FAIZAN CHOUDHARY

20BCS021

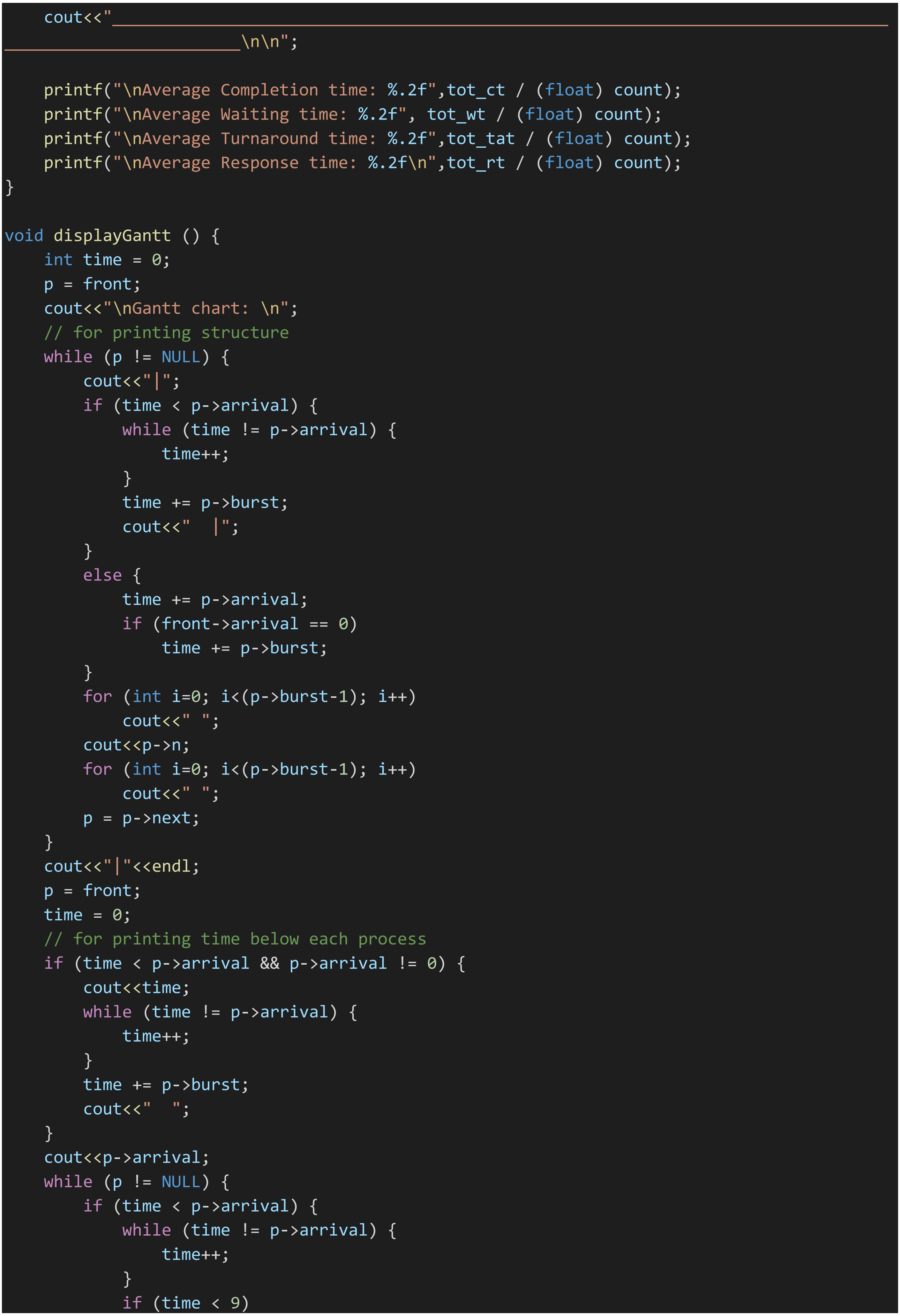
OS LAB

3rd February 2022

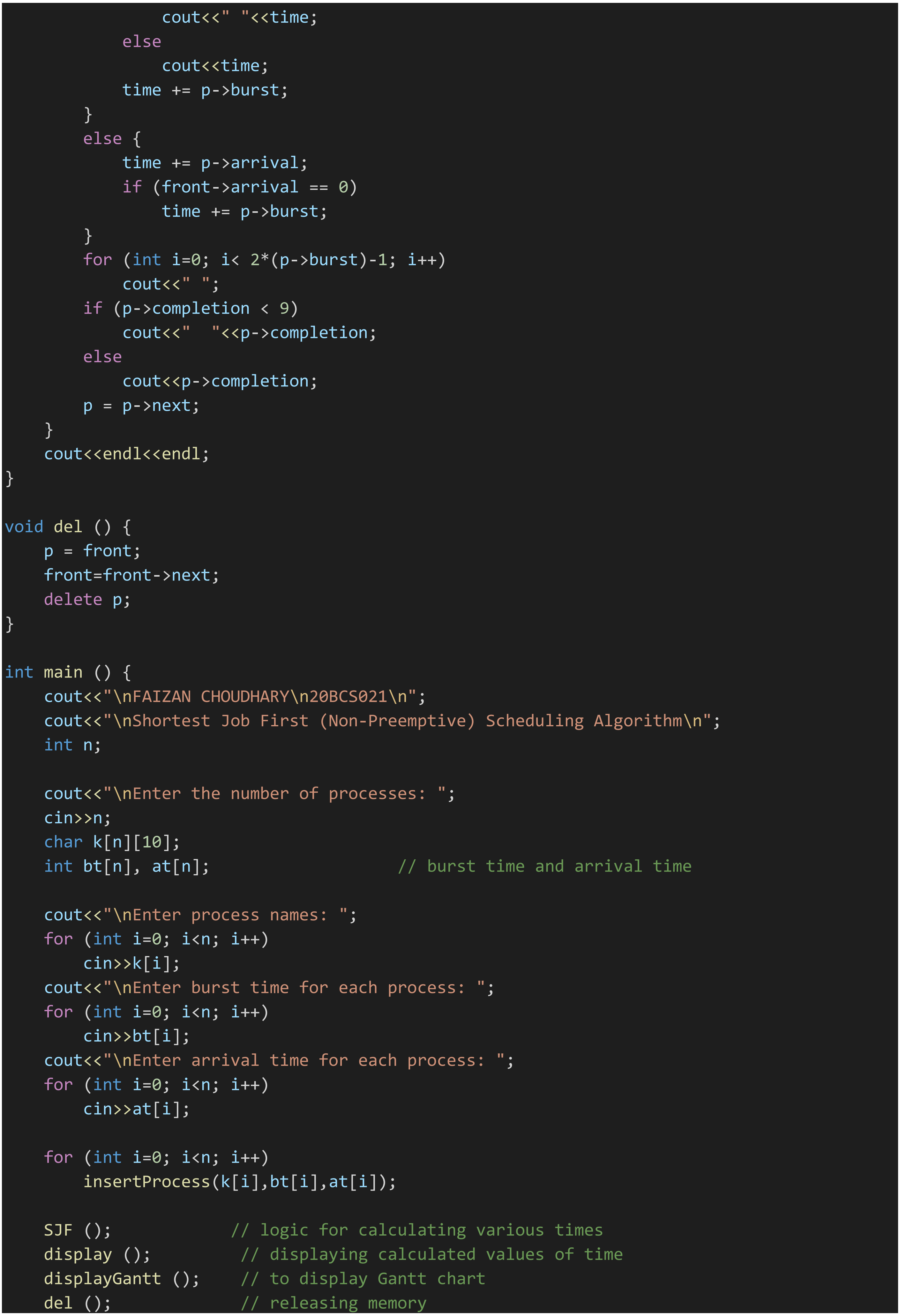
CODE: (code pasted in this format for readability)

```
#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;
    }
```

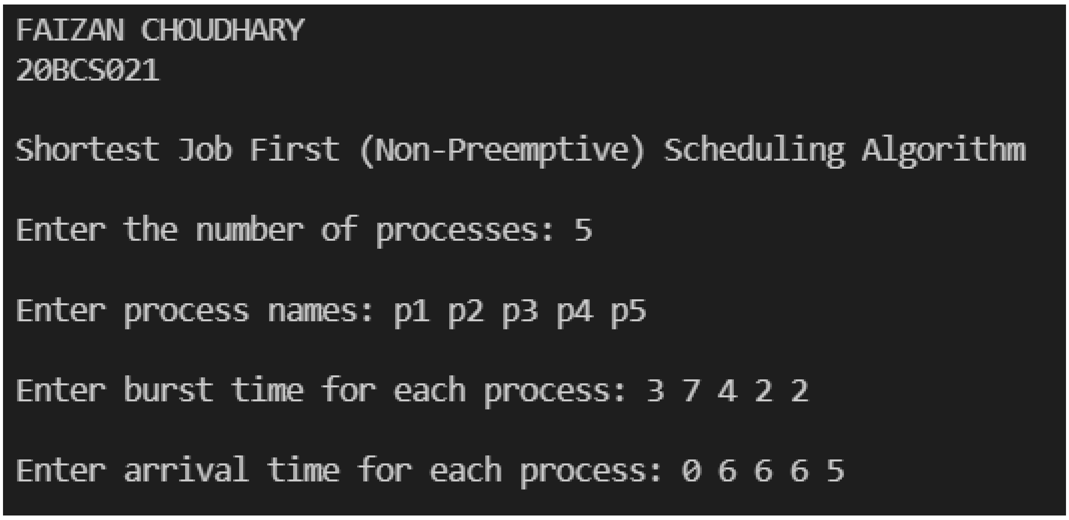


```
    cout<<"_______________________________________________________________________________________________________\n\n";

    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)
```

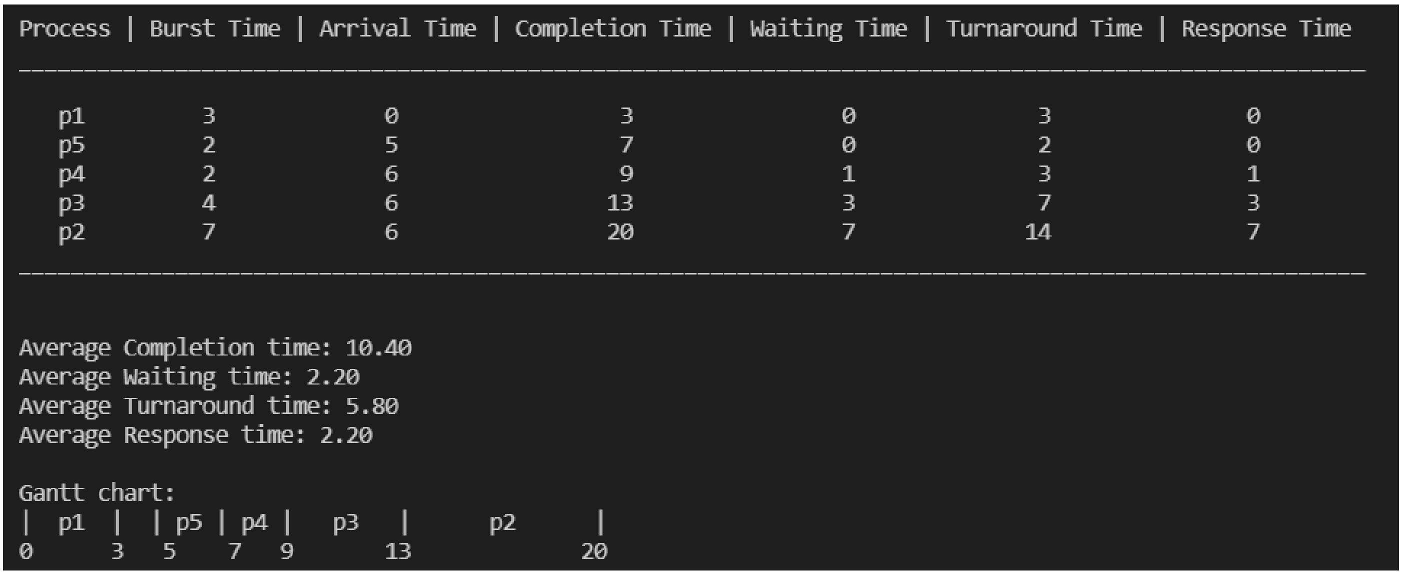


```
                cout<<" "<<time;
            else
                cout<<time;
            time += p->burst;
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i< 2*(p->burst)-1; i++)
    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

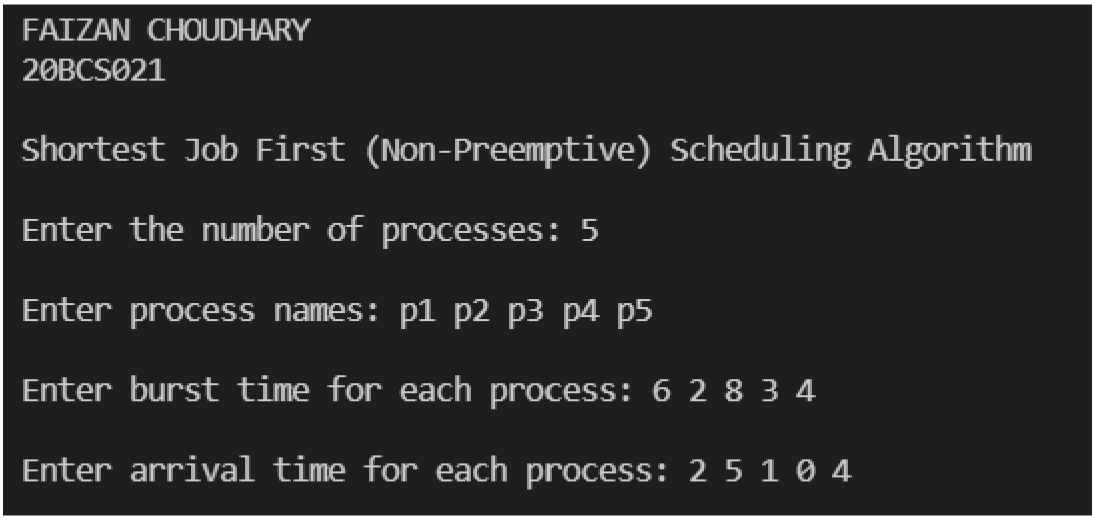
OUTPUT:



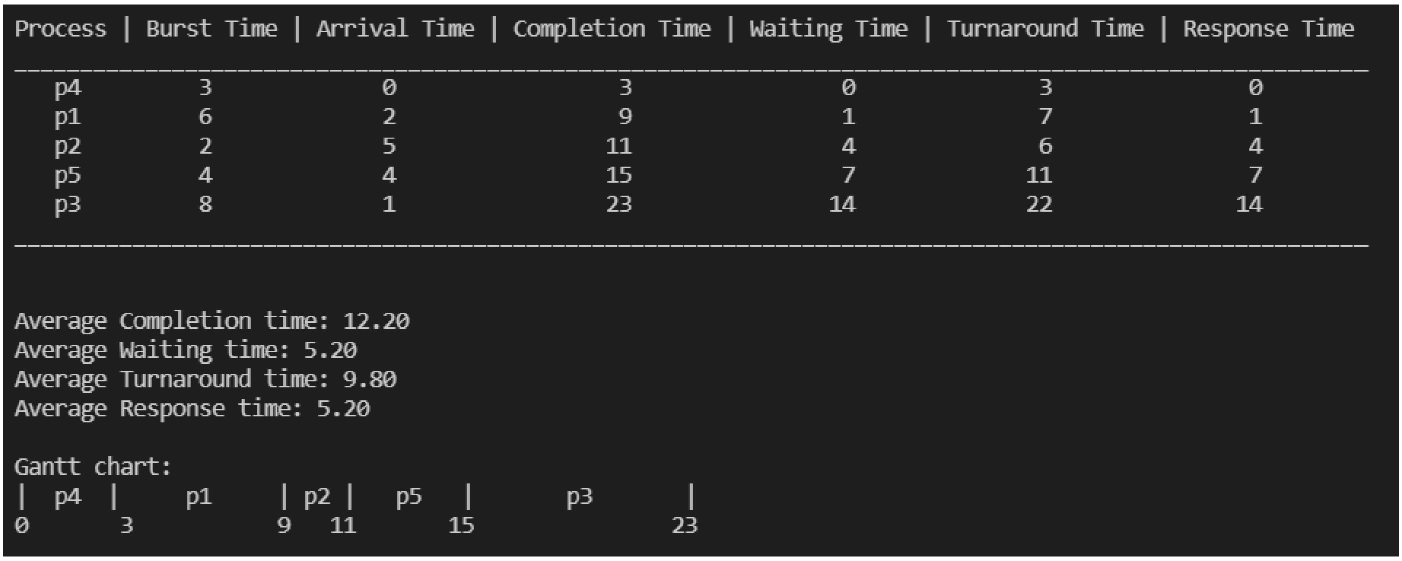
```
FAIZAN CHOUDHARY
20BCS021

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
```