JAMIA MILLIA ISLAMIA, NEW DELHI

OPERATING SYSTEM LAB

NAME: FAIZAN CHOUDHARY

ROLL NO: 20BCS021

SUBJECT CODE: CEN 493

SEMESTER: 4th

COURSE: B.TECH.(COMPUTER ENGG.)

DEPT: DEPT OF COMPUTER ENGG.

FAIZAN CHOUDHARY

20BCS021

OS LAB

20th January 2022

CODE: (code pasted in this format for readability)

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

struct PQueue
{
    char n[10];
    int pr;
    struct PQueue *next;
};
struct PQueue *front=NULL, *rear=NULL, *p, *ptr;

bool isEmpty () {
    if (front==NULL)
        return true;
    else
        return false;
}

void display () {
    if (isEmpty()==true) {
        cout<<"\nPriority Queue is empty! Nothing to display\n";
        return;
    }

    else {
        p=front;
        cout<<endl;
        while (p->next!=NULL) {
            cout<<"|| "<<p->n<<" | "<<p->pr<<" || --> ";
            p=p->next;
        }
        cout<<"|| "<<p->n<<" | "<<p->pr<<" || --> NULL"<<endl;
    }
}

int totalProcess () {
    int count=1;
    if (isEmpty() == true)
        return 0;
    else {
        p=front;
```

```
        while (p->next!=NULL) {
            count++;
            p=p->next;
        }
    }
    return count;
}

void insertProcess (char* n, int pr) {
    ptr = (struct PQueue *) malloc (sizeof(struct PQueue));
    if (ptr == NULL) {
        cout<<"\nMemory could not be allocated!\n";
        return;
    }
    strcpy(ptr->n, n);
    ptr->pr = pr;
    ptr->next=NULL;
    if (front == NULL || pr < (front->pr)) {
        ptr->next = front;
        front=ptr;
    }
    else {
        p=front;
        while (p->next != NULL && p->next->pr <= pr)
            p=p->next;
        ptr->next = p->next;
        p->next = ptr;
    }
    display();
}

void executeProcess () {
    if (isEmpty() == true)
        cout<<"\nPriority Queue Underflow!"<<endl;
    else {
        p = front;
        cout<<"\nExexcuted process is: || "<<p->n<<" | "<<p->pr<<" ||"<<endl;
        front=front->next;
        delete p;
        display();
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    int ch,pr;
    char n[10];

    while (true) {
        A:
        cout<<"\nMENU:\n1. Insert Process\n2. Execute Process\n3. Total number of 
processes\n4. Display priority queue\n5. Exit\n";
        cin>>ch;
        switch (ch) {
```

```
            case 1: cout<<"\nEnter the process name: ";
                    cin>>n;
                    cout<<"\nEnter the priority: ";
                    cin>>pr;
                    insertProcess(n,pr);
                    break;
            case 2: executeProcess();
                    break;
            case 3: cout<<"\nTotal number of processes in priority queue are: 
"<<totalProcess()<<endl;
                    break;
            case 4: cout<<"\nPriority Queue elements: "<<endl;
                    display();
                    break;
            case 5: exit(0);
            default: cout<<"\nWrong choice! Enter again...\n";
                     goto A;
        }
    }
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
1

Enter the process name: p4

Enter the priority: 5

|| p4 | 5 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
1

Enter the process name: p9

Enter the priority: 2

|| p9 | 2 || --> || p4 | 5 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
1

Enter the process name: p3

Enter the priority: 7

|| p9 | 2 || --> || p4 | 5 || --> || p3 | 7 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
1

Enter the process name: p6

Enter the priority: 5

|| p9 | 2 || --> || p4 | 5 || --> || p6 | 5 || --> || p3 | 7 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
3

Total number of processes in priority queue are: 4
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
2

Exexcuted process is: || p9 | 2 ||

|| p4 | 5 || --> || p6 | 5 || --> || p3 | 7 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
2

Exexcuted process is: || p4 | 5 ||

|| p6 | 5 || --> || p3 | 7 || --> NULL
```

```
MENU:
1. Insert Process
2. Execute Process
3. Total number of processes
4. Display priority queue
5. Exit
5
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

27th January 2022

CODE: (code pasted in this format for readability)

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

struct node
{
    char n[10];
    int burst;
    int arrival;
    int completion;
    int waiting;
    int turnaround;
    int response;
    struct node *next;
};
struct node *front=NULL, *p, *ptr, *temp;

bool isEmpty () {
    if (front==NULL)
        return true;
    else
        return false;
}

void insertProcess (char *pr, int bt, int at) {
    ptr = (struct node *) malloc (sizeof(struct node));
    if (ptr == NULL) {
        cout<<"\nMemory could not be allocated!\n";
        return;
    }

    strcpy(ptr->n, pr);
    ptr->burst = bt;
    ptr->arrival = at;
    ptr->next=NULL;

    if (front == NULL || at < (front->arrival)) {
        ptr->next = front;
        front=ptr;
    }
    else {
        p=front;
```

```
        while (p->next != NULL && p->next->arrival <= at)
            p=p->next;
        ptr->next = p->next;
        p->next = ptr;
    }
}

void FCFS () {
    int time = 0;
    p = front;
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival)
                time++;
        }
        p->response = time - p->arrival;
        time += p->burst;
        p->completion = time;   // completion occurs after burst time ends
        p->turnaround = p->completion - p->arrival;     // tat = ct - at = wt + bt
        p->waiting = p->turnaround - p->burst;          // wt = tat - bt
        p = p->next;
    }
}

void display () {
    double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
    int count = 0;
    p = front;
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time | 
Turnaround Time | Response Time\n";
    cout<<"________________________________________________________________________________
________________________\n\n";
    while (p != NULL) {
        printf("   %s       %2d            %2d                 %2d                %2d
      %2d              %2d\n", p->n, p->burst, p->arrival, p->completion, p->waiting, p-
>turnaround, p->response);

        tot_ct += p->completion;
        tot_wt += p->waiting;
        tot_tat += p->turnaround;
        tot_rt += p->response;

        count++;
        p = p->next;
    }
    cout<<"________________________________________________________________________________
________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) count);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) count);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) count);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) count);
}
```

```
void displayGantt () {
    int time = 0;
    p = front;
    cout<<"\nGantt chart: \n";
    // for printing structure
    while (p != NULL) {
        cout<<"|";
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            time += p->burst;
            cout<<"   |";
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        cout<<p->n;
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        p = p->next;
    }
    cout<<"|"<<endl;
    p = front;
    time = 0;
    // for printing time below each process
    if (time < p->arrival && p->arrival != 0) {
        cout<<time;
        while (time != p->arrival) {
            time++;
        }
        time += p->burst;
        cout<<"   ";
    }
    cout<<p->arrival;
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            if (time < 9)
                cout<<"  "<<time;
            else
                cout<<" "<<time;
            time += p->burst;
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
```

```
        }
        for (int i=0; i< 2*(p->burst)-1; i++)
            cout<<" ";
        if (p->completion < 9)
            cout<<" "<<p->completion;
        else
            cout<<p->completion;
        p = p->next;
    }
    cout<<endl<<endl;
}

void del () {
    p = front;
    front=front->next;
    delete p;
}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nFirst Come First Serve Scheduling Algorithm\n";
    int n;

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    char k[n][10];
    int bt[n], at[n];                   // burst time and arrival time

    cout<<"\nEnter process names: ";
    for (int i=0; i<n; i++)
        cin>>k[i];
    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];

    for (int i=0; i<n; i++)
        insertProcess(k[i],bt[i],at[i]);

    FCFS ();            // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // to display Gantt chart
    del ();             // releasing memory

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

First Come First Serve Scheduling Algorithm

Enter the number of processes: 3

Enter process names: p1 p2 p3

Enter burst time for each process: 2 1 6

Enter arrival time for each process: 0 3 5
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
______________________________________________________________________________________________________

   p1        2            0               2               0              2               0
   p2        1            3               4               0              1               0
   p3        6            5               11              0              6               0

______________________________________________________________________________________________________


Average Completion time: 5.67
Average Waiting time: 0.00
Average Turnaround time: 3.00
Average Response time: 0.00

Gantt chart:
| p1 |  |p2|  |     p3     |
0    2  3  4  5            11
```

```
FAIZAN CHOUDHARY
20BCS021

First Come First Serve Scheduling Algorithm

Enter the number of processes: 5

Enter process names: p1 p2 p3 p4 p5

Enter burst time for each process: 6 2 8 3 4

Enter arrival time for each process: 2 5 1 0 4
```

```
Process | 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: 15.00
Average Waiting time: 8.00
Average Turnaround time: 12.60
Average Response time: 8.00

Gantt chart:
|  p4  |       p3       |      p1     |    p5    | p2 |
0      3                11            17        21   23
```

```
FAIZAN CHOUDHARY
20BCS021

First Come First Serve Scheduling Algorithm

Enter the number of processes: 6

Enter process names: p1 p2 p3 p4 p5 p6

Enter burst time for each process: 3 1 2 1 2 3

Enter arrival time for each process: 5 7 6 1 1 8
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
________________________________________________________________________________________________________

   p4        1           1              2              0              1               0
   p5        2           1              4              1              3               1
   p1        3           5              8              0              3               0
   p3        2           6             10              2              4               2
   p2        1           7             11              3              4               3
   p6        3           8             14              3              6               3
________________________________________________________________________________________________________


Average Completion time: 8.17
Average Waiting time: 1.50
Average Turnaround time: 3.50
Average Response time: 1.50

Gantt chart:
|  |p4| p5 |  |  p1   | p3 |p2|  p6   |
0  1  2    4  5       8   10 11      14
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

3rd February 2022

CODE: (code pasted in this format for readability)

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

struct node
{
    char n[10];
    int burst;
    int arrival;
    int completion;
    int waiting;
    int turnaround;
    int response;
    struct node *next;
};
struct node *front=NULL, *p, *ptr, *temp, *sjf=NULL;

// on the basis of arrival time
void insertProcess (char *pr, int bt, int at) {
    ptr = (struct node *) malloc (sizeof(struct node));
    if (ptr == NULL) {
        cout<<"\nMemory could not be allocated!\n";
        return;
    }

    strcpy(ptr->n, pr);
    ptr->burst = bt;
    ptr->arrival = at;
    ptr->next=NULL;

    if (front == NULL || at < (front->arrival)) {
        ptr->next = front;
        front=ptr;
    }
    else {
        p=front;
        while (p->next != NULL && p->next->arrival <= at)
            p=p->next;
        ptr->next = p->next;
        p->next = ptr;
    }
}
```

```
void displayQ (struct node *a) {
    struct node *t = a;
    cout<<"\nQueue: ";
    while (t != NULL) {
        cout<<"|"<<t->n<<"|"<<t->burst<<"|"<<t->arrival<<"|->";
        t = t->next;
    }
    cout<<endl;
}

// on the basis of burst time
void SJFQueue (struct node **start, struct node **newp) {
    if ((*start) == NULL || (*newp)->burst < (*start)->burst) {
        (*newp)->next = *start;
        *start=*newp;
    }
    else {
        struct node *x = *start;
        while (x->next != NULL && x->next->burst <= (*newp)->burst)
            x=x->next;
        (*newp)->next = x->next;
        x->next = (*newp);
    }
}

void SJF () {
    p = front;
    struct node *r = sjf;
    int current = p->arrival;       // time which begins from the process that arrived
earliest
    struct node *q = NULL;                          // sjf/burst time queue pointer

    while (p != NULL) {
        int t = 0;                  // time for executing all process in queue
        while (p != NULL && p->arrival <= current) {
            temp = p;           // dequeueing from ready queue
            p = p->next;
            temp->next = NULL;
            t += temp->burst;
            SJFQueue (&q, &temp);
        }

        int exTime = q->arrival;        // execution time of sjf queue
        while (q != NULL && q->arrival >= exTime) {
            if (p == NULL) {            // when ready queue is empty.
                if (sjf == NULL)
                    sjf = q;
                else
                    while (r->next != NULL)
                        r = r->next;
                    r->next = q;
                    break;
            }
```

```
            struct node *n = q;         // dequeueing from burst time queue
            q = q->next;
            n->next = NULL;
            if (sjf == NULL) {
                sjf = n;
                r = sjf;
            }
            else {
                while (r->next != NULL)
                    r = r->next;
                r->next = n;
            }
            exTime += n->burst;
        }
        if (p != NULL)
            if (current + t < p->arrival)
                current = p->arrival;       // updating current process' arrival time
            else
                current += t;
    }
}

void display () {
    double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
    int count = 0, time = 0;
    p = front;
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"________________________________________________________________________________
________________________\n\n";
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival)
                time++;
        }
        p->response = time - p->arrival;
        time += p->burst;
        p->completion = time;    // completion occurs after burst time ends
        p->turnaround = p->completion - p->arrival;     // tat = ct - at = wt + bt
        p->waiting = p->turnaround - p->burst;          // wt = tat - bt

        printf("   %s        %2d             %2d                 %2d                %2d
      %2d              %2d\n", p->n, p->burst, p->arrival, p->completion, p->waiting, p-
>turnaround, p->response);

        tot_ct += p->completion;
        tot_wt += p->waiting;
        tot_tat += p->turnaround;
        tot_rt += p->response;

        count++;
        p = p->next;
    }
```

```
    cout<<"____________________________________________________________________________________________________________________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) count);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) count);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) count);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) count);
}

void displayGantt () {
    int time = 0;
    p = front;
    cout<<"\nGantt chart: \n";
    // for printing structure
    while (p != NULL) {
        cout<<"|";
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            time += p->burst;
            cout<<"   |";
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        cout<<p->n;
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        p = p->next;
    }
    cout<<"|"<<endl;
    p = front;
    time = 0;
    // for printing time below each process
    if (time < p->arrival && p->arrival != 0) {
        cout<<time;
        while (time != p->arrival) {
            time++;
        }
        time += p->burst;
        cout<<"  ";
    }
    cout<<p->arrival;
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            if (time < 9)
```

```
                cout<<" "<<time;
            else
                cout<<time;
            time += p->burst;
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i< 2*(p->burst)-1; i++)
            cout<<" ";
        if (p->completion < 9)
            cout<<"  "<<p->completion;
        else
            cout<<p->completion;
        p = p->next;
    }
    cout<<endl<<endl;
}

void del () {
    p = front;
    front=front->next;
    delete p;
}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nShortest Job First (Non-Preemptive) Scheduling Algorithm\n";
    int n;

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    char k[n][10];
    int bt[n], at[n];                   // burst time and arrival time

    cout<<"\nEnter process names: ";
    for (int i=0; i<n; i++)
        cin>>k[i];
    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];

    for (int i=0; i<n; i++)
        insertProcess(k[i],bt[i],at[i]);

    SJF ();             // logic for calculating various times
    display ();          // displaying calculated values of time
    displayGantt ();     // to display Gantt chart
    del ();              // releasing memory
```

```
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Shortest Job First (Non-Preemptive) Scheduling Algorithm

Enter the number of processes: 5

Enter process names: p1 p2 p3 p4 p5

Enter burst time for each process: 3 7 4 2 2

Enter arrival time for each process: 0 6 6 6 5
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
--------------------------------------------------------------------------------------------------------

   p1        3            0             3             0              3              0
   p5        2            5             7             0              2              0
   p4        2            6             9             1              3              1
   p3        4            6            13             3              7              3
   p2        7            6            20             7             14              7

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


Average Completion time: 10.40
Average Waiting time: 2.20
Average Turnaround time: 5.80
Average Response time: 2.20

Gantt chart:
|  p1  |  | p5 | p4 |  p3   |      p2      |
0      3   5    7   9      13             20
```

```
FAIZAN CHOUDHARY
20BCS021

Shortest Job First (Non-Preemptive) Scheduling Algorithm

Enter the number of processes: 5

Enter process names: p1 p2 p3 p4 p5

Enter burst time for each process: 6 2 8 3 4

Enter arrival time for each process: 2 5 1 0 4
```

```
Process | 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: 12.20
Average Waiting time: 5.20
Average Turnaround time: 9.80
Average Response time: 5.20

Gantt chart:
|  p4  |     p1     | p2 |   p5   |       p3       |
0      3           9   11       15               23
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

10th February 2022

CODE: (code pasted in this format for readability)

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

struct process
{
    char n[10];
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[100];
int n;                          // no of processes input from user

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[100];
int count = 0;                      // to count number of indexed processes for Gantt chart

int remaining[100];             // to store remaining burst time for each process
bool completed[100];            // to store if the process is completed or not
int current_time = 0;
int num = 0;                    // to store the number of processes completed

double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;

void SRTF () {
    int temp=-1;
    while (num != n) {
        int index = -1;                 // to store the index of the process with minimum
burst time
        int mn = INT_MAX;               // to store minimum burst time from all processes
        for (int i=0; i<n; i++) {
```

```
            if (pr[i].arrival <= current_time && completed[i] == false) {
                // if there exists a process with burst time lower than mn, update mn
                if (remaining[i] < mn) {
                    mn = remaining[i];
                    index = i;
                }
                // if two processes have the same burst time, priority given to the one
arriving first
                if (remaining[i] == mn) {
                    if (pr[i].arrival < pr[index].arrival) {
                        mn = remaining[i];
                        index = i;
                    }
                }
            }
        }

        // if it is a valid index, processes present in ready queue
        if (index != -1) {
            // if the burst time matches with the remaining burst time, the process starts
executing for the first time
            if (remaining[index] == pr[index].burst)
                pr[index].start = current_time;

            // updating remaining burst time with a time quantum of 1 unit, and increasing
current time
            remaining[index] -= 1;
            current_time++;

            if (remaining[index] == 0) {
                pr[index].completion = current_time;
                pr[index].turnaround = pr[index].completion - pr[index].arrival;
                pr[index].waiting = pr[index].turnaround - pr[index].burst;
                pr[index].response = pr[index].start - pr[index].arrival;

                tot_tat += pr[index].turnaround;
                tot_wt += pr[index].waiting;
                tot_ct += pr[index].completion;
                tot_rt += pr[index].response;

                completed[index] = true;
                num++;
            }

            // printing Gantt chart
            // cout<<"|"<<current_time-1<<"  "<<pr[index].n<<"  "<<current_time<<"|  ";
            g[count].idx = index;
            g[count].start = current_time - 1;
            g[count].end = current_time;
            count++;

        }
        // if no process in ready queue, increase current_time
        else
```

```
            current_time++;
    }
    g[count].end = current_time;
}

void display () {
    int time = 0;
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"________________________________________________________________________________
________________________\n\n";

    for (int i=0; i<n; i++) {
        printf("   %s        %2d             %2d                  %2d                %2d
      %2d                %2d\n", pr[i].n, pr[i].burst, pr[i].arrival, pr[i].completion,
pr[i].waiting, pr[i].turnaround, pr[i].response);
    }

    cout<<"________________________________________________________________________________
________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    for (int i=0; i<count; i++) {
        cout<<"|";
        for (int j=-1; j <= (g[i].start-g[i].end); j++)
            cout<<" ";
        cout<<pr[g[i].idx].n;
        for (int j=-1; j <= (g[i].start-g[i].end); j++)
            cout<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<count; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"   ";
        else if (g[i].start <= 9)
            cout<<g[i].start<<"    ";
    }
    cout<<g[i].end<<endl;

}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nShortest Job First (Preemptive) / Shortest Remaining Time First Scheduling
Algorithm\n";
```

```
    cout<<"\nEnter the number of processes: ";
    cin>>n;
    char k[n][10];
    int bt[n], at[n];                    // burst time and arrival time

    cout<<"\nEnter process names: ";
    for (int i=0; i<n; i++)
        cin>>k[i];
    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];

    for (int i=0; i<n; i++) {
        strcpy(pr[i].n, k[i]);
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        remaining[i] = pr[i].burst;
    }

    SRTF ();            // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Shortest Job First (Preemptive) / Shortest Remaining Time First Scheduling Algorithm

Enter the number of processes: 5

Enter process names: p1 p2 p3 p4 p5

Enter burst time for each process: 6 2 8 3 4

Enter arrival time for each process: 2 5 1 0 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

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


Average Completion time: 11.60
Average Waiting time: 4.60
Average Turnaround time: 9.20
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
```

```
FAIZAN CHOUDHARY
20BCS021

Shortest Job First (Preemptive) / Shortest Remaining Time First Scheduling Algorithm

Enter the number of processes: 5

Enter process names: p1 p2 p3 p4 p5

Enter burst time for each process: 3 7 4 2 2

Enter arrival time for each process: 0 6 6 6 5
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
---------------------------------------------------------------------------------------------------------

   p1        3            0                3              0              3                0
   p2        7            6               20              7             14                7
   p3        4            6               13              3              7                3
   p4        2            6                9              1              3                1
   p5        2            5                7              0              2                0

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


Average Completion time: 10.40
Average Waiting time: 2.20
Average Turnaround time: 5.80
Average Response time: 2.20

Gantt chart:
| p1 | p1 | p1 | p5 | p5 | p4 | p4 | p3 | p3 | p3 | p3 | p2 | p2 | p2 | p2 | p2 | p2 | p2 |
0    1    2    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20
```

```
FAIZAN CHOUDHARY
20BCS021

Shortest Job First (Preemptive) / Shortest Remaining Time First Scheduling Algorithm

Enter the number of processes: 4

Enter process names: p1 p2 p3 p4

Enter burst time for each process: 7 4 1 4

Enter arrival time for each process: 0 2 4 5
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
------------------------------------------------------------------------------------------------------

   p1        7            0               16              9               16               0
   p2        4            2                7              1                5               0
   p3        1            4                5              0                1               0
   p4        4            5               11              2                6               2
------------------------------------------------------------------------------------------------------


Average Completion time: 9.75
Average Waiting time: 3.00
Average Turnaround time: 7.00
Average Response time: 0.50

Gantt chart:
| p1 | p1 | p2 | p2 | p3 | p2 | p2 | p4 | p4 | p4 | p4 | p1 | p1 | p1 | p1 | p1 |
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

17th February 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <string.h>
#include <algorithm>
using namespace std;
const int SIZE = 50;

struct process
{
    int pid;
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[SIZE];
int n;

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[SIZE];
int cnt=0;                          // to count number of indexed processes for Gantt
chart

// ready queue (circular queue) FIFO
int ready_queue[SIZE];
int front=-1, rear=-1;

int current_time=0, time_quantum;
int remaining[SIZE];                    // to store remaining burst time for each process
int temp[SIZE];                         // to store remaining burst times for Gantt chart
bool completed[SIZE] = {false};        // to store if the process is completed or not
int idx;
int num = 0;                            // to store the number of processes completed
double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
```

```
// comparing wrt arrival time
bool compare1 (process &p1, process &p2) {
    return p1.arrival < p2.arrival;
}

// comparing wrt pid
bool compare2 (process &p1, process &p2) {
    return p1.pid < p2.pid;
}

// to insert a process in the ready queue
void insertProcess (int l) {
    if (front == -1)
        front = 0;
    rear = (rear + 1) % SIZE;
    ready_queue[rear] = l;
}

// deleting from ready queue
void executeProcess () {
    if (front == -1)
        return;
    if (front == rear)
        front=rear=-1;
    else
        front = (front + 1) % SIZE;
}

void RR () {
    // sorting wrt arrival times
    sort (pr,pr+n,compare1);
    // inserting first process
    insertProcess(0);
    completed[0] = true;

    // loop until the num of processes executed is equal to no of processes input by user
    while (num != n) {
        // dispatching the process at the front of ready queue
        idx = ready_queue[front];
        executeProcess();

        // if the remaining burst time for a process is equal to the current process at
that idx, update current_time and start time of process
        if (remaining[idx] == pr[idx].burst) {
            pr[idx].start = max (current_time, pr[idx].arrival);
            current_time = pr[idx].start;
        }

        // if the burst time remaining for a process is greater than the time quantum
        if (remaining[idx] > time_quantum) {
            temp[idx] = remaining[idx];
            remaining[idx] -= time_quantum;
            current_time += time_quantum;
        }
```

```
        else {
            // if the process has remaining burst time less than time quantum
            current_time += remaining[idx];
            temp[idx] = remaining[idx];
            remaining[idx] = 0;
            // updating no of processes completed
            num++;

            pr[idx].completion = current_time;
            pr[idx].turnaround = pr[idx].completion - pr[idx].arrival;
            pr[idx].waiting = pr[idx].turnaround - pr[idx].burst;
            pr[idx].response = pr[idx].start - pr[idx].arrival;

            tot_tat += pr[idx].turnaround;
            tot_wt += pr[idx].waiting;
            tot_ct += pr[idx].completion;
            tot_rt += pr[idx].response;
        }

        for (int i=1; i<n; i++) {
            if (remaining[i] > 0 && pr[i].arrival <= current_time && completed[i] ==
false) {
                insertProcess(i);
                completed[i] = true;
            }
        }

        if (remaining[idx] > 0)
            insertProcess(idx);

        // if queue is empty
        if (front == -1) {
            for (int i=1; i<n; i++) {
                if (remaining[i] > 0) {
                    insertProcess(i);
                    completed[i] = true;
                    break;
                }
            }
        }
        // for Gantt chart
        g[cnt].idx = idx;
        if (current_time - time_quantum < 0)
            g[cnt].start = 0;
        else if (temp[idx] < time_quantum)
            g[cnt].start = current_time - time_quantum + 1;
        else
            g[cnt].start = current_time - time_quantum;
        g[cnt].end = current_time+1;
        cnt++;
    }
    g[cnt].end = current_time;
}
```

```
void display () {
    int time = 0;
    sort(pr,pr+n,compare2);
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"_______________________________________________________________________________
________________________\n\n";

    for (int i=0; i<n; i++) {
        printf("   P%d         %2d             %2d                  %2d                %2d
        %2d              %2d\n", pr[i].pid, pr[i].burst, pr[i].arrival, pr[i].completion,
pr[i].waiting, pr[i].turnaround, pr[i].response);
    }

    cout<<"_______________________________________________________________________________
________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    for (int i=0; i<cnt; i++) {
        cout<<"| ";
        cout<<"P"<<pr[g[i].idx].pid<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<cnt; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"   ";
        else if (g[i].start <= 9)
            cout<<g[i].start<<"    ";
    }
    cout<<g[i].end<<endl;
}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nRound Robin Scheduling Algorithm\n";

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    int bt[n], at[n];                    // burst time and arrival time

    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
```

```
        cin>>at[i];

    for (int i=0; i<n; i++) {
        // pr[i].pid = k[i];
        pr[i].pid = i+1;
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        remaining[i] = pr[i].burst;
    }

    cout<<"\nEnter the time quantum: ";
    cin>>time_quantum;

    RR ();                  // logic for calculating various times
    display ();             // displaying calculated values of time
    displayGantt ();        // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 5

Enter burst time for each process: 5 3 1 2 3

Enter arrival time for each process: 0 1 2 3 4

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
------------------------------------------------------------------------------------------------------
   P2          3              1              12               8               11                1
   P3          1              2               5               2                3                2
   P4          2              3               9               4                6                4
   P5          3              4              14               7               10                5

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


Average Completion time: 10.60
Average Waiting time: 5.80
Average Turnaround time: 8.60
Average Response time: 2.40

Gantt chart:
| P1 | P2 | P3 | P1 | P4 | P5 | P2 | P1 | P5 |
0    2    4    5    7    9    11   12   13   14
```

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 6

Enter burst time for each process: 4 5 2 1 6 3

Enter arrival time for each process: 0 1 2 3 4 6

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
-----------------------------------------------------------------------------------------------------------

   P1        4            0                8                4               8                0
   P2        5            1               18               12              17                1
   P3        2            2                6                2               4                2
   P4        1            3                9                5               6                5
   P5        6            4               21               11              17                5
   P6        3            6               19               10              13                7
-----------------------------------------------------------------------------------------------------------


Average Completion time: 13.50
Average Waiting time: 7.33
Average Turnaround time: 10.83
Average Response time: 3.33

Gantt chart:
| P1 | P2 | P3 | P1 | P4 | P5 | P2 | P6 | P5 | P2 | P6 | P5 |
0    2    4    6    8    9    11   13   15   17   18   19   21
```

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 3

Enter burst time for each process: 4 3 5

Enter arrival time for each process: 0 0 0

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
-----------------------------------------------------------------------------------------------------------

   P1        4            0                8                4               8                0
   P2        3            0                9                6               9                2
   P3        5            0               12                7              12                4
-----------------------------------------------------------------------------------------------------------


Average Completion time: 9.67
Average Waiting time: 5.67
Average Turnaround time: 9.67
Average Response time: 2.00

Gantt chart:
| P1 | P2 | P3 | P1 | P2 | P3 | P3 |
0    2    4    6    8    9    11   12
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

24th February 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <string.h>
#include <algorithm>
#include <limits.h>
using namespace std;
const int SIZE = 50;

struct process
{
    int pid;
    int priority;
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[SIZE];
int n;

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[SIZE];
int cnt=0;                      // to count number of indexed processes for Gantt
chart

int current_time = 0;
bool completed[SIZE] = {false};         // to store if the process is completed or not
int idx = -1;
int num = 0;                    // to store the number of processes completed
double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;

// comparing wrt arrival time
bool compare1 (process &p1, process &p2) {
    return p1.arrival < p2.arrival;
}
```

```
// comparing wrt pid
bool compare2 (process &p1, process &p2) {
    return p1.pid < p2.pid;
}

void PriorityScheduling () {
    sort(pr,pr+n,compare1);
    while (num != n) {
        int idx = -1;                   // stores the index of process with highest 
priority
        int mn = INT_MAX;               // stores the highest priority (lowest number)
        for (int i=0; i<n; i++) {
            if (pr[i].arrival <= current_time && completed[i] == false) {
                // if a process has greater priority
                if (pr[i].priority < mn) {
                    mn = pr[i].priority;
                    idx = i;
                }
                // if a process has priority equal to max priority (min number) so far
                if (pr[i].priority == mn) {
                    // we chose the one that arrives first
                    if (pr[i].arrival < pr[idx].arrival) {
                        mn = pr[i].priority;
                        idx = i;
                    }
                }
            }
        }
        // if there exists a process
        if (idx != -1) {
            pr[idx].start = current_time;
            pr[idx].completion = pr[idx].start + pr[idx].burst;
            pr[idx].turnaround = pr[idx].completion - pr[idx].arrival;
            pr[idx].waiting = pr[idx].turnaround - pr[idx].burst;
            pr[idx].response = pr[idx].start - pr[idx].arrival;

            tot_tat += pr[idx].turnaround;
            tot_wt += pr[idx].waiting;
            tot_ct += pr[idx].completion;
            tot_rt += pr[idx].response;

            // since Non Preemptive
            completed[idx] = true;
            num++;
            current_time = pr[idx].completion;

        }

        else
            current_time++;

        // for Gantt chart
        g[cnt].idx = idx;
```

```
        g[cnt].start = pr[idx].start;
        g[cnt].end = pr[idx].completion;
        cnt++;
    }
    g[cnt].end = current_time;
}

void display () {
    int time = 0;
    // sort(pr,pr+n,compare2);
    process k[SIZE];
    for (int i=0; i<n; i++)
        k[i] = pr[i];
    sort(k,k+n,compare2);

    cout<<"\n\nProcess | Priority | Burst Time | Arrival Time | Completion Time | Waiting
Time | Turnaround Time | Response Time\n";
    cout<<"________________________________________________________________________________
__________________________________\n\n";

    for (int i=0; i<n; i++) {
        printf("    P%d         %2d          %2d             %2d                  %2d
    %2d              %2d\n", k[i].pid, k[i].priority, k[i].burst,
k[i].arrival, k[i].completion, k[i].waiting, k[i].turnaround, k[i].response);
    }

    cout<<"________________________________________________________________________________
__________________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    for (int i=0; i<cnt; i++) {
        cout<<"| ";
        cout<<"P"<<pr[g[i].idx].pid<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<cnt; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"   ";
        else if (g[i].start <= 9)
            cout<<g[i].start<<"    ";
    }
    cout<<g[i].end<<endl;
}

int main () {
```

```
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nNon-Preemptive Priority Scheduling Algorithm\n";

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    int *bt = new int[n];
    int *at = new int[n];                       // burst time and arrival time
    int *p = new int[n];                        // priority

    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];
    cout<<"\nEnter the priority for each process: ";
    for (int i=0; i<n; i++)
        cin>>p[i];

    for (int i=0; i<n; i++) {
        // pr[i].pid = k[i];
        pr[i].pid = i+1;
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        pr[i].priority = p[i];
    }


    PriorityScheduling ();                  // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Non-Preemptive Priority Scheduling Algorithm

Enter the number of processes: 7

Enter burst time for each process: 3 5 4 2 9 4 10

Enter arrival time for each process: 0 2 1 4 6 5 7

Enter the priority for each process: 2 6 3 5 7 4 10
```

Process	Priority	Burst Time	Arrival Time	Completion Time	Waiting Time	Turnaround Time	Response Time
P1	2	3	0	3	0	3	0
P2	6	5	2	18	11	16	11
P3	3	4	1	7	2	6	2
P4	5	2	4	13	7	9	7
P5	7	9	6	27	12	21	12
P6	4	4	5	11	2	6	2
P7	10	10	7	37	20	30	20

```
Average Completion time: 16.57
Average Waiting time: 7.71
Average Turnaround time: 13.00
Average Response time: 7.71

Gantt chart:
| P1 | P3 | P6 | P4 | P2 | P5 | P7 |
0    3    7    11   13   18   27   37
```

```
FAIZAN CHOUDHARY
20BCS021

Non-Preemptive Priority Scheduling Algorithm

Enter the number of processes: 5

Enter burst time for each process: 11 28 2 10 16

Enter arrival time for each process: 0 5 12 2 9

Enter the priority for each process: 2 0 3 1 4
```

Process	Priority	Burst Time	Arrival Time	Completion Time	Waiting Time	Turnaround Time	Response Time
P1	2	11	0	11	0	11	0
P2	0	28	5	39	6	34	6
P3	3	2	12	51	37	39	37
P4	1	10	2	49	37	47	37
P5	4	16	9	67	42	58	42

```
Average Completion time: 43.40
Average Waiting time: 24.40
Average Turnaround time: 37.80
Average Response time: 24.40

Gantt chart:
| P1 | P2 | P4 | P3 | P5 |
0    11   39   49   51   67
```


Go to Index

FAIZAN CHOUDHARY

20BCS021

OS LAB

10th March 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <string.h>
#include <algorithm>
#include <limits.h>
using namespace std;
const int SIZE = 50;

struct process
{
    int pid;
    int priority;
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[SIZE];
int n;

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[SIZE];
int cnt=0;                          // to count number of indexed processes for Gantt
chart

int remaining[100];         // to store remaining burst time for each process
int current_time = 0;
bool completed[SIZE] = {false};         // to store if the process is completed or not
int idx = -1;
int num = 0;                            // to store the number of processes completed

double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;

// comparing wrt arrival time
bool compare1 (process &p1, process &p2) {
```

```
    return p1.arrival < p2.arrival;
}

// comparing wrt pid
bool compare2 (process &p1, process &p2) {
    return p1.pid < p2.pid;
}

void PrePriorityScheduling () {
    // sort(pr,pr+n,compare1);
    while (num != n) {
        int idx = -1;                   // stores the index of process with highest 
priority
        int mn = INT_MAX;               // stores the highest priority (lowest number)
        for (int i=0; i<n; i++) {
            if (pr[i].arrival <= current_time && completed[i] == false) {
                // if a process has greater priority
                if (pr[i].priority < mn) {
                    mn = pr[i].priority;
                    idx = i;
                }
                // if a process has priority equal to max priority (min number) so far
                if (pr[i].priority == mn) {
                    // we chose the one that arrives first
                    if (pr[i].arrival < pr[idx].arrival) {
                        mn = pr[i].priority;
                        idx = i;
                    }
                }
            }
        }
        // if there exists a process
        if (idx != -1) {
            if (remaining[idx] == pr[idx].burst)
                pr[idx].start = current_time;
            remaining[idx] -= 1;
            current_time++;

            if (remaining[idx] == 0) {
                pr[idx].start = current_time;
                pr[idx].completion = pr[idx].start + pr[idx].burst;
                pr[idx].turnaround = pr[idx].completion - pr[idx].arrival;
                pr[idx].waiting = pr[idx].turnaround - pr[idx].burst;
                pr[idx].response = pr[idx].start - pr[idx].arrival;

                tot_tat += pr[idx].turnaround;
                tot_wt += pr[idx].waiting;
                tot_ct += pr[idx].completion;
                tot_rt += pr[idx].response;

                completed[idx] = true;
                num++;
            }
        }
```

```
        else
            current_time++;

        // for Gantt chart
        g[cnt].idx = idx;
        g[cnt].start = current_time - 1;
        g[cnt].end = current_time;
        cnt++;
    }
    g[cnt].end = current_time;
}

void display () {
    int time = 0;
    // sort(pr,pr+n,compare2);
    process k[SIZE];
    for (int i=0; i<n; i++)
        k[i] = pr[i];
    sort(k,k+n,compare2);

    cout<<"\n\nProcess | Priority | Burst Time | Arrival Time | Completion Time | Waiting
Time | Turnaround Time | Response Time\n";
    cout<<"________________________________________________________________________________
___________________________________\n\n";

    for (int i=0; i<n; i++) {
        printf("    P%d        %2d          %2d              %2d                  %2d
    %2d              %2d              %2d\n", k[i].pid, k[i].priority, k[i].burst,
k[i].arrival, k[i].completion, k[i].waiting, k[i].turnaround, k[i].response);
    }

    cout<<"________________________________________________________________________________
___________________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    for (int i=0; i<cnt; i++) {
        cout<<"| ";
        cout<<"P"<<pr[g[i].idx].pid<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<cnt; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"    ";
        else if (g[i].start <= 9)
```

```
            cout<<g[i].start<<"    ";
    }
    cout<<g[i].end<<endl;
}

int main () {

    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nPreemptive Priority Scheduling Algorithm\n";

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    int *bt = new int[n];
    int *at = new int[n];                    // burst time and arrival time
    int *p = new int[n];                     // priority

    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];
    cout<<"\nEnter the priority for each process: ";
    for (int i=0; i<n; i++)
        cin>>p[i];

    for (int i=0; i<n; i++) {
        // pr[i].pid = k[i];
        pr[i].pid = i+1;
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        pr[i].priority = p[i];
        remaining[i] = pr[i].burst;
    }

    PrePriorityScheduling ();              // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Preemptive Priority Scheduling Algorithm

Enter the number of processes: 6

Enter burst time for each process: 4 5 6 1 2 3

Enter arrival time for each process: 1 2 3 0 4 5

Enter the priority for each process: 5 2 6 4 7 8
```

```
Process | Priority | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
------------------------------------------------------------------------------------------------------------------

   P1        5          4            1              14              9              13               9
   P2        2          5            2              12              5              10               5
   P3        6          6            3              22             13              19              13
   P4        4          1            0               2              1               2               1
   P5        7          2            4              20             14              16              14
   P6        8          3            5              24             16              19              16

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


Average Completion time: 15.67
Average Waiting time: 9.67
Average Turnaround time: 13.17
Average Response time: 9.67

Gantt chart:
| P4 | P1 | P2 | P2 | P2 | P2 | P2 | P1 | P1 | P1 | P3 | P3 | P3 | P3 | P3 | P3 | P5 | P5 | P6 | P6 | P6 |
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20   21
```

```
FAIZAN CHOUDHARY
20BCS021

Preemptive Priority Scheduling Algorithm

Enter the number of processes: 7

Enter burst time for each process: 4 2 3 5 1 4 6

Enter arrival time for each process: 0 1 2 3 4 5 6

Enter the priority for each process: 2 4 6 10 8 12 9
```

```
Process | Priority | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
------------------------------------------------------------------------------------------------------------------

   P1        2          4            0              8               4              8                4
   P2        4          2            1              8               5              7                5
   P3        6          3            2             12               7             10                7
   P4       10          5            3             26              18             23               18
   P5        8          1            4             11               6              7                6
   P6       12          4            5             29              20             24               20
   P7        9          6            6             22              10             16               10

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


Average Completion time: 16.57
Average Waiting time: 10.00
Average Turnaround time: 13.57
Average Response time: 10.00

Gantt chart:
| P1 | P1 | P1 | P1 | P2 | P2 | P3 | P3 | P3 | P5 | P7 | P7 | P7 | P7 | P7 | P7 | P4 | P4 | P4 | P4 | P4 | P6 | P6 | P6 | P6 |
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

10th March 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <algorithm>
#include <limits.h>
using namespace std;
const int SIZE = 50;

struct process
{
    int pid;
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[SIZE];
int n;

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[SIZE];
int cnt=0;                          // to count number of indexed processes for Gantt
chart

int current_time = 0;
bool completed[SIZE] = {false};        // to store if the process is completed or not
int idx = -1;
int num = 0;                           // to store the number of processes completed
int tot_bt = 0;
double mx = -1.0;                       // to store the max response ratio

double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
double hrrn[SIZE];                      // to store the response ratios
double RR;

// comparing wrt arrival time
```

```
bool compare1 (process &p1, process &p2) {
    return p1.arrival < p2.arrival;
}

// comparing wrt pid
bool compare2 (process &p1, process &p2) {
    return p1.pid < p2.pid;
}

void HRRN () {
    sort(pr,pr+n,compare1);
    if (current_time < pr[0].arrival)
        current_time = pr[0].arrival;
    while (num < n) {
        for (int i=0; i<n; i++) {
            RR = ((double)(current_time - pr[i].arrival + pr[i].burst)) / ((double)
pr[i].burst);

            if (RR == mx) {
                if (pr[i].arrival < pr[idx].arrival)
                    idx = i;
            }

            if (RR > mx) {
                if (pr[i].arrival <= current_time && completed[i] == false) {
                    mx = RR;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            pr[idx].start = current_time;
            pr[idx].completion = pr[idx].start + pr[idx].burst;
            pr[idx].turnaround = pr[idx].completion - pr[idx].arrival;
            pr[idx].waiting = pr[idx].turnaround - pr[idx].burst;
            pr[idx].response = pr[idx].start - pr[idx].arrival;

            tot_tat += pr[idx].turnaround;
            tot_wt += pr[idx].waiting;
            tot_ct += pr[idx].completion;
            tot_rt += pr[idx].response;

            completed[idx] = true;
            num++;
            current_time = pr[idx].completion;
        }

        else
            current_time++;

        // for Gantt chart
        g[cnt].idx = idx;
        g[cnt].start = pr[idx].start;
```

```
        g[cnt].end = pr[idx].completion;
        cnt++;
    }
    g[cnt].end = current_time;
}

void display () {
    int time = 0;
    // sort(pr,pr+n,compare2);
    process k[SIZE];
    for (int i=0; i<n; i++)
        k[i] = pr[i];
    sort(k,k+n,compare2);

    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"____________________________________________________________________________________________________________________\n\n";

    for (int i=0; i<n; i++) {
        printf("    P%d        %2d              %2d                   %2d                 %2d
        %2d             %2d\n", k[i].pid, k[i].burst, k[i].arrival, k[i].completion,
k[i].waiting, k[i].turnaround, k[i].response);
    }

    cout<<"____________________________________________________________________________________________________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    // if (time < pr[g[0].idx].arrival)
    //     time = pr[g[0].idx].arrival;
    for (int i=0; i<cnt; i++) {
        cout<<"| ";
        cout<<"P"<<pr[g[i].idx].pid<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<cnt; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"    ";
        else if (g[i].start <= 9)
            cout<<g[i].start<<"     ";
    }
    cout<<g[i].end<<endl;
}
```

```cpp
int main () {

    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nHighest Response Ratio Next Scheduling Algorithm\n";

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

    int *bt = new int[n];
    int *at = new int[n];                   // burst time and arrival time

    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];

    for (int i=0; i<n; i++) {
        pr[i].pid = i+1;
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        // bt_copy[i] = bt[i];
        tot_bt += bt[i];
    }

    HRRN ();              // logic for calculating various times
    display ();           // displaying calculated values of time
    displayGantt ();      // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Highest Response Ratio Next Scheduling Algorithm

Enter the number of processes: 5

Enter burst time for each process: 3 6 8 4 5

Enter arrival time for each process: 1 3 5 7 8
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
-----------------------------------------------------------------------------------------------------

   P1        3             1               4              0                3               0
   P2        6             3              10              1                7               1
   P3        8             5              27             14               22              14
   P4        4             7              14              3                7               3
   P5        5             8              19              6               11               6

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


Average Completion time: 14.80
Average Waiting time: 4.80
Average Turnaround time: 10.00
Average Response time: 4.80

Gantt chart:
| P1 | P2 | P4 | P5 | P3 |
1    4    10   14   19   27
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

24th March 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
// array to store process indices for each block index
int allocation_block[100] = {-1};
int totIntFrag=0, totExtFrag=0;
// temp array to store size of blocks for display
int temp[100];
// array to store internal fragmentation of each block
int intFrag[100] = {0};
// array to store the occupancy status of each block
bool occupied_block[100] = {false};
// counter to keep track of allocated processes
int counter=0;


void display (int *s_b, int *s_p) {
    cout<<"\nAfter allocation:\n";
    cout<<"\nBLOCK ID\tBLOCK SIZE\tPROCESS\t\tINTERNAL FRAGMENTATION\n";
    for (int i=0; i<n; i++) {
        cout<<i+1<<"\t\t  "<<temp[i]<<"\t\t";
        // if block is actually allocated a process
        if (occupied_block[i] == false)
            cout<<"--\t\t\t--";
        else if (allocation_block[i] != -1) {
            cout<<s_p[allocation_block[i]]<<" (P"<<allocation_block[i] + 1<<")\t\t";
            cout<<intFrag[i];
        }
        cout<<endl;
    }
    cout<<"\nTotal Internal Fragmentation: "<<totIntFrag;
    cout<<"\nTotal External Fragmentation: "<<totExtFrag<<endl<<endl;

}

void firstFit (int *s_b, int *s_p) {
    for (int i=0; i<n; i++)
        temp[i] = s_b[i];

    for (int i=0; i<no; i++) {
        for (int j=0; j<n; j++) {
```

```
            if (s_b[j] >= s_p[i]) {
                counter++;

                allocation_block[j] = i;
                occupied_block[j] = true;

                intFrag[j] = s_b[j] - s_p[i];
                // subtracting the value of memory that has been allocated
                s_b[j] -= s_p[i];
                break;
            }
        }
    }
    for (int i=0; i<n; i++) {
        totIntFrag += intFrag[i];
        if (occupied_block[i] == false && counter < no)
            totExtFrag += s_b[i];
    }

}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nFirst Fit Memory Management\n";

    cout<<"\nEnter number of memory blocks: ";
    cin>>n;

    int size_blocks[100];
    cout<<"\nEnter the size of each block:\n";
    for (int i=0; i<n; i++)
        cin>>size_blocks[i];

    cout<<"\nEnter number of processes: ";
    cin>>no;

    int size_processes[100];
    cout<<"\nEnter the size of each process:\n";
    for (int i=0; i<no; i++)
        cin>>size_processes[i];

    firstFit (size_blocks, size_processes);
    display (size_blocks, size_processes);
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

First Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
250 200 100 350
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           200 (P2)                0
2                 100           100 (P3)                0
3                 300           250 (P1)                50
4                 400           350 (P4)                50
5                 500           --                      --

Total Internal Fragmentation: 100
Total External Fragmentation: 0
```

```
FAIZAN CHOUDHARY
20BCS021

First Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
450 210 210 350
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           --                      --
2                 100           --                      --
3                 300           210 (P2)                90
4                 400           210 (P3)                190
5                 500           450 (P1)                50

Total Internal Fragmentation: 330
Total External Fragmentation: 300
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

24th March 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
// array to store process indices for each block index
int allocation_block[100] = {-1};
int totIntFrag=0, totExtFrag=0;
// temp array to store size of blocks for display
int temp[100];
// array to store internal fragmentation of each block
int intFrag[100] = {0};
// array to store the occupancy status of each block
bool occupied_block[100] = {false};
// counter to keep track of allocated processes
int counter=0;


void display (int *s_b, int *s_p) {
    cout<<"\nAfter allocation:\n";
    cout<<"\nBLOCK ID\tBLOCK SIZE\tPROCESS\t\tINTERNAL FRAGMENTATION\n";
    for (int i=0; i<n; i++) {
        cout<<i+1<<"\t\t  "<<temp[i]<<"\t\t";
        // if block is actually allocated a process
        if (occupied_block[i] == false || allocation_block[i] == -1)
            cout<<"--\t\t\t--";
        else if (allocation_block[i] != -1) {
            cout<<s_p[allocation_block[i]]<<" (P"<<allocation_block[i] + 1<<")\t\t";
            cout<<intFrag[i];
        }
        cout<<endl;
    }
    cout<<"\nTotal Internal Fragmentation: "<<totIntFrag;
    cout<<"\nTotal External Fragmentation: "<<totExtFrag<<endl<<endl;
}

void nextFit (int *s_b, int *s_p) {
    for (int i=0; i<n; i++)
        temp[i] = s_b[i];

    int j=0;

    for (int i=0; i<no; i++) {
```

```
        while (j<n) {
            if (s_b[j] >= s_p[i]) {

                if (occupied_block[j] == false) {
                    counter++;
                    allocation_block[j] = i;
                    occupied_block[j] = true;

                    intFrag[j] = s_b[j] - s_p[i];
                    // cout<<intFrag[j]<<endl;
                    // subtracting the value of memory that has been allocated
                    s_b[j] -= s_p[i];
                    j = (j+1) % n;
                }
                break;
            }
            // to maintain the property of the next fit
            j = (j+1) % n;
        }
    }

    for (int i=0; i<n; i++) {
        // cout<<allocation_block[i]<<endl;
        if (occupied_block[i] == true)
            totIntFrag += intFrag[i];
        if (occupied_block[i] == false && counter < no)
            totExtFrag += s_b[i];
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nNext Fit Memory Management\n";

    cout<<"\nEnter number of memory blocks: ";
    cin>>n;

    int size_blocks[100];
    cout<<"\nEnter the size of each block:\n";
    for (int i=0; i<n; i++)
        cin>>size_blocks[i];

    cout<<"\nEnter number of processes: ";
    cin>>no;

    int size_processes[100];
    cout<<"\nEnter the size of each process:\n";
    for (int i=0; i<no; i++)
        cin>>size_processes[i];

    nextFit (size_blocks, size_processes);
    display (size_blocks, size_processes);
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Next Fit Memory Management

Enter number of memory blocks: 3

Enter the size of each block:
5 10 20

Enter number of processes: 3

Enter the size of each process:
10 20 5
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 5             5 (P3)          0
2                 10            10 (P1)         0
3                 20            20 (P2)         0

Total Internal Fragmentation: 0
Total External Fragmentation: 0
```

```
FAIZAN CHOUDHARY
20BCS021

Next Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
100 500 200 450 600

Enter number of processes: 4

Enter the size of each process:
212 417 112 426
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 100           --                      --
2                 500           212 (P1)                288
3                 200           --                      --
4                 450           417 (P2)                33
5                 600           112 (P3)                488

Total Internal Fragmentation: 809
Total External Fragmentation: 300
```

```
FAIZAN CHOUDHARY
20BCS021

Next Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
250 200 100 350
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           --                      --
2                 100           --                      --
3                 300           250 (P1)                50
4                 400           200 (P2)                200
5                 500           100 (P3)                400

Total Internal Fragmentation: 650
Total External Fragmentation: 300
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

31st March 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
// array to store process indices for each block index
int allocation_block[100] = {-1};
int totIntFrag=0, totExtFrag=0;
// temp array to store size of blocks for display
int temp[100];
// array to store internal fragmentation of each block
int intFrag[100] = {0};
// array to store the occupancy status of each block
bool occupied_block[100] = {false};
// counter to keep track of allocated processes
int counter=0;


void display (int *s_b, int *s_p) {
    cout<<"\nAfter allocation:\n";
    cout<<"\nBLOCK ID\tBLOCK SIZE\tPROCESS\t\tINTERNAL FRAGMENTATION\n";
    for (int i=0; i<n; i++) {
        cout<<i+1<<"\t\t  "<<temp[i]<<"\t\t";
        // if block is actually allocated a process
        if (occupied_block[i] == false || allocation_block[i] == -1)
            cout<<"--\t\t\t--";
        else if (allocation_block[i] != -1) {
            cout<<s_p[allocation_block[i]]<<" (P"<<allocation_block[i] + 1<<")\t\t";
            cout<<intFrag[i];
        }
        cout<<endl;
    }
    cout<<"\nTotal Internal Fragmentation: "<<totIntFrag;
    cout<<"\nTotal External Fragmentation: "<<totExtFrag<<endl<<endl;

}

void bestFit (int *s_b, int *s_p) {
    for (int i=0; i<n; i++)
        temp[i] = s_b[i];

    for (int i=0; i<no; i++) {
        // to store the index of the best fit
```

```
        int idx = -1;
        for (int j=0; j<n; j++) {
            if (s_b[j] >= s_p[i] && (idx == -1 || s_b[idx] > s_b[j]) && occupied_block[j]
== false)
                idx = j;
        }

        // for a successful best fit
        if (idx != -1) {
            counter++;
            allocation_block[idx] = i;
            occupied_block[idx] = true;
            intFrag[idx] = s_b[idx] - s_p[i];
            s_b[idx] -= s_p[i];
        }
    }

    for (int i=0; i<n; i++) {
        // cout<<allocation_block[i]<<endl;
        if (occupied_block[i] == true)
            totIntFrag += intFrag[i];
        if (occupied_block[i] == false && counter < no)
            totExtFrag += s_b[i];
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nBest Fit Memory Management\n";

    cout<<"\nEnter number of memory blocks: ";
    cin>>n;

    int size_blocks[100];
    cout<<"\nEnter the size of each block:\n";
    for (int i=0; i<n; i++)
        cin>>size_blocks[i];

    cout<<"\nEnter number of processes: ";
    cin>>no;

    int size_processes[100];
    cout<<"\nEnter the size of each process:\n";
    for (int i=0; i<no; i++)
        cin>>size_processes[i];

    bestFit (size_blocks, size_processes);
    display (size_blocks, size_processes);
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Best Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
100 500 200 300 600

Enter number of processes: 4

Enter the size of each process:
212 417 112 426
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 100           --                      --
2                 500           417 (P2)                83
3                 200           112 (P3)                88
4                 300           212 (P1)                88
5                 600           426 (P4)                174

Total Internal Fragmentation: 433
Total External Fragmentation: 0
```

```
FAIZAN CHOUDHARY
20BCS021

Best Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
250 200 100 350
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           200 (P2)                0
2                 100           100 (P3)                0
3                 300           250 (P1)                50
4                 400           350 (P4)                50
5                 500           --                      --

Total Internal Fragmentation: 100
Total External Fragmentation: 0
```


Go to Index

```
FAIZAN CHOUDHARY
20BCS021

Best Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
450 210 210 350
```

```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           --                      --
2                 100           --                      --
3                 300           210 (P2)                90
4                 400           210 (P3)                190
5                 500           450 (P1)                50

Total Internal Fragmentation: 330
Total External Fragmentation: 300
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

7th April 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
// array to store process indices for each block index
int allocation_block[100] = {-1};
int totIntFrag=0, totExtFrag=0;
// temp array to store size of blocks for display
int temp[100];
// array to store internal fragmentation of each block
int intFrag[100] = {0};
// array to store the occupancy status of each block
bool occupied_block[100] = {false};
// counter to keep track of allocated processes
int counter=0;


void display (int *s_b, int *s_p) {
    cout<<"\nEntered block sizes:\n";
    cout<<"| ";
    for (int i=0; i<n; i++)
        cout<<temp[i]<<" | ";
    cout<<endl;
    cout<<"Entered process sizes:\n";
    cout<<"| ";
    for (int i=0; i<no; i++)
        cout<<s_p[i]<<" | ";
    cout<<endl;
    cout<<"\nAfter allocation:\n";
    cout<<"\nBLOCK ID\tBLOCK SIZE\tPROCESS\t\tINTERNAL FRAGMENTATION\n";
    for (int i=0; i<n; i++) {
        cout<<i+1<<"\t\t  "<<temp[i]<<"\t\t";
        // if block is actually allocated a process
        if (occupied_block[i] == false || allocation_block[i] == -1)
            cout<<"--\t\t\t--";
        else if (allocation_block[i] != -1) {
            cout<<s_p[allocation_block[i]]<<" (P"<<allocation_block[i] + 1<<")\t\t";
            cout<<intFrag[i];
        }
        cout<<endl;
    }
    cout<<"\nTotal Internal Fragmentation: "<<totIntFrag;
```

```
    cout<<"\nTotal External Fragmentation: "<<totExtFrag<<endl<<endl;

}

void worstFit (int *s_b, int *s_p) {
    for (int i=0; i<n; i++)
        temp[i] = s_b[i];

    for (int i=0; i<no; i++) {
        // to store the index of the worst fit
        int idx = -1;
        for (int j=0; j<n; j++) {
            if (s_b[j] >= s_p[i] && (idx == -1 || s_b[idx] < s_b[j]) && occupied_block[j]
== false)
                idx = j;
        }

        // for a successful worst fit
        if (idx != -1) {
            counter++;
            allocation_block[idx] = i;
            occupied_block[idx] = true;
            intFrag[idx] = s_b[idx] - s_p[i];
            s_b[idx] -= s_p[i];
        }
    }

    for (int i=0; i<n; i++) {
        // cout<<allocation_block[i]<<endl;
        if (occupied_block[i] == true)
            totIntFrag += intFrag[i];
        if (occupied_block[i] == false && counter < no)
            totExtFrag += s_b[i];
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nWorst Fit Memory Management\n";

    cout<<"\nEnter number of memory blocks: ";
    cin>>n;

    int size_blocks[100];
    cout<<"\nEnter the size of each block:\n";
    for (int i=0; i<n; i++)
        cin>>size_blocks[i];

    cout<<"\nEnter number of processes: ";
    cin>>no;

    int size_processes[100];
    cout<<"\nEnter the size of each process:\n";
    for (int i=0; i<no; i++)
```

```
        cin>>size_processes[i];

    worstFit (size_blocks, size_processes);
    display (size_blocks, size_processes);
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Worst Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
100 500 200 300 600

Enter number of processes: 4

Enter the size of each process:
212 417 112 426

Entered block sizes:
| 100 | 500 | 200 | 300 | 600 |
Entered process sizes:
| 212 | 417 | 112 | 426 |
```

```
After allocation:

BLOCK ID         BLOCK SIZE       PROCESS          INTERNAL FRAGMENTATION
1                  100            --                         --
2                  500            417 (P2)                   83
3                  200            --                         --
4                  300            112 (P3)                   188
5                  600            212 (P1)                   388

Total Internal Fragmentation: 659
Total External Fragmentation: 300
```

```
FAIZAN CHOUDHARY
20BCS021

Worst Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
250 200 100 350

Entered block sizes:
| 200 | 100 | 300 | 400 | 500 |
Entered process sizes:
| 250 | 200 | 100 | 350 |
```

```
After allocation:

BLOCK ID          BLOCK SIZE        PROCESS           INTERNAL FRAGMENTATION
1                   200             --                       --
2                   100             --                       --
3                   300             100 (P3)                 200
4                   400             200 (P2)                 200
5                   500             250 (P1)                 250

Total Internal Fragmentation: 650
Total External Fragmentation: 300
```

```
FAIZAN CHOUDHARY
20BCS021

Worst Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
450 210 210 350

Entered block sizes:
| 200 | 100 | 300 | 400 | 500 |
Entered process sizes:
| 450 | 210 | 210 | 350 |
```

```
After allocation:

BLOCK ID          BLOCK SIZE        PROCESS           INTERNAL FRAGMENTATION
1                   200             --                       --
2                   100             --                       --
3                   300             210 (P3)                 90
4                   400             210 (P2)                 190
5                   500             450 (P1)                 50

Total Internal Fragmentation: 330
Total External Fragmentation: 300
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

28th April 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
using namespace std;
int n, no;
int hit_indices[100];
int counter=0;
int page_faults=0;

int findIndex (int ref_ele, int *page_slots) {
    for (int i=0; i<no; i++) {
        if (page_slots[i] == ref_ele)
            return i;
    }
    return -1;
}

void display (int ref_ele, int *page_slots, int hit_index) {
    cout<<"|\t      "<<ref_ele<<"\t          |\t"<<(hit_index != -1 ? "Hit   " :
"Fault")<<"      |";
    for (int i=0; i<no; i++)
        cout<<" ";
    for (int i=0; i<no; i++) {
        if (page_slots[i] != -1)
            cout<<page_slots[i]<<"  ";
        else
            cout<<"-  ";
    }
    for (int i=2; i<no; i++)
        cout<<" ";
    cout<<"|\n";
}

void FIFO_replacement(int *ref_str, int *page_slots) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<no; j++) {
            if (page_slots[j] == -1) {
                page_faults++;
                page_slots[j++] = ref_str[i];
                break;
            }
            else if (page_slots[j] != -1 && findIndex(ref_str[i], page_slots) != -1 ) {
                hit_indices[i] = findIndex(ref_str[i], page_slots);
                break;
```

```
            }
            else {
                page_faults++;
                counter = (counter + 1) % no;
                page_slots[counter] = ref_str[i];
                break;
            }
        }
        display(ref_str[i], page_slots, hit_indices[i]);
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nFirst In First Out (FIFO) Page Replacement\n";
    cout<<"\nEnter the number of elements in page reference string: ";
    cin>>n;
    int *ref_str = new int[n];
    cout<<"\nEnter the reference string: ";
    for (int i=0; i<n; i++)
        cin>>ref_str[i];

    cout<<"\nEnter the number of page slots (pages that can be accomodated in memory): ";
    cin>>no;
    int *page_slots = new int[no];
    for (int i=0; i<no; i++)
        page_slots[i] = -1;

    for (int i=0; i<n; i++)
        hit_indices[i] = -1;

    // cout<<endl<<" -------------------------------------------------------- ";
    cout<<"\n|  Reference String Entry |   Hit/Fault   |";
    for (int i=1; i<no; i++)
        cout<<" ";
    if (no < 4)
        cout<<"Page Slots";
    else
        cout<<" Page Slots ";
    for (int i=1; i<no; i++)
        cout<<" ";
    cout<<"|\n\n";
    // cout<<" -------------------------------------------------------- \n";
    FIFO_replacement (ref_str, page_slots);
    // cout<<" -------------------------------------------------------- \n";

    double avg_page_fault = (double)page_faults/n;
    cout<<"\nNumber of page faults: "<<page_faults<<endl;
    cout<<"Number of page hits: "<<n-page_faults<<endl;
    cout<<"\nHit Ratio: "<<(1-avg_page_fault)<<endl;
    cout<<"Average number of page faults (Miss ratio): "<<avg_page_fault<<endl<<endl;

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

First In First Out (FIFO) Page Replacement

Enter the number of elements in page reference string: 6

Enter the reference string: 1 3 0 3 5 6

Enter the number of page slots (pages that can be accomodated in memory): 3

|  Reference String Entry |   Hit/Fault    |  Page Slots  |

|             1           |     Fault      |  1  -  -     |
|             3           |     Fault      |  1  3  -     |
|             0           |     Fault      |  1  3  0     |
|             3           |    Hit         |  1  3  0     |
|             5           |     Fault      |  5  3  0     |
|             6           |     Fault      |  5  6  0     |

Number of page faults: 5
Number of page hits: 1

Hit Ratio: 0.166667
Average number of page faults (Miss ratio): 0.833333
```

```
FAIZAN CHOUDHARY
20BCS021

First In First Out (FIFO) Page Replacement

Enter the number of elements in page reference string: 8

Enter the reference string: 4 0 1 0 1 5 4 1

Enter the number of page slots (pages that can be accomodated in memory): 4

|  Reference String Entry |   Hit/Fault    |    Page Slots    |

|             4           |     Fault      |    4  -  -  -    |
|             0           |     Fault      |    4  0  -  -    |
|             1           |     Fault      |    4  0  1  -    |
|             0           |    Hit         |    4  0  1  -    |
|             1           |    Hit         |    4  0  1  -    |
|             5           |     Fault      |    4  0  1  5    |
|             4           |    Hit         |    4  0  1  5    |
|             1           |    Hit         |    4  0  1  5    |

Number of page faults: 4
Number of page hits: 4

Hit Ratio: 0.5
Average number of page faults (Miss ratio): 0.5
```

```
FAIZAN CHOUDHARY
20BCS021

First In First Out (FIFO) Page Replacement

Enter the number of elements in page reference string: 12

Enter the reference string: 0 2 1 6 4 0 1 0 3 1 2 1

Enter the number of page slots (pages that can be accomodated in memory): 4

|  Reference String Entry |   Hit/Fault    |    Page Slots    |

|            0            |     Fault      |    0  -  -  -    |
|            2            |     Fault      |    0  2  -  -    |
|            1            |     Fault      |    0  2  1  -    |
|            6            |     Fault      |    0  2  1  6    |
|            4            |     Fault      |    4  2  1  6    |
|            0            |     Fault      |    4  0  1  6    |
|            1            |     Hit        |    4  0  1  6    |
|            0            |     Hit        |    4  0  1  6    |
|            3            |     Fault      |    4  0  3  6    |
|            1            |     Fault      |    4  0  3  1    |
|            2            |     Fault      |    2  0  3  1    |
|            1            |     Hit        |    2  0  3  1    |

Number of page faults: 9
Number of page hits: 3

Hit Ratio: 0.25
Average number of page faults (Miss ratio): 0.75
```

```
FAIZAN CHOUDHARY
20BCS021

First In First Out (FIFO) Page Replacement

Enter the number of elements in page reference string: 10

Enter the reference string: 2 5 3 6 3 7 6 4 8 1

Enter the number of page slots (pages that can be accomodated in memory): 3

|  Reference String Entry |   Hit/Fault    |  Page Slots  |

|            2            |     Fault      |   2  -  -    |
|            5            |     Fault      |   2  5  -    |
|            3            |     Fault      |   2  5  3    |
|            6            |     Fault      |   6  5  3    |
|            3            |     Hit        |   6  5  3    |
|            7            |     Fault      |   6  7  3    |
|            6            |     Hit        |   6  7  3    |
|            4            |     Fault      |   6  7  4    |
|            8            |     Fault      |   8  7  4    |
|            1            |     Fault      |   8  1  4    |

Number of page faults: 8
Number of page hits: 2

Hit Ratio: 0.2
Average number of page faults (Miss ratio): 0.8
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

28th April 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
int hit_indices[100];
// counter variable to keep track of number of page slots filled
int counter=0;
int page_faults=0;
// pointer for the dist array to store the distance of each page from the current page in
the ref_str
int *dist;

int findIndex (int ref_ele, int *page_slots) {
    for (int i=0; i<no; i++) {
        if (page_slots[i] == ref_ele)
            return i;
    }
    return -1;
}

void display (int ref_ele, int *page_slots, int hit_index) {
    cout<<"|\t      "<<ref_ele<<"\t          |\t"<<(hit_index != -1 ? "Hit  " :
"Fault")<<"     |";
    for (int i=0; i<no; i++)
        cout<<" ";
    for (int i=0; i<no; i++) {
        if (page_slots[i] != -1)
            cout<<page_slots[i]<<"  ";
        else
            cout<<"-  ";
    }
    for (int i=2; i<no; i++)
        cout<<" ";
    cout<<"|\n";
}

void LRU_replacement(int *ref_str, int *page_slots) {
    for (int i=0; i<n; i++) {
        // condition for empty page slots (frames)
        if (counter < no) {
            page_faults++;
            page_slots[counter++] = ref_str[i];
```

```
        }
        // page hit condition
        else if (findIndex(ref_str[i], page_slots) != -1) {
            hit_indices[i] = findIndex(ref_str[i], page_slots);
        }
        // LRU replacement
        else {
            // mx variable to store max value of dist array, idx to store the index of
this max value
            int mx = INT_MIN, idx;
            // looping through page slots to find the max value of dist array
            for (int j=0; j<no; j++) {
                // initializing dist array for each element in page_slots
                dist[j] = 0;
                // reverse looping through the ref_str (only for the elements in
page_slots) to update the distance of each page from the current page
                // the greater the distance the least used the page will be
                for (int k=i-1; k>=0; k--) {
                    ++dist[j];
                    // if match found, stop increasing the distance
                    if (page_slots[j] == ref_str[k])
                        break;
                }
                // replacing mx with the max value of dist array and storing index in idx
                if (mx < dist[j]) {
                    mx = dist[j];
                    idx = j;
                }
            }
            page_faults++;
            // inserting at the max idx found
            page_slots[idx] = ref_str[i];
        }
        display(ref_str[i], page_slots, hit_indices[i]);
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nLeast Recently Used (LRU) Page Replacement\n";
    cout<<"\nEnter the number of elements in page reference string: ";
    cin>>n;

    int *ref_str = new int[n];
    dist = new int[n];

    cout<<"\nEnter the reference string: ";
    for (int i=0; i<n; i++)
        cin>>ref_str[i];

    cout<<"\nEnter the number of page slots (pages that can be accomodated in memory): ";
    cin>>no;
    int *page_slots = new int[no];
    for (int i=0; i<no; i++)
```

```
        page_slots[i] = -1;

    for (int i=0; i<n; i++)
        hit_indices[i] = -1;

    cout<<"\n|  Reference String Entry |   Hit/Fault   |";
    for (int i=1; i<no; i++)
        cout<<" ";
    if (no < 4)
        cout<<"Page Slots";
    else
        cout<<" Page Slots ";
    for (int i=1; i<no; i++)
        cout<<" ";
    cout<<"|\n\n";
    LRU_replacement (ref_str, page_slots);

    double avg_page_fault = (double)page_faults/n;
    cout<<"\nNumber of page faults: "<<page_faults<<endl;
    cout<<"Number of page hits: "<<n-page_faults<<endl;
    cout<<"\nHit Ratio: "<<(1-avg_page_fault)<<endl;
    cout<<"Average number of page faults (Miss ratio): "<<avg_page_fault<<endl<<endl;

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Least Recently Used (LRU) Page Replacement

Enter the number of elements in page reference string: 12

Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5

Enter the number of page slots (pages that can be accomodated in memory): 4

|  Reference String Entry |   Hit/Fault   |    Page Slots    |

|            1            |     Fault     |   1  -  -  -     |
|            2            |     Fault     |   1  2  -  -     |
|            3            |     Fault     |   1  2  3  -     |
|            4            |     Fault     |   1  2  3  4     |
|            1            |     Hit       |   1  2  3  4     |
|            2            |     Hit       |   1  2  3  4     |
|            5            |     Fault     |   1  2  5  4     |
|            1            |     Hit       |   1  2  5  4     |
|            2            |     Hit       |   1  2  5  4     |
|            3            |     Fault     |   1  2  5  3     |
|            4            |     Fault     |   1  2  4  3     |
|            5            |     Fault     |   5  2  4  3     |

Number of page faults: 8
Number of page hits: 4

Hit Ratio: 0.333333
Average number of page faults (Miss ratio): 0.666667
```

```
FAIZAN CHOUDHARY
20BCS021

Least Recently Used (LRU) Page Replacement

Enter the number of elements in page reference string: 10

Enter the reference string: 2 3 4 2 1 3 7 5 4 3

Enter the number of page slots (pages that can be accomodated in memory): 3

|  Reference String Entry |   Hit/Fault    |  Page Slots  |

|            2            |     Fault      |  2  -  -    |
|            3            |     Fault      |  2  3  -    |
|            4            |     Fault      |  2  3  4    |
|            2            |     Hit        |  2  3  4    |
|            1            |     Fault      |  2  1  4    |
|            3            |     Fault      |  2  1  3    |
|            7            |     Fault      |  7  1  3    |
|            5            |     Fault      |  7  5  3    |
|            4            |     Fault      |  7  5  4    |
|            3            |     Fault      |  3  5  4    |

Number of page faults: 9
Number of page hits: 1

Hit Ratio: 0.1
Average number of page faults (Miss ratio): 0.9
```

```
FAIZAN CHOUDHARY
20BCS021

Least Recently Used (LRU) Page Replacement

Enter the number of elements in page reference string: 20

Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of page slots (pages that can be accomodated in memory): 4

|  Reference String Entry |   Hit/Fault    |    Page Slots     |

|            7            |     Fault      |   7  -  -  -      |
|            0            |     Fault      |   7  0  -  -      |
|            1            |     Fault      |   7  0  1  -      |
|            2            |     Fault      |   7  0  1  2      |
|            0            |     Hit        |   7  0  1  2      |
|            3            |     Fault      |   3  0  1  2      |
|            0            |     Hit        |   3  0  1  2      |
|            4            |     Fault      |   3  0  4  2      |
|            2            |     Hit        |   3  0  4  2      |
|            3            |     Hit        |   3  0  4  2      |
|            0            |     Hit        |   3  0  4  2      |
|            3            |     Hit        |   3  0  4  2      |
|            2            |     Hit        |   3  0  4  2      |
|            1            |     Fault      |   3  0  1  2      |
|            2            |     Hit        |   3  0  1  2      |
|            0            |     Hit        |   3  0  1  2      |
|            1            |     Hit        |   3  0  1  2      |
|            7            |     Fault      |   7  0  1  2      |
|            0            |     Hit        |   7  0  1  2      |
|            1            |     Hit        |   7  0  1  2      |

Number of page faults: 8
Number of page hits: 12

Hit Ratio: 0.6
Average number of page faults (Miss ratio): 0.4
```

FAIZAN CHOUDHARY

20BCS021

OS LAB

5th May 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <algorithm>
#include <math.h>
using namespace std;
// head movement data for FCFS (index 0), SJF (index 1), and Elevator (index 2)
int **head_movement = new int*[3];
int *total_head_movement = new int [3];
void sort_sstf (int *disk, int n, int start_pos) {
    int i, j, temp, min_index;
    for (i=0; i<n-1; i++) {
        min_index = i;
        for (j=i+1; j<n; j++) {
            if (abs (disk[j] - start_pos) < abs (disk[min_index] - start_pos))
                min_index = j;
        }
        temp = disk[i];
        disk[i] = disk[min_index];
        disk[min_index] = temp;
    }
}
void sort_elevator (int *disk, int n, int start_pos) {
    int i, j;
    int left_idx = 0, right_idx = 0;
    // partitioning disk elements into two halves, left and right
    int left[n], right[n];
    for (i=0; i<n; i++) {
        if (disk[i] <= start_pos) {
            left [left_idx++] = disk[i];
        }
        else {
            right [right_idx++] = disk[i];
        }
    }
    // sorting them according to distance from start_pos
    sort (left, left + left_idx, greater<int>());
    sort (right, right + right_idx);
    // merging them back
    for (i=0; i<left_idx; i++)
        disk[i] = left[i];
    for (i=0; i<right_idx; i++)
        disk[i+left_idx] = right[i];
}
```

```
void display_pointer_movement (int *disk, int n, int start_pos) {
    int i;
    cout<<"\nPointer movement: ";
    for (i=0; i<n; i++) {
        if (i == 0)
            cout<<start_pos<<" -> "<<disk[i]<<" -> ";
        else {
            if (i == n-1)
                cout<<disk[i];
            else
                cout<<disk[i]<<" -> ";
        }
    }
    cout<<endl;
}
void FCFS (int *disk, int n, int start_pos) {
    for (int i=0; i<n; i++) {
        head_movement[0][i] = abs (disk[i] - start_pos);
        start_pos = disk[i];
        total_head_movement[0] += head_movement[0][i];
    }
}
void Elevator (int *disk, int n, int start_pos) {
    // sorting data for elevator movement
    sort_elevator (disk, n, start_pos);
    for (int i=0; i<n; i++) {
        head_movement[2][i] = abs (disk[i] - start_pos);
        start_pos = disk[i];
        total_head_movement[2] += head_movement[2][i];
    }
}
void SSTF (int *disk, int n, int start_pos) {
    // sorting data for SSTF
    sort_sstf (disk, n, start_pos);
    for (int i=0; i<n; i++) {
        head_movement[1][i] = abs (disk[i] - start_pos);
        start_pos = disk[i];
        total_head_movement[1] += head_movement[1][i];
    }
}
int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nFirst Come First Served (FCFS), Shortest Seek Time First (SSTF) and Elevator
Disk Scheduling\n";
    int n, i;
    cout<<"\nEnter the number of disk requests in the queue: ";
    cin>>n;
    for (i=0; i<3; i++) {
        head_movement[i] = new int [n];
        total_head_movement[i] = 0;
    }
    // initializing disk requests for different scheduling algorithms, since they require
sorting and partitioning
    int *disk_requests = new int [n];
```

```
    int *disk_requests_sstf = new int [n];
    int *disk_requests_elevator = new int [n];
    cout<<"\nEnter the disk requests: ";
    for (i=0; i<n; i++) {
        cin>>disk_requests[i];
        disk_requests_sstf[i] = disk_requests[i];
        disk_requests_elevator[i] = disk_requests[i];
    }
    int start_position;
    cout<<"\nEnter the starting position of the disk head: ";
    cin>>start_position;
    FCFS(disk_requests, n, start_position);
    SSTF(disk_requests_sstf, n, start_position);
    Elevator(disk_requests_elevator, n, start_position);
    for (int i=0; i<3; i++) {
        if (i==0) {
            cout<<"\nFCFS:\n";
            display_pointer_movement (disk_requests, n, start_position);
        }
        else if (i==1) {
            cout<<"\nSSTF:\n";
            display_pointer_movement (disk_requests_sstf, n, start_position);
        }
        else if (i==2) {
            cout<<"\nElevator:\n";
            display_pointer_movement (disk_requests_elevator, n, start_position);
        }
        cout<<"Total head movement: ";
        for (int j=0; j<n; j++) {
            if (j == n-1)
                cout<<head_movement[i][j]<<" = ";
            else
                cout<<head_movement[i][j]<<" + ";
        }
        cout<<total_head_movement[i]<<" tracks"<<endl;
    }
    cout<<endl;
    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

First Come First Served (FCFS), Shortest Seek Time First (SSTF) and Elevator Disk Scheduling

Enter the number of disk requests in the queue: 8

Enter the disk requests: 98 183 37 122 14 124 65 67

Enter the starting position of the disk head: 53

FCFS:

Pointer movement: 53 -> 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67
Total head movement: 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 = 640 tracks

SSTF:

Pointer movement: 53 -> 65 -> 67 -> 37 -> 14 -> 98 -> 122 -> 124 -> 183
Total head movement: 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 = 236 tracks

Elevator:

Pointer movement: 53 -> 37 -> 14 -> 65 -> 67 -> 98 -> 122 -> 124 -> 183
Total head movement: 16 + 23 + 51 + 2 + 31 + 24 + 2 + 59 = 208 tracks
```

```
FAIZAN CHOUDHARY
20BCS021

First Come First Served (FCFS), Shortest Seek Time First (SSTF) and Elevator Disk Scheduling

Enter the number of disk requests in the queue: 4

Enter the disk requests: 65 40 18 78

Enter the starting position of the disk head: 30

FCFS:

Pointer movement: 30 -> 65 -> 40 -> 18 -> 78
Total head movement: 35 + 25 + 22 + 60 = 142 tracks

SSTF:

Pointer movement: 30 -> 40 -> 18 -> 65 -> 78
Total head movement: 10 + 22 + 47 + 13 = 92 tracks

Elevator:

Pointer movement: 30 -> 18 -> 40 -> 65 -> 78
Total head movement: 12 + 22 + 25 + 13 = 72 tracks
```