 0
```



```
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 250 p2 210 p3 100 p4 350
        Block Number    Size    Process Allocated       Internal Fragmentation
                1       200             ---                     ---
                2       100             ---                     ---
                3       300             p3                      200
                4       400             p2                      190
                5       500             p1                      250
Total Internal Fragmentation = 640
Total External Fragmentation = 0
PS C:\Users\aadil\Desktop\CSE\OS Lab> 
```

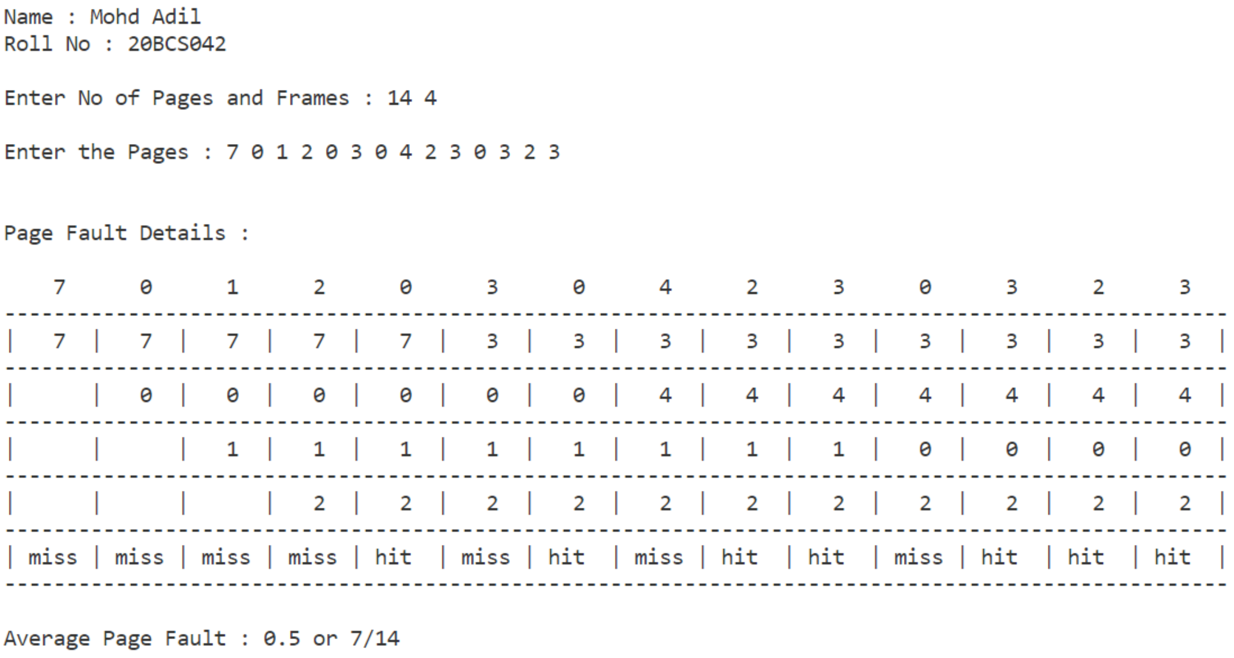
Thank you

20BCS402 MOHD ADIL Program 13 | First in First Out (FIFO) page replacement algorithm

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

bool check(int *present, int noFrames, int e){
    for(int i=0; i<noFrames; i++){
        if(present[i]==e) return true;
    }
    return false;
}

void FIFOPageRepAlgo(int *pages, int noPages, int noFrames){
    int chance=0, miss=0, hits=0;

    int *present = new int[noFrames];
    for(int i=0; i<noFrames; i++) present[i]=-1;

    // declare a chart for printing
    int **chart = new int*[noFrames+2];
    for(int i=0; i<noFrames+2; i++){
        chart[i] = new int[noPages];
        for(int j=0; j<noPages; j++){
            chart[i][j]=-1;
        }
    }

    for(int i=0; i<noPages; i++){
        chart[0][i] = pages[i];
    }

    int k=0;
    for(int i=0; i<noPages; i++){

        bool missOrHit=true;

        // if page no was not found in any of the frames
        // miss case
        if(!check(present, noFrames, pages[i])){
            present[chance]=pages[i];
            chance=(chance+1)%noFrames;
            missOrHit=false;
            miss++;
        }
        // hit case
        else{
            hits++;
        }

        // add the values in the chart
        int j;
        for(j=0; j<noFrames; j++){
            chart[j+1][k] = present[j];
        }

        // update miss or hit in chart
        missOrHit ? chart[j+1][k]=1 : chart[j+1][k]=0;
        k++;
    }

    cout<<endl<<endl<<"Page Fault Details : "<<endl<<endl;

    // Printing the chart
    int NOH = (7*noPages)+1;

    // First row
    for(int j=0; j<noPages; j++){
        printf("   %2d  ", chart[0][j]);
    }
    cout<<endl;
    for(int k=0; k<NOH; k++){
        cout<<"-";
    }
    cout<<endl;

    // middle portion
    for(int i=1; i<noFrames+1; i++){
        for(int j=0; j<noPages; j++){
            if(chart[i][j]==-1) printf("|       ", chart[i][j]);
            else printf("|  %2d  ", chart[i][j]);
        }
        cout<<"|"<<endl;
        for(int k=0; k<NOH; k++){
            cout<<"-";
        }
        cout<<endl;
    }


    // last row
    for(int j=0; j<noPages; j++){
        if(chart[noFrames+1][j]==1) cout<<"| hit  ";
```

```cpp
            else cout<<"| miss ";
        }
        cout<<"|"<<endl;
        for(int k=0; k<NOH; k++){
            cout<<"-";
        }

        cout<<endl<<endl<<"Average Page Fault : "<<((float)miss/noPages)<<" or
    "<<miss<<"/"<<noPages<<endl<<endl;
    }

    int main(){
        int noPages, noFrames;

        cout<<"\n\nName : Mohd Adil \nRoll No : 20BCS042";
        cout<<"\n\nEnter No of Pages and Frames : ";
        cin>>noPages>>noFrames;

        int *pages = new int[noPages];
        cout<<"\nEnter the Pages : ";
        for(int i=0; i<noPages; i++) cin>>pages[i];

        FIFOPageRepAlgo(pages, noPages, noFrames);
        return 0;
    }

    // sample input:
    // 14 4 7 0 1 2 0 3 0 4 2 3 0 3 2 3
```

OUTPUT

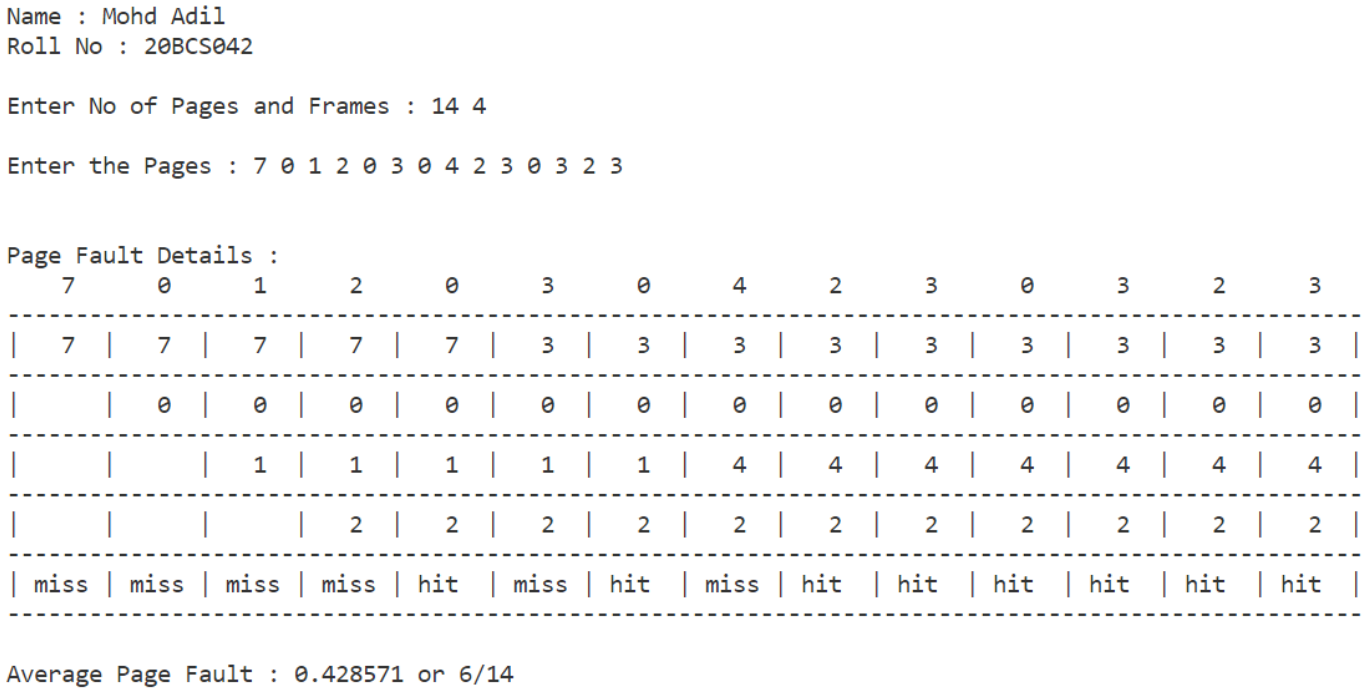


```
Name : Mohd Adil
Roll No : 20BCS042

Enter No of Pages and Frames : 14 4

Enter the Pages : 7 0 1 2 0 3 0 4 2 3 0 3 2 3


Page Fault Details :

   7      0      1      2      0      3      0      4      2      3      0      3      2      3
---------------------------------------------------------------------------------------------------
|  7   |  7   |  7   |  7   |  7   |  3   |  3   |  3   |  3   |  3   |  3   |  3   |  3   |  3   |
---------------------------------------------------------------------------------------------------
|      |  0   |  0   |  0   |  0   |  0   |  0   |  4   |  4   |  4   |  4   |  4   |  4   |  4   |
---------------------------------------------------------------------------------------------------
|      |      |  1   |  1   |  1   |  1   |  1   |  1   |  1   |  1   |  0   |  0   |  0   |  0   |
---------------------------------------------------------------------------------------------------
|      |      |      |  2   |  2   |  2   |  2   |  2   |  2   |  2   |  2   |  2   |  2   |  2   |
---------------------------------------------------------------------------------------------------
| miss | miss | miss | miss | hit  | miss | hit  | miss | hit  | hit  | miss | hit  | hit  | hit  |
---------------------------------------------------------------------------------------------------

Average Page Fault : 0.5 or 7/14
```

20BCS402 MOHD ADIL Program 14 | Least Recently Used (LRU) page replacement algorithm

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

bool check(int *present, int noFrames, int e){
    for(int i=0; i<noFrames; i++){
        if(present[i]==e) return true;
    }
    return false;
}

int search(int *arr, int sze, int e, bool st){
    if(st){
        for(int i=0; i<sze; i++){
            if(arr[i]==e) return i;
        }
    }
    else{
        for(int i=sze-1; i>=0; i--){
            if(arr[i]==e) return i;
        }
    }
    return -1;
}

int FindLRU(int *present, int *pages, int noFrames, int sze){
    int mn=INT16_MAX;
    for(int i=0; i<noFrames; i++){
        mn=min(search(pages, sze, present[i], false), mn);
    }
    return mn;
}

void LRUPageRepAlgo(int *pages, int noPages, int noFrames){
    int chance=0, miss=0, hits=0;

    int *present = new int[noFrames];
    for(int i=0; i<noFrames; i++) present[i]=-1;

    // declare a chart for printing
    int **chart = new int*[noFrames+2];
    for(int i=0; i<noFrames+2; i++){
        chart[i] = new int[noPages];
        for(int j=0; j<noPages; j++){
            chart[i][j]=-1;
        }
    }

    for(int i=0; i<noPages; i++){
        chart[0][i] = pages[i];
    }

    int k=0;

    // FIFO
    for(int i=0; i<noFrames; i++){

        bool missOrHit=true;

        // if page no was not found in any of the frames
        // miss case
        if(!check(present, noFrames, pages[i])){
            present[chance]=pages[i];
            chance=(chance+1)%noFrames;
            missOrHit=false;
            miss++;
        }
        // hit case
        else{
            hits++;
        }

        // add the values in the chart
        int j;
        for(j=0; j<noFrames; j++){
            chart[j+1][k] = present[j];
        }

        // update miss or hit in chart
        missOrHit ? chart[j+1][k]=1 : chart[j+1][k]=0;
        k++;
    }

    // LRU
    for(int i=noFrames; i<noPages; i++){

        bool missOrHit=true;
        // if page no was not found in any of the frames
        // miss case
        if(!check(present, noFrames, pages[i])){
            int lruIdx = FindLRU(present, pages, noFrames, i+1);
            int presentPageIDx = search(present, noFrames, pages[lruIdx],
true);

            present[presentPageIDx]=pages[i];
            missOrHit=false;
            miss++;
        }
        // hit case
        else{
            hits++;
        }

        // add the values in the chart
        int j;
        for(j=0; j<noFrames; j++){
            chart[j+1][k] = present[j];
        }

        // update miss or hit in chart
        missOrHit ? chart[j+1][k]=1 : chart[j+1][k]=0;
        k++;
    }

    cout<<endl<<endl<<"Page Fault Details : "<<endl<<endl;

    // Printing the chart
    int NOH = (7*noPages)+1;

    // First row
    for(int j=0; j<noPages; j++){
        printf("   %2d  ", chart[0][j]);
    }
    cout<<endl;
    for(int k=0; k<NOH; k++){
        cout<<"-";
    }
    cout<<endl;

    // middle portion
    for(int i=1; i<noFrames+1; i++){
        for(int j=0; j<noPages; j++){
            if(chart[i][j]==-1) printf("|      ", chart[i][j]);
            else printf("|  %2d  ", chart[i][j]);
        }
        cout<<"|"<<endl;
        for(int k=0; k<NOH; k++){
            cout<<"-";
        }
        cout<<endl;
    }


    // last row
    for(int j=0; j<noPages; j++){
        if(chart[noFrames+1][j]==1) cout<<"| hit  ";
        else cout<<"| miss ";
    }
    cout<<"|"<<endl;
    for(int k=0; k<NOH; k++){
        cout<<"-";
    }

    cout<<endl<<endl<<"Average Page Fault : "<<((float)miss/noPages)<<" or 
"<<miss<<"/"<<noPages<<endl<<endl;
}

int main(){
    int noPages, noFrames;

    cout<<"\n\nName : Mohd Adil \nRoll No : 20BCS042";
    cout<<"\n\nEnter No of Pages and Frames : ";
    cin>>noPages>>noFrames;

    int *pages = new int[noPages];
    cout<<"\nEnter the Pages : ";
    for(int i=0; i<noPages; i++) cin>>pages[i];

    LRUPageRepAlgo(pages, noPages, noFrames);
    return 0;
}

// sample input:
// 7 3 1 3 0 3 5 6 3
// 14 4 7 0 1 2 0 3 0 4 2 3 0 3 2 3
```

OUTPUT

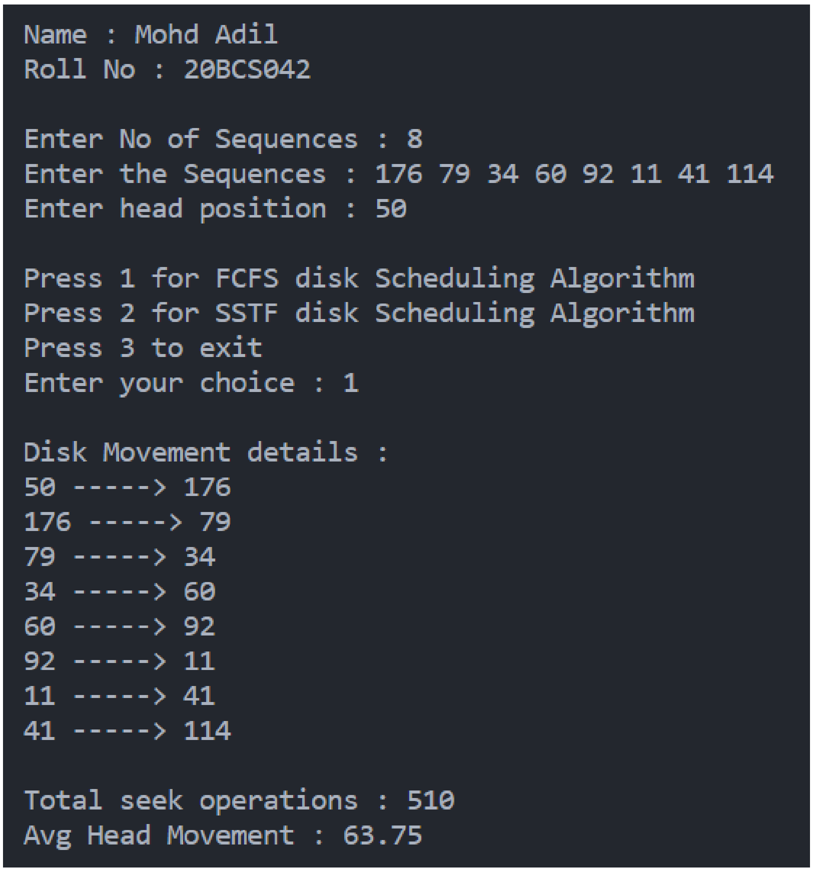


```
Name : Mohd Adil
Roll No : 20BCS042

Enter No of Pages and Frames : 14 4

Enter the Pages : 7 0 1 2 0 3 0 4 2 3 0 3 2 3


Page Fault Details :
    7      0      1      2      0      3      0      4      2      3      0      3      2      3
---------------------------------------------------------------------------------------------------
|   7  |   7  |   7  |   7  |   7  |   3  |   3  |   3  |   3  |   3  |   3  |   3  |   3  |   3  |
---------------------------------------------------------------------------------------------------
|      |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |
---------------------------------------------------------------------------------------------------
|      |      |   1  |   1  |   1  |   1  |   1  |   4  |   4  |   4  |   4  |   4  |   4  |   4  |
---------------------------------------------------------------------------------------------------
|      |      |      |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |
---------------------------------------------------------------------------------------------------
| miss | miss | miss | miss | hit  | miss | hit  | miss | hit  | hit  | hit  | hit  | hit  | hit  |
---------------------------------------------------------------------------------------------------

Average Page Fault : 0.428571 or 6/14
```

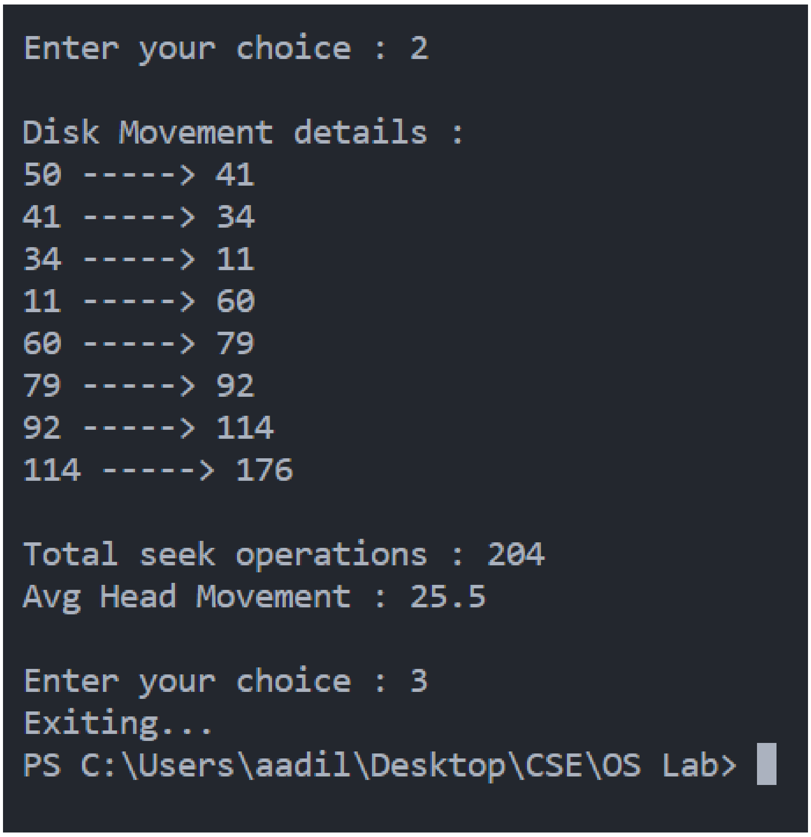
```
#include <iostream>
using namespace std;

void FCFS(int *arr, int n, int head){
    int seq_op=0;
    cout<<"\nDisk Movement details : ";

    for(int i=0; i<n; i++){
        cout<<"\n"<<head<<" -----> "<<arr[i];

        int dist = abs(arr[i]-head);
        seq_op+=dist;
        head=arr[i];

    }

    cout<<"\n\nTotal seek operations : "<<seq_op;
    cout<<"\nAvg Head Movement : "<<float(seq_op)/n<<"\n";

}

int search(int *arr, bool *done, int n, int head){
    int idx=-1, mn = INT16_MAX;
    for(int i=0; i<n; i++){
        if(!done[i] && arr[i]!=head && abs(head-arr[i])<mn){
            mn=abs(head-arr[i]);
            idx=i;
        }
    }
    return idx;
}

void SSTF(int *arr, int n, int head){
    int seq_op=0;
    bool done[n]={false};
    cout<<"\nDisk Movement details : ";

    for(int i=0; i<n; i++){
        int findIdx = search(arr, done, n, head);
        done[findIdx]=true;

        cout<<"\n"<<head<<" -----> "<<arr[findIdx];

        int dist = abs(arr[findIdx]-head);
        seq_op+=dist;
        head=arr[findIdx];

    }

    cout<<"\n\nTotal seek operations : "<<seq_op;
    cout<<"\nAvg Head Movement : "<<float(seq_op)/n<<"\n";
}

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

    int n;
    cout<<"\nEnter No of Sequences : ";
    cin>>n;
    int *arr = new int[n];
    cout<<"Enter the Sequences : ";
    for(int i=0; i<n; i++){
        cin>>arr[i];
    }
    int head;
    cout<<"Enter head position : ";
    cin>>head;

    cout << "\nPress 1 for FCFS disk Scheduling Algorithm";
    cout << "\nPress 2 for SSTF disk Scheduling Algorithm";
    cout << "\nPress 3 to exit";
    while (1){
        cout << "\nEnter your choice : ";
        int ch;
        cin>>ch;

        switch (ch){
        case 1:
            FCFS(arr,n,head);
            break;
        case 2:
            SSTF(arr,n,head);
            break;
        case 3: exit(0);
        default: cout<<"\nEnter a correct choice please";
            break;
        }
    }

    return 0;
```

```
}
```

OUTPUT



```
Name : Mohd Adil
Roll No : 20BCS042

Enter No of Sequences : 8
Enter the Sequences : 176 79 34 60 92 11 41 114
Enter head position : 50

Press 1 for FCFS disk Scheduling Algorithm
Press 2 for SSTF disk Scheduling Algorithm
Press 3 to exit
Enter your choice : 1

Disk Movement details :
50 -----> 176
176 -----> 79
79 -----> 34
34 -----> 60
60 -----> 92
92 -----> 11
11 -----> 41
41 -----> 114

Total seek operations : 510
Avg Head Movement : 63.75
```



```
Enter your choice : 2

Disk Movement details :
50 -----> 41
41 -----> 34
34 -----> 11
11 -----> 60
60 -----> 79
79 -----> 92
92 -----> 114
114 -----> 176

Total seek operations : 204
Avg Head Movement : 25.5

Enter your choice : 3
Exiting...
PS C:\Users\aadil\Desktop\CSE\OS Lab>
```