Name: Jogalekar Ishan Sagar

Reg no: 19BCE2250
Course: Operating System CSE2005 (LAB)

Slot: L35+L36

LAB Digital Assignment 2

(a) Process and Thread Management
Write a program to create a thread and perform the following (Easy)

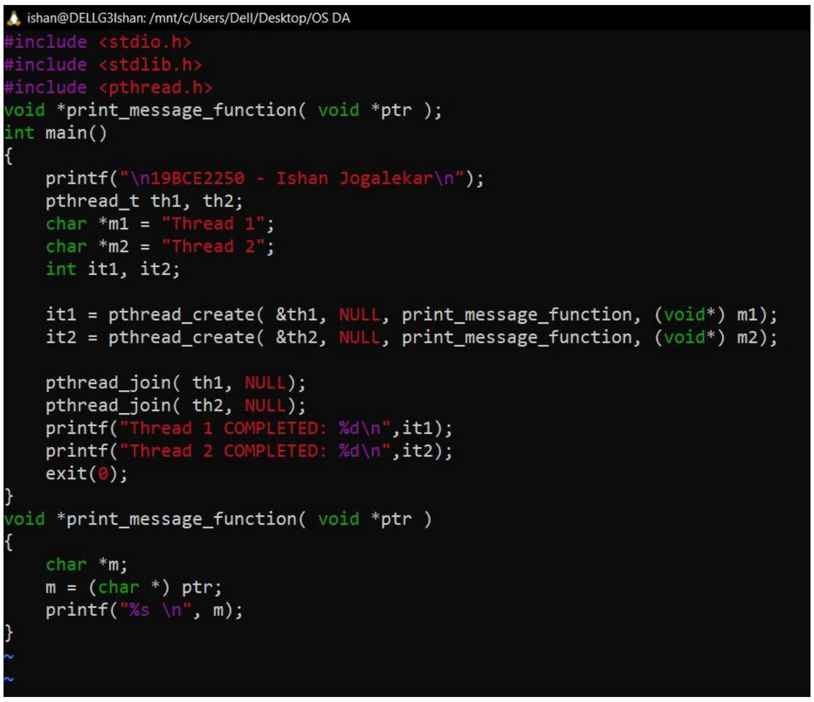
- Create a thread runner function
- Set the thread attributes
- Join the parent and thread
- Wait for the thread to complete

Ans:

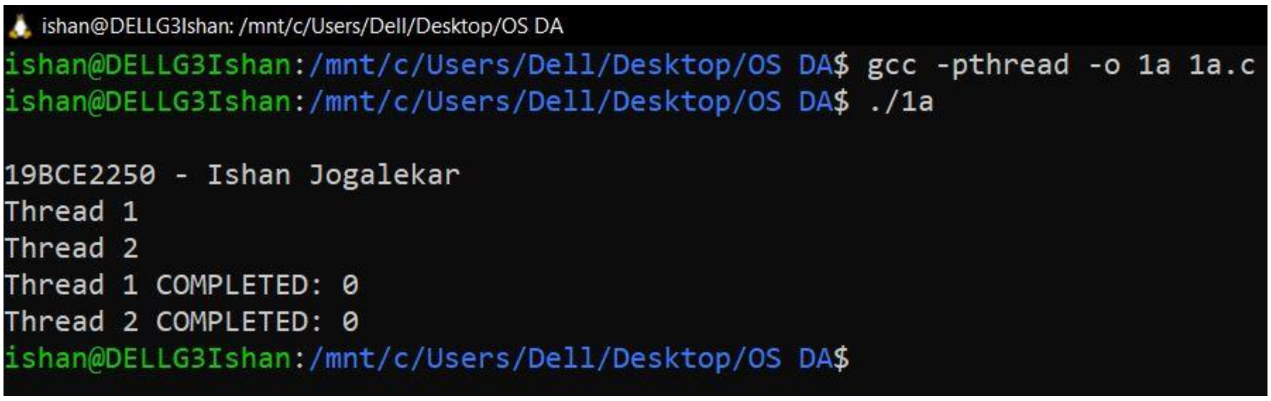
Code :-

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void *print_message_function( void *ptr );

int main()

{

    printf("\n19BCE2250 - Ishan Jogalekar\n");

    pthread_t th1, th2;

    char *m1 = "Thread 1";

    char *m2 = "Thread 2";

    int it1, it2;
    it1 = pthread_create( &th1, NULL, print_message_function, (void*)
m1);
    it2 = pthread_create( &th2, NULL, print_message_function, (void*)
m2);


    pthread_join( th1, NULL);
    pthread_join( th2, NULL);
    printf("Thread 1 COMPLETED: %d\n",it1);
    printf("Thread 2 COMPLETED: %d\n",it2);
    exit(0);
}
void *print_message_function( void *ptr )
{
    char *m;
    m = (char *) ptr;
    printf("%s \n", m);
}
```

Output :-



```
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *ptr );
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");
    pthread_t th1, th2;
    char *m1 = "Thread 1";
    char *m2 = "Thread 2";
    int it1, it2;

    it1 = pthread_create( &th1, NULL, print_message_function, (void*) m1);
    it2 = pthread_create( &th2, NULL, print_message_function, (void*) m2);

    pthread_join( th1, NULL);
    pthread_join( th2, NULL);
    printf("Thread 1 COMPLETED: %d\n",it1);
    printf("Thread 2 COMPLETED: %d\n",it2);
    exit(0);
}
void *print_message_function( void *ptr )
{
    char *m;
    m = (char *) ptr;
    printf("%s \n", m);
}
~
~
```



```
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ gcc -pthread -o 1a 1a.c
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./1a

19BCE2250 - Ishan Jogalekar
Thread 1
Thread 2
Thread 1 COMPLETED: 0
Thread 2 COMPLETED: 0
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$
```

(b) Write a program to create a thread to find the factorial of a natural number ‘n’. (Medium)

Ans:

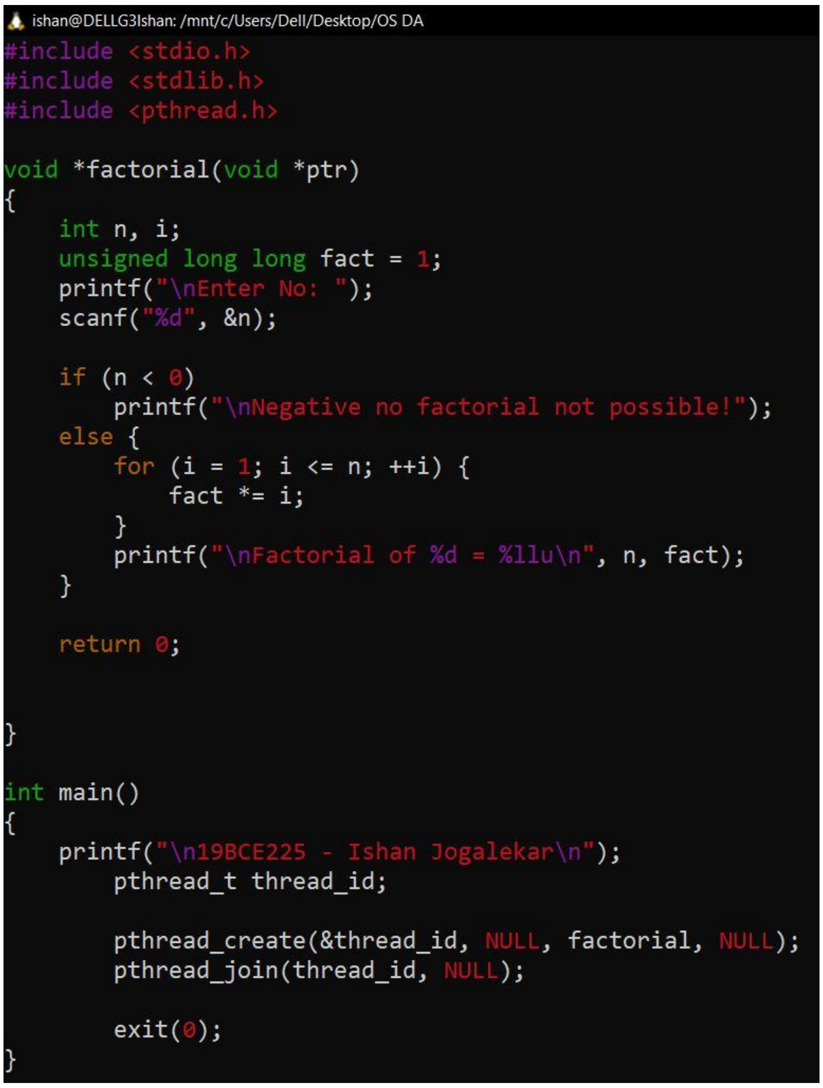
<u>Code</u> :-

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>
```

```
void *factorial(void *ptr)
{
    int n, i;
    unsigned long long fact = 1;
    printf("\nEnter No: ");
    scanf("%d", &n);

    if (n < 0)
        printf("\nNegative no factorial not possible!");
    else {
        for (i = 1; i <= n; ++i) {
            fact *= i;
        }
        printf("\nFactorial of %d = %llu\n", n, fact);
    }

    return 0;
}
int main()
{
    printf("\n19BCE225 - Ishan Jogalekar\n");
         pthread_t thread_id;

         pthread_create(&thread_id, NULL, factorial, NULL);
         pthread_join(thread_id, NULL);

         exit(0);
```

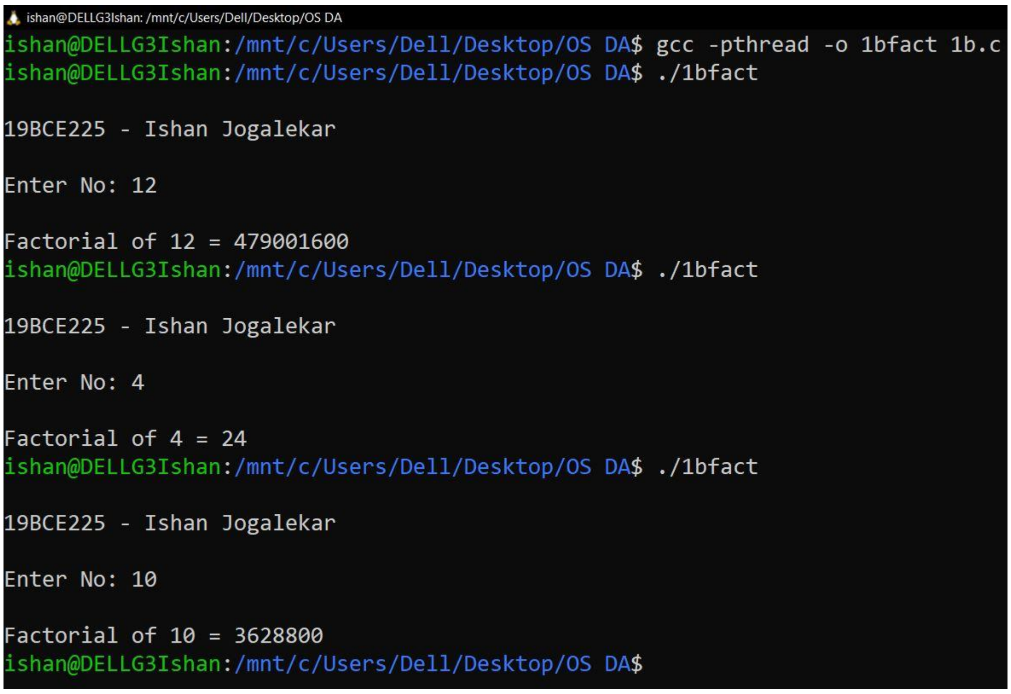
}

Output :-



ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

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

void *factorial(void *ptr)
{
    int n, i;
    unsigned long long fact = 1;
    printf("\nEnter No: ");
    scanf("%d", &n);

    if (n < 0)
        printf("\nNegative no factorial not possible!");
    else {
        for (i = 1; i <= n; ++i) {
            fact *= i;
        }
        printf("\nFactorial of %d = %llu\n", n, fact);
    }

    return 0;


}

int main()
{
    printf("\n19BCE225 - Ishan Jogalekar\n");
        pthread_t thread_id;

        pthread_create(&thread_id, NULL, factorial, NULL);
        pthread_join(thread_id, NULL);

        exit(0);
}
```



```
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ gcc -pthread -o 1bfact 1b.c
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./1bfact

19BCE225 - Ishan Jogalekar

Enter No: 12

Factorial of 12 = 479001600
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./1bfact

19BCE225 - Ishan Jogalekar

Enter No: 4

Factorial of 4 = 24
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./1bfact

19BCE225 - Ishan Jogalekar

Enter No: 10

Factorial of 10 = 3628800
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$
```

(c) Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A…Z. Write a program to demonstrate the above mentioned scenario. (Medium)

Ans :

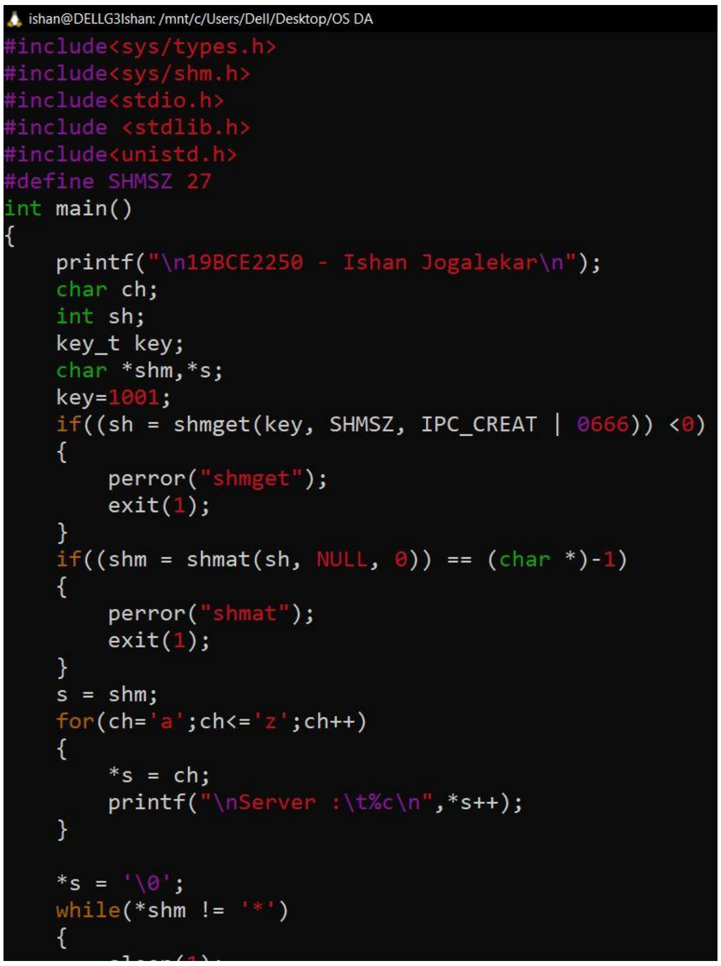
Code:-

1. **Server** :

```
#include<sys/types.h>
#include<sys/shm.h>
#include<stdio.h>
#include <stdlib.h>
#include<unistd.h>
```

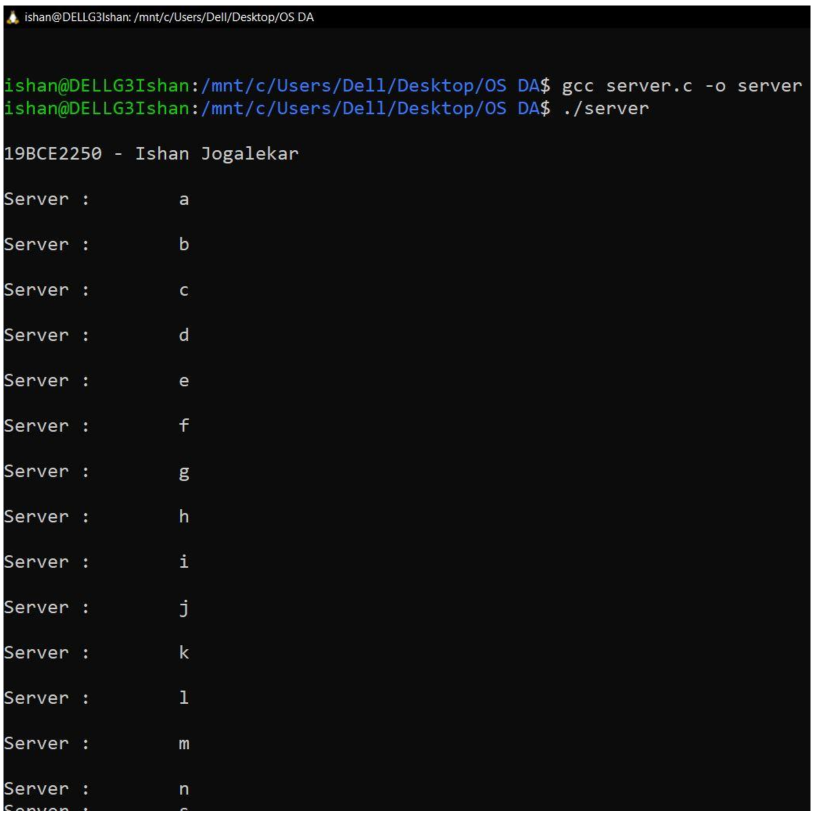
```
#define SHMSZ 27
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");
    char ch;
    int sh;
    key_t key;
    char *shm,*s;
    key=1001;
    if((sh = shmget(key, SHMSZ, IPC_CREAT | 0666)) <0)
    {
        perror("shmget");
        exit(1);
    }
    if((shm = shmat(sh, NULL, 0)) == (char *)-1)
    {
        perror("shmat");
        exit(1);
    }
    s = shm;
    for(ch='a';ch<='z';ch++)
    {
        *s = ch;
        printf("\nServer :\t%c\n",*s++);
    }

    *s = '\0';
    while(*shm != '*')
    {
        sleep(1);
    }
    exit(0);
}
```

<u>Output</u> :-



ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

```c
#include<sys/types.h>
#include<sys/shm.h>
#include<stdio.h>
#include <stdlib.h>
#include<unistd.h>
#define SHMSZ 27
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");
    char ch;
    int sh;
    key_t key;
    char *shm,*s;
    key=1001;
    if((sh = shmget(key, SHMSZ, IPC_CREAT | 0666)) <0)
    {
        perror("shmget");
        exit(1);
    }
    if((shm = shmat(sh, NULL, 0)) == (char *)-1)
    {
        perror("shmat");
        exit(1);
    }
    s = shm;
    for(ch='a';ch<='z';ch++)
    {
        *s = ch;
        printf("\nServer :\t%c\n",*s++);
    }

    *s = '\0';
    while(*shm != '*')
    {
```



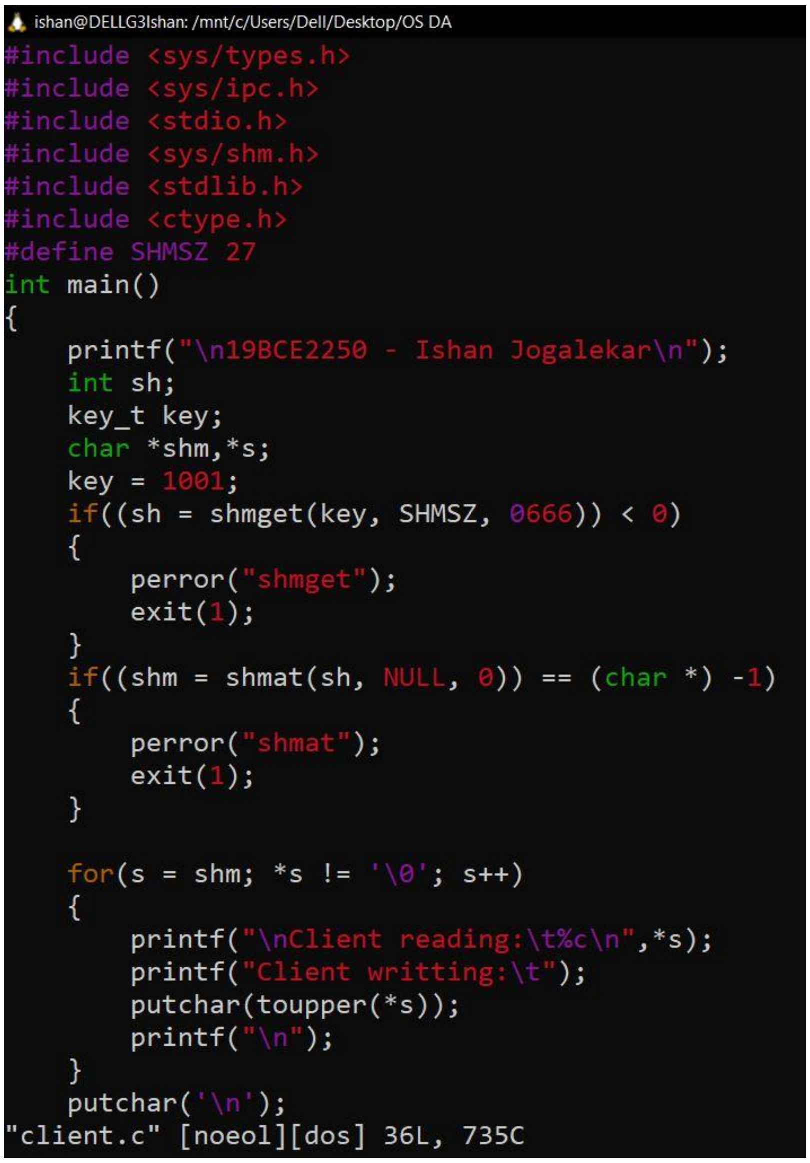
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

```
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ gcc server.c -o server
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./server

19BCE2250 - Ishan Jogalekar

Server :        a

Server :        b

Server :        c

Server :        d

Server :        e

Server :        f

Server :        g

Server :        h

Server :        i

Server :        j

Server :        k

Server :        l

Server :        m

Server :        n
```

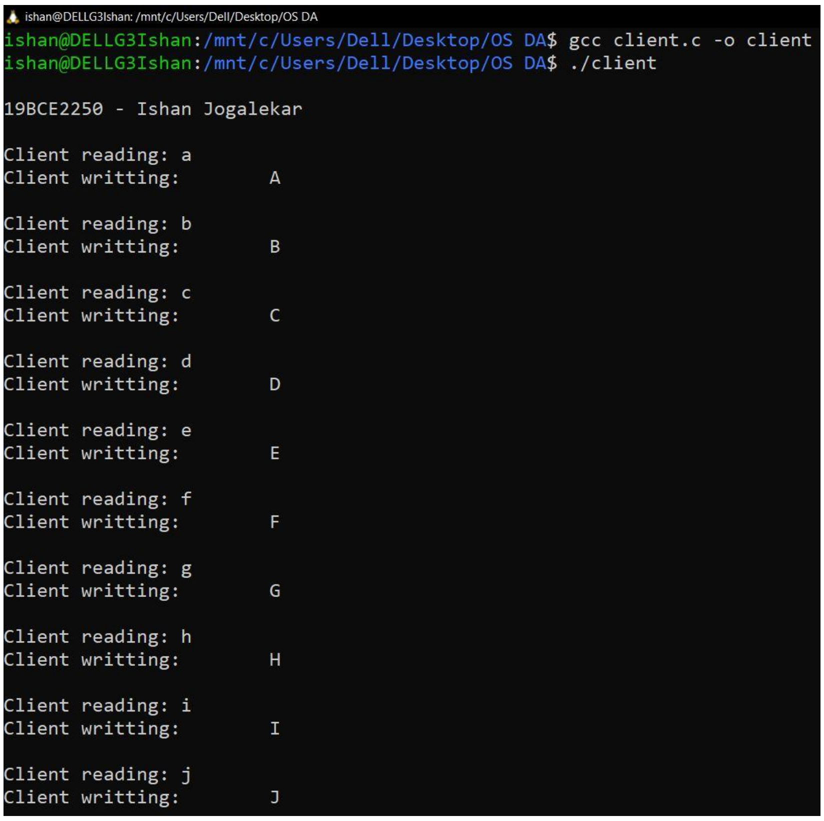
2. <u>**Client**</u> :

```
        #include <sys/types.h>
        #include <sys/ipc.h>
        #include <stdio.h>
        #include <sys/shm.h>
        #include <stdlib.h>
        #include <ctype.h>
        #define SHMSZ 27
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");
    int sh;
    key_t key;
    char *shm,*s;
    key = 1001;
    if((sh = shmget(key, SHMSZ, 0666)) < 0)
    {
        perror("shmget");
        exit(1);
    }
    if((shm = shmat(sh, NULL, 0)) == (char *) -1)
    {
        perror("shmat");
        exit(1);
    }

    for(s = shm; *s != '\0'; s++)
    {
        printf("\nClient reading:\t%c\n",*s);
        printf("Client writting:\t");
        putchar(toupper(*s));
        printf("\n");
    }
    putchar('\n');
    *shm = '*';
    exit(0);
}
```

<u>Output</u> :-


ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <stdio.h>
#include <sys/shm.h>
#include <stdlib.h>
#include <ctype.h>
#define SHMSZ 27
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");
    int sh;
    key_t key;
    char *shm,*s;
    key = 1001;
    if((sh = shmget(key, SHMSZ, 0666)) < 0)
    {
        perror("shmget");
        exit(1);
    }
    if((shm = shmat(sh, NULL, 0)) == (char *) -1)
    {
        perror("shmat");
        exit(1);
    }

    for(s = shm; *s != '\0'; s++)
    {
        printf("\nClient reading:\t%c\n",*s);
        printf("Client writting:\t");
        putchar(toupper(*s));
        printf("\n");
    }
    putchar('\n');
"client.c" [noeol][dos] 36L, 735C
```



```
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ gcc client.c -o client
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./client

19BCE2250 - Ishan Jogalekar

Client reading: a
Client writting:        A

Client reading: b
Client writting:        B

Client reading: c
Client writting:        C

Client reading: d
Client writting:        D

Client reading: e
Client writting:        E

Client reading: f
Client writting:        F

Client reading: g
Client writting:        G

Client reading: h
Client writting:        H

Client reading: i
Client writting:        I

Client reading: j
Client writting:        J
```

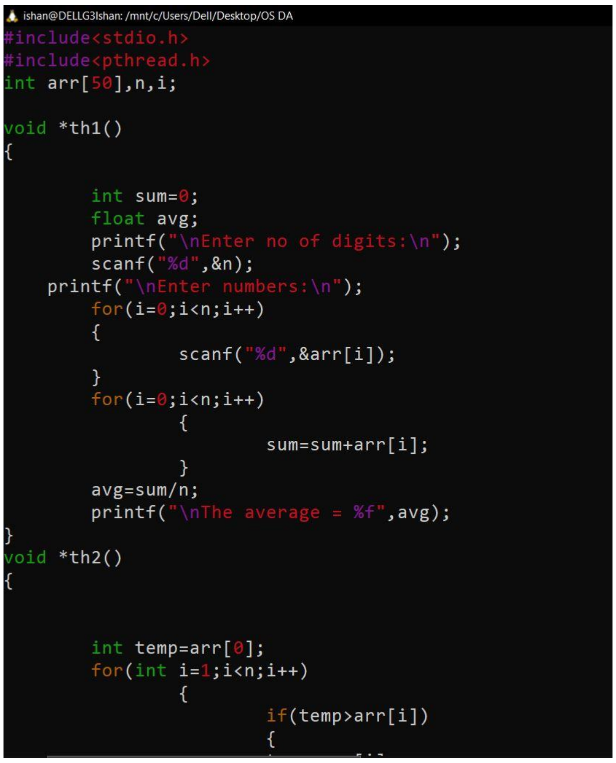
d) Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 , the program will report the average value as 82. The minimum value as 72. The maximum value as 95. The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited. (High)

Ans :

Code :-

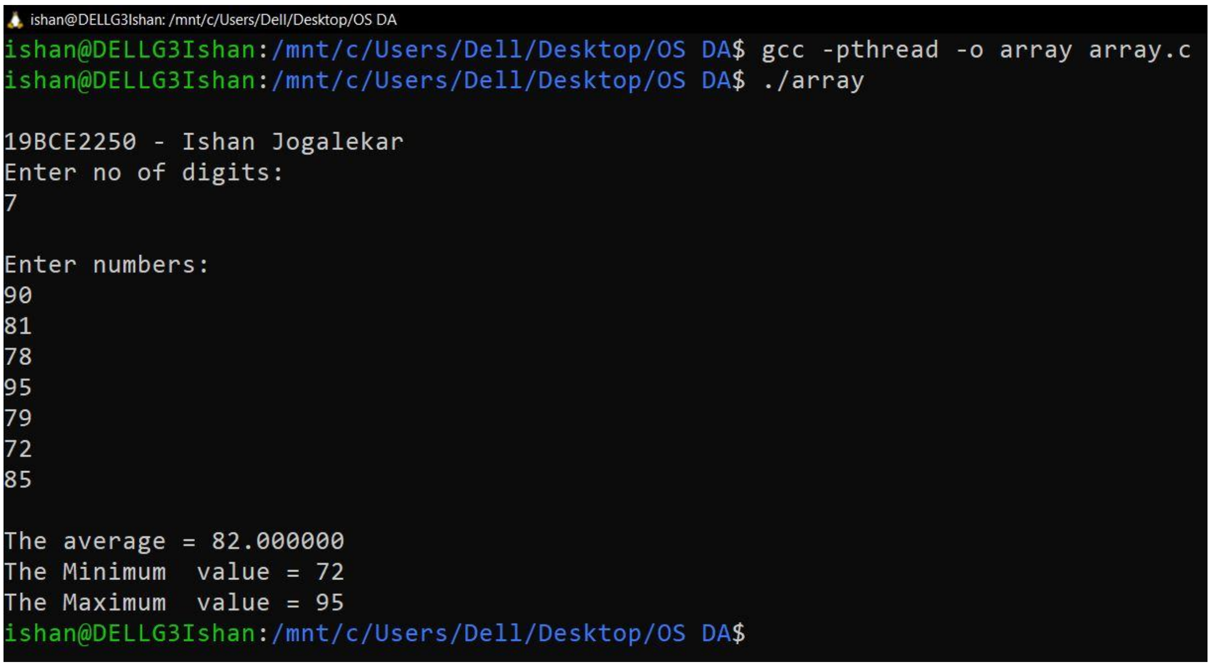
```
#include<stdio.h>
#include<pthread.h>
int arr[50],n,i;

void *th1()
{

        int sum=0;
        float avg;
        printf("\nEnter no of digits:\n");
        scanf("%d",&n);
    printf("\nEnter numbers:\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
        }
        for(i=0;i<n;i++)
                {
                        sum=sum+arr[i];
                }
        avg=sum/n;
        printf("\nThe average = %f",avg);
}
void *th2()
{


        int temp=arr[0];
        for(int i=1;i<n;i++)
                {
                        if(temp>arr[i])
                        {
                        temp=arr[i];
                        }
                }
        printf("\nThe Minimum  value = %d",temp);

}
void *th3()
{

        int temp=arr[0];
        for(int i=1;i<n;i++)
                {
                        if(temp<arr[i])
                        {
                        temp=arr[i];
                        }
                }
        printf("\nThe Maximum  value = %d\n",temp);
        }
int main()
{
   printf("\n19BCE2250 - Ishan Jogalekar");
   int n,i;
   pthread_t t1;
   pthread_t t2;
   pthread_t t3;
        n=pthread_create(&t1,NULL,&th1,NULL);
        pthread_join(t1,NULL);

        n=pthread_create(&t2,NULL,&th2,NULL);
       pthread_join(t2,NULL);

        n=pthread_create(&t3,NULL,&th3,NULL);
       pthread_join(t3,NULL);
}
```

<u>Output</u> :-



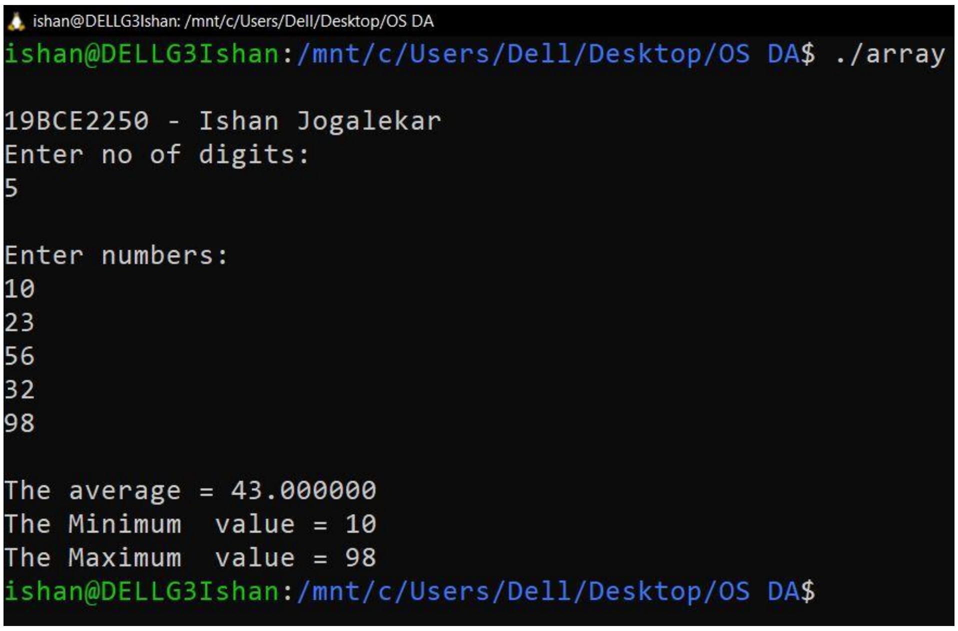
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

```c
#include<stdio.h>
#include<pthread.h>
int arr[50],n,i;

void *th1()
{

        int sum=0;
        float avg;
        printf("\nEnter no of digits:\n");
        scanf("%d",&n);
    printf("\nEnter numbers:\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
        }
        for(i=0;i<n;i++)
                {
                        sum=sum+arr[i];
                }
        avg=sum/n;
        printf("\nThe average = %f",avg);
}
void *th2()
{


        int temp=arr[0];
        for(int i=1;i<n;i++)
                {
                        if(temp>arr[i])
                        {
```



ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA

```
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ gcc -pthread -o array array.c
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./array

19BCE2250 - Ishan Jogalekar
Enter no of digits:
7

Enter numbers:
90
81
78
95
79
72
85

The average = 82.000000
The Minimum  value = 72
The Maximum  value = 95
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$
```



```
ishan@DELLG3Ishan: /mnt/c/Users/Dell/Desktop/OS DA
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$ ./array

19BCE2250 - Ishan Jogalekar
Enter no of digits:
5

Enter numbers:
10
23
56
32
98

The average = 43.000000
The Minimum  value = 10
The Maximum  value = 98
ishan@DELLG3Ishan:/mnt/c/Users/Dell/Desktop/OS DA$
```

CPU Scheduling

a.Implement the various process scheduling algorithms such as FCFS, SJF, Priority (Non Preemptive). (Easy)

Ans :

1. FCFS

Algorithm :-

1- Input the processes along with their burst time(bt)
and arrival time(at)

2- Find waiting time for all other processes i.e. for
a given process i:
wt[i] = (bt[0] + bt[1] +...... bt[i-1]) - at[i]

3- Now find turn around time
= waiting_time + burst_time for all processes

4- Average waiting time =
total_waiting_time / no_of_processes

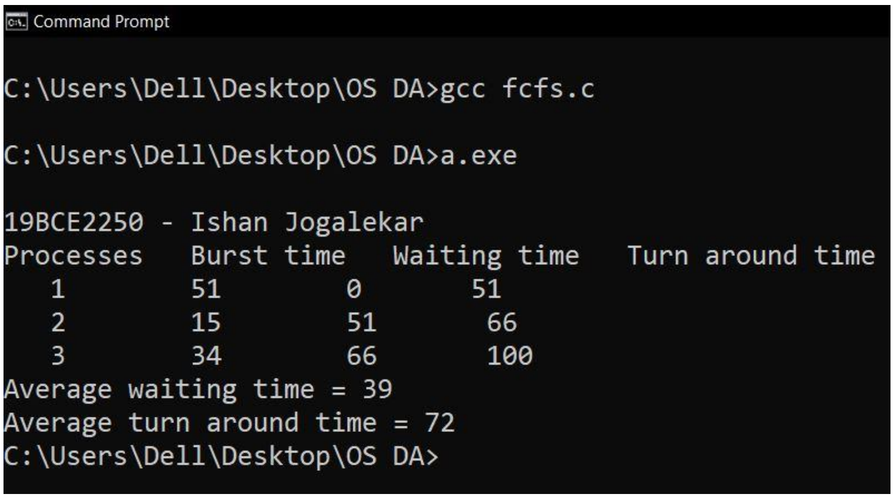
5- Average turn around time =
total_turn_around_time / no_of_processes

Code :-

```
#include<stdio.h>

void findWaitingTime(int processes[], int n,int bt[], int wt[])
{

   wt[0] = 0;


   for (int  i = 1; i < n ; i++ )
      wt[i] =  bt[i-1] + wt[i-1] ;
}


void findTurnAroundTime( int processes[], int n,
                 int bt[], int wt[], int tat[])
{


   for (int  i = 0; i < n ; i++)
      tat[i] = bt[i] + wt[i];
}

void findavgTime( int processes[], int n, int bt[])
{
   int wt[n], tat[n], total_wt = 0, total_tat = 0;


   findWaitingTime(processes, n, bt, wt);


   findTurnAroundTime(processes, n, bt, wt, tat);
```

```
    printf("Processes   Burst time   Waiting time   Turn around time\n");



    for (int  i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("   %d ",(i+1));
        printf("       %d ", bt[i] );
        printf("       %d",wt[i] );
        printf("       %d\n",tat[i] );
    }
    int s=(float)total_wt / (float)n;
    int t=(float)total_tat / (float)n;
    printf("Average waiting time = %d",s);
    printf("\n");
    printf("Average turn around time = %d ",t);
}


int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar\n");

    int processes[] = { 2,10 , 13};
    int n = sizeof processes / sizeof processes[0];


    int  burst_time[] = {51, 15, 34};

    findavgTime(processes, n,  burst_time);
    return 0;
}
```

<u>Output</u> :-



```
C:\Users\Dell\Desktop\OS DA>gcc fcfs.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Processes   Burst time   Waiting time   Turn around time
   1        51        0       51
   2        15        51       66
   3        34        66       100
Average waiting time = 39
Average turn around time = 72
C:\Users\Dell\Desktop\OS DA>
```

2. SJF

Algorithm :-

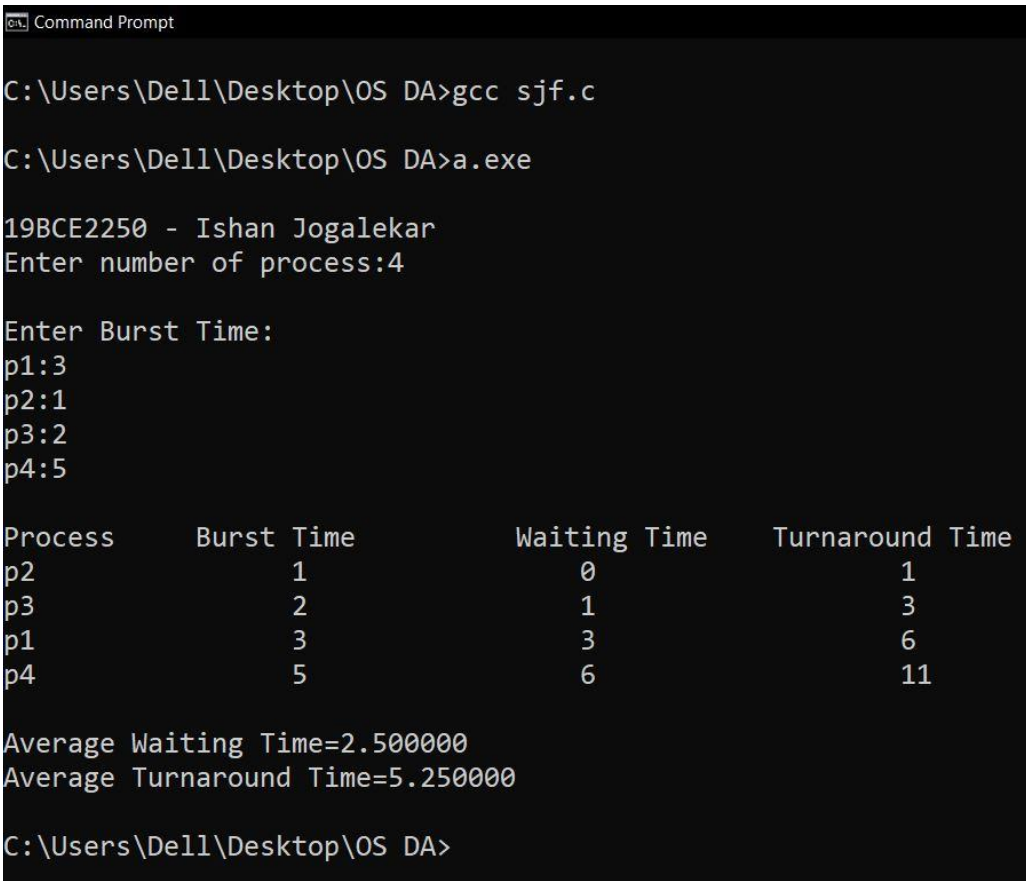
1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

Code :-

```
#include<stdio.h>

void main()
{
    printf("\n19BCE2250 - Ishan Jogalekar");
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }


    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
```

```
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output :-



```
Command Prompt

C:\Users\Dell\Desktop\OS DA>gcc sjf.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Enter number of process:4

Enter Burst Time:
p1:3
p2:1
p3:2
p4:5

Process     Burst Time          Waiting Time    Turnaround Time
p2                1                  0                  1
p3                2                  1                  3
p1                3                  3                  6
p4                5                  6                  11

Average Waiting Time=2.500000
Average Turnaround Time=5.250000

C:\Users\Dell\Desktop\OS DA>
```

3.Priority non pre-emptive

Algorithm :-

- **Step 1** : Input the number of processes required to be scheduled using Non-Pre-emptive Priority Scheduling Algorithm, burst time for each process, arrival time and there respective scheduling priority.
- **Step 2** : Using enhanced bubble sort technique, sort the all given processes in ascending order according to arrival

time and if two or more processes having same arrival time then processes are sorted according to their priorities **(low value represents high priority)** in a ready queue.

- **Step 3** : Calculate the Finish Time, Turn Around Time and Waiting Time for each process which in turn help to calculate Average Waiting Time and Average Turn Around Time required by CPU to schedule given set of processes.
    - **Step 3.1** : for i = 0, Finish Time $_{T\,0}$ = Arrival Time $_{T\,0}$ + Burst Time $_{T\,0}$
    - **Step 3.2** : for i >= 1, Finish Time $_{T\,i}$ = Burst Time $_{T\,i}$ + Finish Time $_{T\,i-1}$
    - **Step 3.3** : for i = 0, Turn Around Time $_{T\,0}$ = Finish Time $_{T\,0}$ - Arrival Time $_{T\,0}$
    - **Step 3.4** : for i >= 1, Turn Around Time $_{T\,i}$ = Finish Time $_{T\,i}$ - Arrival Time $_{T\,i}$
    - **Step 3.5** : for i = 0, Waiting Time $_{T\,0}$ = Turn Around Time $_{T\,0}$ - Burst Time $_{T\,0}$
    - **Step 3.6** : for i >= 1, Waiting Time $_{T\,i}$ = Turn Around Time $_{T\,i}$ - Burst Time $_{T\,i-1}$
- **Step 4** : Process with less arrival time (not necessary this process will have highest priority) comes first and gets scheduled first by the CPU.
- **Step 5** : Calculate the Average Waiting Time and Average Turn Around Time.
- **Step 6** : Stop.

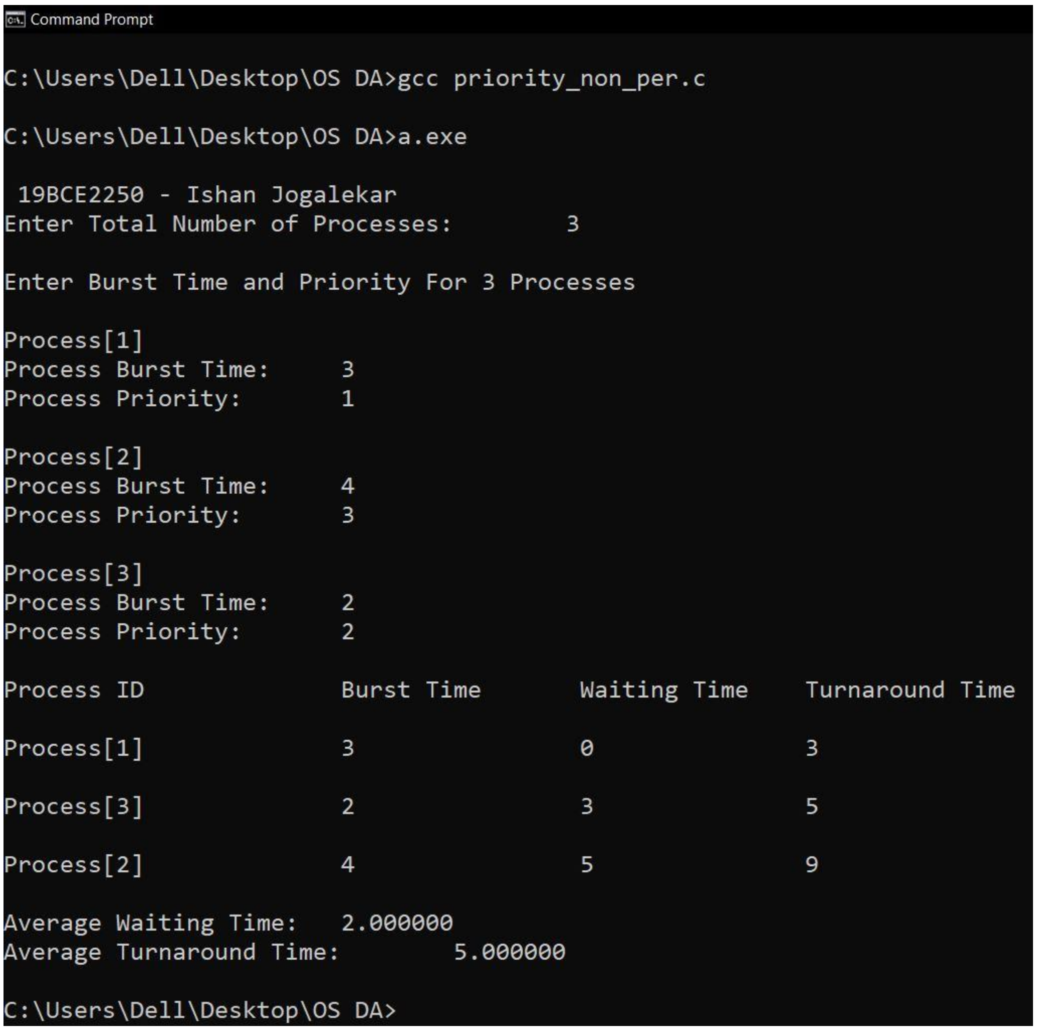
Code :-

```
#include<stdio.h>

int main()
{
	printf("\n 19BCE2250 - Ishan Jogalekar\n");
	int burst_time[20], process[20], waiting_time[20], 
turnaround_time[20], priority[20];
	int i, j, limit, sum = 0, position, temp;
	float average_wait_time, average_turnaround_time;
```

```
printf("Enter Total Number of Processes:\t");
scanf("%d", &limit);
printf("\nEnter Burst Time and Priority For %d Processes\n", limit);
for(i = 0; i < limit; i++)
{
      printf("\nProcess[%d]\n", i + 1);
      printf("Process Burst Time:\t");
      scanf("%d", &burst_time[i]);
      printf("Process Priority:\t");
      scanf("%d", &priority[i]);
      process[i] = i + 1;
}
for(i = 0; i < limit; i++)
{
      position = i;
      for(j = i + 1; j < limit; j++)
      {
            if(priority[j] < priority[position])
            {
                  position = j;
            }
      }
      temp = priority[i];
      priority[i] = priority[position];
      priority[position] = temp;
      temp = burst_time[i];
      burst_time[i] = burst_time[position];
      burst_time[position] = temp;
      temp = process[i];
      process[i] = process[position];
      process[position] = temp;
}
waiting_time[0] = 0;
for(i = 1; i < limit; i++)
{
      waiting_time[i] = 0;
      for(j = 0; j < i; j++)
      {
            waiting_time[i] = waiting_time[i] + burst_time[j];
      }
      sum = sum + waiting_time[i];
}
average_wait_time = sum / limit;
```

```
	sum = 0;
	printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround 
Time\n");
	for(i = 0; i < limit; i++)
	{
		turnaround_time[i] = burst_time[i] + waiting_time[i];
		sum = sum + turnaround_time[i];
		printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i], 
burst_time[i], waiting_time[i], turnaround_time[i]);
	}
	average_turnaround_time = sum / limit;
	printf("\nAverage Waiting Time:\t%f", average_wait_time);
	printf("\nAverage Turnaround Time:\t%f\n", 
average_turnaround_time);
	return 0;
}
```

Output :-



Command Prompt

```
C:\Users\Dell\Desktop\OS DA>gcc priority_non_per.c

C:\Users\Dell\Desktop\OS DA>a.exe

 19BCE2250 - Ishan Jogalekar
Enter Total Number of Processes:        3

Enter Burst Time and Priority For 3 Processes

Process[1]
Process Burst Time:     3
Process Priority:       1

Process[2]
Process Burst Time:     4
Process Priority:       3

Process[3]
Process Burst Time:     2
Process Priority:       2

Process ID              Burst Time       Waiting Time    Turnaround Time

Process[1]              3                0               3

Process[3]              2                3               5

Process[2]              4                5               9

Average Waiting Time:   2.000000
Average Turnaround Time:        5.000000

C:\Users\Dell\Desktop\OS DA>
```

(b) Implement the various process scheduling algorithms such as Priority, Round Robin (preemptive). (Medium)

Ans :

1. **Priority Pre-emptive**

Algorithm :-
1- First input the processes with their burst time and priority.

2- Sort the processes, burst time and priority according to the priority.

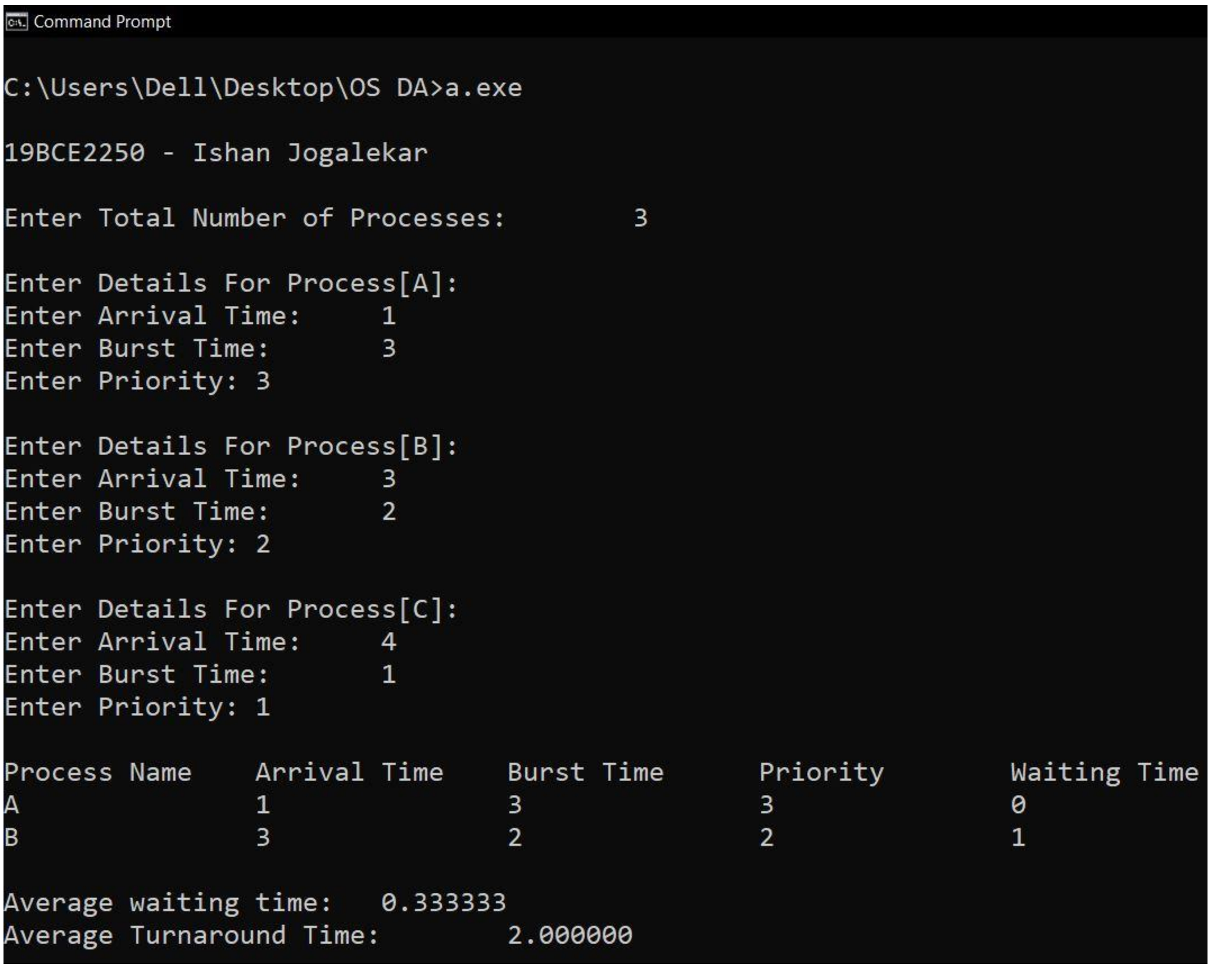
3- Now simply apply FCFS algorithm.

Code :-
```
#include<stdio.h>

struct process
{
      char process_name;
      int arrival_time, burst_time, ct, waiting_time, turnaround_time,
priority;
      int status;
}process_queue[10];

int limit;

void Arrival_Time_Sorting()
{
      struct process temp;
      int i, j;
      for(i = 0; i < limit - 1; i++)
      {
            for(j = i + 1; j < limit; j++)
            {
                  if(process_queue[i].arrival_time >
process_queue[j].arrival_time)
```

```
                {
                        temp = process_queue[i];
                        process_queue[i] = process_queue[j];
                        process_queue[j] = temp;
                }
            }
        }
}

void main()
{
        printf("\n19BCE2250 - Ishan Jogalekar\n");
        int i, time = 0, burst_time = 0, largest;
        char c;
        float wait_time = 0, turnaround_time = 0, average_waiting_time,
average_turnaround_time;
        printf("\nEnter Total Number of Processes:\t");
        scanf("%d", &limit);
        for(i = 0, c = 'A'; i < limit; i++, c++)
        {
                process_queue[i].process_name = c;
                printf("\nEnter Details For Process[%C]:\n",
process_queue[i].process_name);
                printf("Enter Arrival Time:\t");
                scanf("%d", &process_queue[i].arrival_time );
                printf("Enter Burst Time:\t");
                scanf("%d", &process_queue[i].burst_time);
                printf("Enter Priority:\t");
                scanf("%d", &process_queue[i].priority);
                process_queue[i].status = 0;
                burst_time = burst_time + process_queue[i].burst_time;
        }
        Arrival_Time_Sorting();
        process_queue[9].priority = -9999;
        printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting
Time");
        for(time = process_queue[0].arrival_time; time < burst_time;)
        {
                largest = 9;
                for(i = 0; i < limit; i++)
                {
```

```
                if(process_queue[i].arrival_time <= time &&
process_queue[i].status != 1 && process_queue[i].priority >
process_queue[largest].priority)
                {
                     largest = i;
                }
           }
           time = time + process_queue[largest].burst_time;
           process_queue[largest].ct = time;
           process_queue[largest].waiting_time = process_queue[largest].ct
- process_queue[largest].arrival_time -
process_queue[largest].burst_time;
           process_queue[largest].turnaround_time =
process_queue[largest].ct - process_queue[largest].arrival_time;
           process_queue[largest].status = 1;
           wait_time = wait_time + process_queue[largest].waiting_time;
           turnaround_time = turnaround_time +
process_queue[largest].turnaround_time;
           printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d",
process_queue[largest].process_name,
process_queue[largest].arrival_time, process_queue[largest].burst_time,
process_queue[largest].priority, process_queue[largest].waiting_time);
      }
      average_waiting_time = wait_time / limit;
      average_turnaround_time = turnaround_time / limit;
      printf("\n\nAverage waiting time:\t%f\n", average_waiting_time);
      printf("Average Turnaround Time:\t%f\n", average_turnaround_time);
}
```

Output :-



```
C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar

Enter Total Number of Processes:        3

Enter Details For Process[A]:
Enter Arrival Time:     1
Enter Burst Time:       3
Enter Priority: 3

Enter Details For Process[B]:
Enter Arrival Time:     3
Enter Burst Time:       2
Enter Priority: 2

Enter Details For Process[C]:
Enter Arrival Time:     4
Enter Burst Time:       1
Enter Priority: 1

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
A               1               3               3               0
B               3               2               2               1

Average waiting time:   0.333333
Average Turnaround Time:        2.000000
```

2.Round robin :

Algorithm :-

```
- Create an array rem_bt[] to keep track of remaining
    burst time of processes. This array is initially a
    copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times
     of processes. Initialize this array as 0.
3- Initialize time : t = 0

4- Keep traversing the all processes while all processes
    are not done. Do following for i'th process if it is
    not done yet.
    a- If rem_bt[i] > quantum
        (i) t = t + quantum
        (ii) bt_rem[i] -= quantum;
```

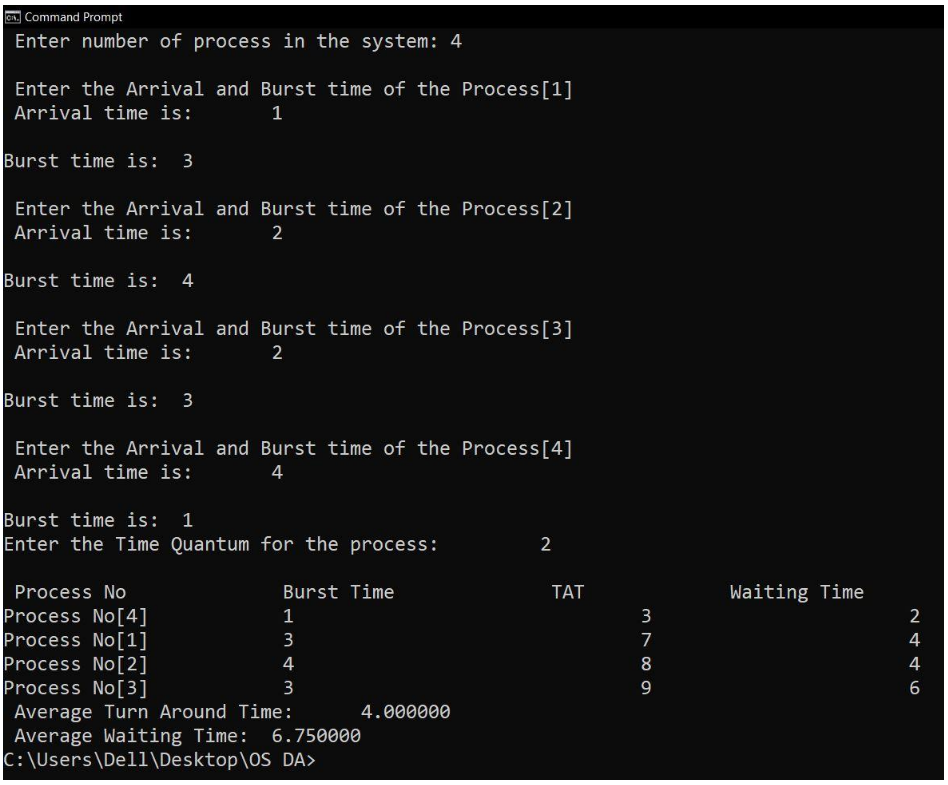
```
c- Else // Last cycle for this process
    (i) t = t + bt_rem[i];
    (ii) wt[i] = t - bt[i]
    (ii) bt_rem[i] = 0; // This process is over
```

Code :-

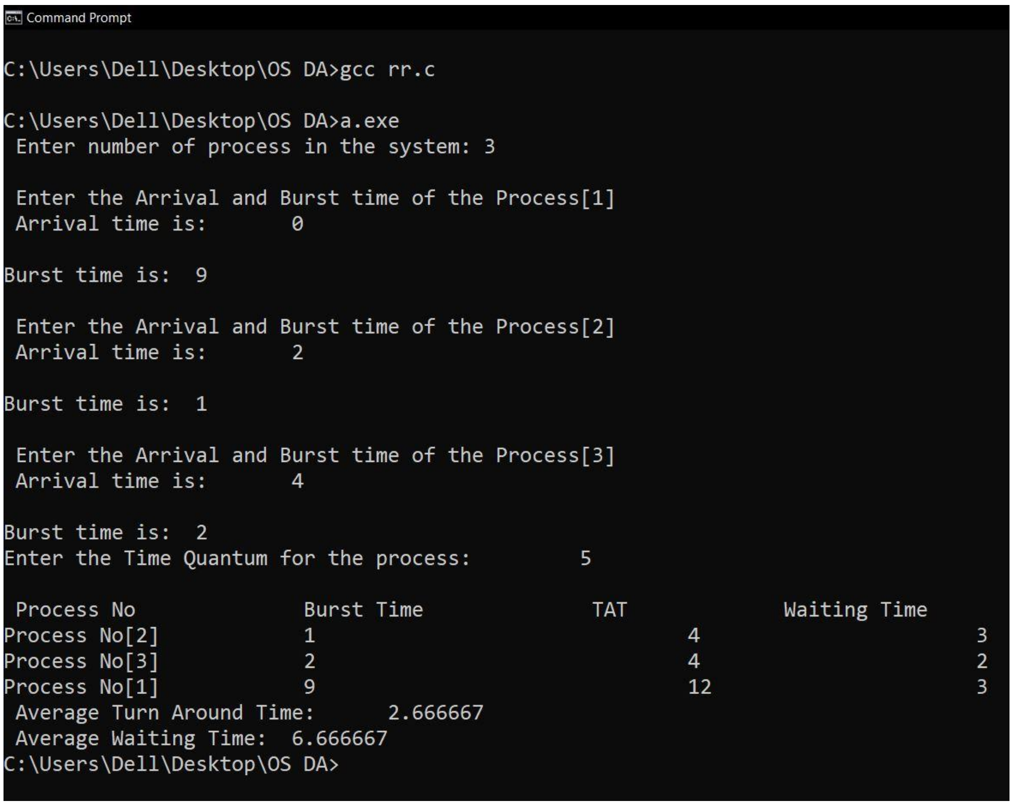
```
#include<stdio.h>
void main()
{

   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10],
temp[10];
   float avgwt, avgtat;
   printf(" Enter number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;


   for(i=0; i<NOP; i++)
   {
      printf("\n Enter the Arrival and Burst time of the Process[%d]\n",
i+1);
      printf(" Arrival time is: \t");
      scanf("%d", &at[i]);
      printf(" \nBurst time is: \t");
      scanf("%d", &bt[i]);
      temp[i] = bt[i];
   }

   printf("Enter the Time Quantum for the process: \t");
   scanf("%d", &quant);

   printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
   for(sum=0, i = 0; y!=0; )
   {
      if(temp[i] <= quant && temp[i] > 0)
      {
         sum = sum + temp[i];
         temp[i] = 0;
```

```
            count=1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - quant;
            sum = sum + quant;
        }
        if(temp[i]==0 && count==1)
        {
            y--;
            printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-
at[i], sum-at[i]-bt[i]);
            wt = wt+sum-at[i]-bt[i];
            tat = tat+sum-at[i];
            count =0;
        }
        if(i==NOP-1)
        {
            i=0;
        }

        else if(at[i+1]<=sum)
        {
            i++;
        }
        else
        {
            i=0;
        }
    }

        avgwt = wt * 1.0/NOP;
        avgtat = tat * 1.0/NOP;
        printf("\n Average Turn Around Time: \t%f", avgwt);
        printf("\n Average Waiting Time: \t%f", avgtat);
}
```

Output :-



```
 Enter number of process in the system: 4

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       1

Burst time is:  3

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       2

Burst time is:  4

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       2

Burst time is:  3

 Enter the Arrival and Burst time of the Process[4]
 Arrival time is:       4

Burst time is:  1
Enter the Time Quantum for the process:         2

 Process No              Burst Time              TAT             Waiting Time
Process No[4]            1                               3                       2
Process No[1]            3                               7                       4
Process No[2]            4                               8                       4
Process No[3]            3                               9                       6
 Average Turn Around Time:      4.000000
 Average Waiting Time:  6.750000
C:\Users\Dell\Desktop\OS DA>
```



```

C:\Users\Dell\Desktop\OS DA>gcc rr.c

C:\Users\Dell\Desktop\OS DA>a.exe
 Enter number of process in the system: 3

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0

Burst time is:  9

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       2

Burst time is:  1

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       4

Burst time is:  2
Enter the Time Quantum for the process:         5

 Process No              Burst Time              TAT             Waiting Time
Process No[2]            1                       4                       3
Process No[3]            2                       4                       2
Process No[1]            9                       12                      3
 Average Turn Around Time:      2.666667
 Average Waiting Time:  6.666667
C:\Users\Dell\Desktop\OS DA>
```

(c) Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency, he /she should be attended immediately before other patients, which may increase the waiting time of other patients. If you are given this problem with the following algorithms how would you devise an effective scheduling so that it optimizes the overall performance such as minimizing the waiting time of all patients. [Single queue or multi-level queue can be used]. Consider the availability of single and multiple doctors • Assign top priority for patients with emergency case, women, children, elders, and youngsters. • Patients coming for review may take less time than others. This can be taken into account while using SJF.

1. FCFS
2. SJF (primitive and non-pre-emptive) (High)

Ans :

1. FCFS :

<u>Algorithm</u> :-

```
1- Input the processes along with their burst time(bt)
   and arrival time(at)

2- Find waiting time for all other processes i.e. for
   a given process i:
   wt[i] = (bt[0] + bt[1] +...... bt[i-1]) - at[i]

3- Now find turn around time
    = waiting_time + burst_time for all processes

4- Average waiting time =
   total_waiting_time / no_of_processes
```

5- Average turn around time =
  total_turn_around_time / no_of_processes

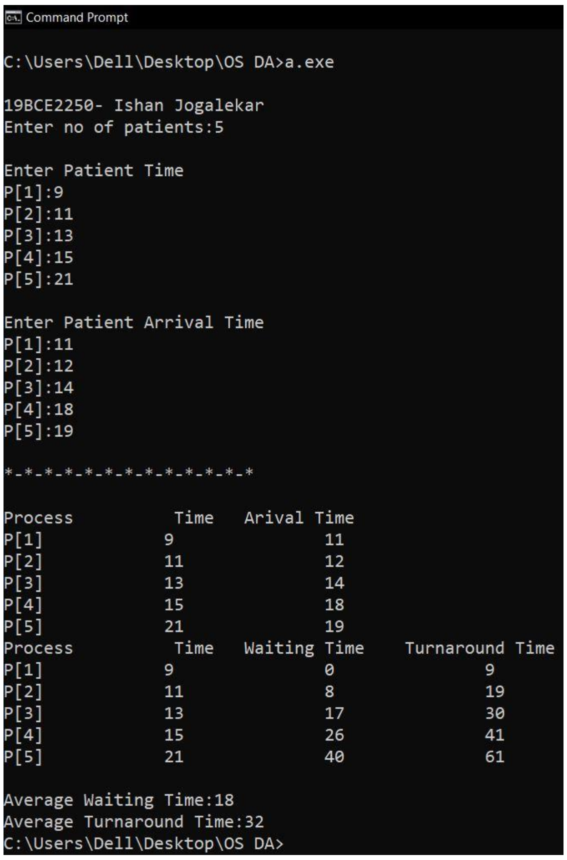
Code :-

```
#include<iostream>
using namespace std;
struct prodet
{
    int bt;
    int art;
    int ft;
    int tat;
    int wt;
}ar[20] , tmp;
int main()
{
    cout<<"\n19BCE2250- Ishan Jogalekar";
    int n,avwt=0,avtat=0,i,j;
    cout<<"\nEnter no of patients:";
    cin>>n;
    cout<<"\nEnter Patient Time\n";
    for(i=0;i<n;i++)
    {
        cout<<"P["<<i+1<<"]:";
        cin>>ar[i].bt;
    }
    cout<<"\nEnter Patient Arrival Time\n";
```

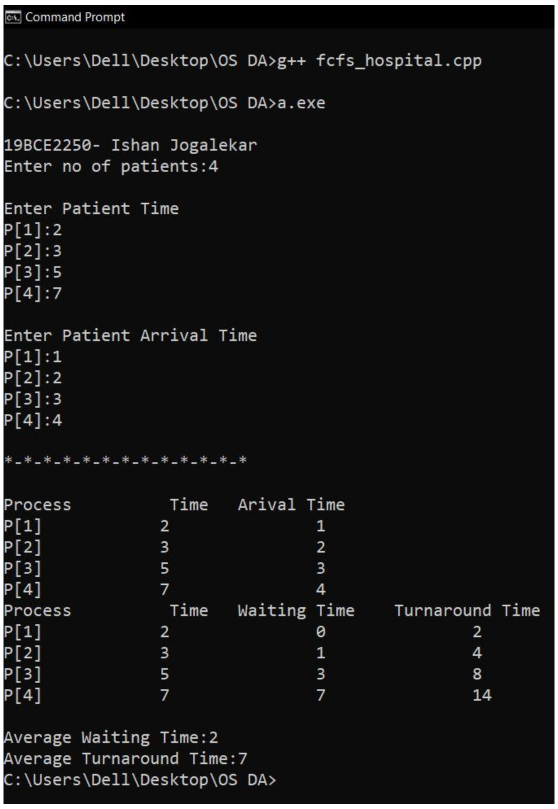
```
for(i=0;i<n;i++)
{
    cout<<"P["<<i+1<<"]:";
    cin>>ar[i].art;
}
for(int i=0 ; i < n-1; i++)
{
    for(j = 0; j < n-1-i; j++) {
    if(ar[j].art>ar[j+1].art)
     {
        tmp = ar[j];
        ar[j]=ar[j+1];
        ar[j+1]=tmp;
     }
    }
}
ar[0].ft = ar[0].bt + ar[0].art;
for(i=1;i<n;i++)
{
    ar[i].ft = ar[i-1].ft+ar[i].bt;
}
cout<<endl<<"*-*-*-*-*-*-*-*-*-*-*-*-*-*"<<endl;
cout<<"\nProcess\t\t Time\tArival Time";
for(i=0;i<n;i++)
{
    cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].art;
}
```

```
    cout<<"\nProcess\t\t Time\tWaiting Time\tTurnaround Time";
    for(i=0;i<n;i++)
    {
        ar[i].tat=ar[i].ft - ar[i].art;
        ar[i].wt = ar[i].tat - ar[i].bt;
        avwt+=ar[i].wt;
        avtat+=ar[i].tat;

cout<<"\nP["<<i+1<<"]"<<"\t\t"<<ar[i].bt<<"\t\t"<<ar[i].wt<<"\t\t"<<ar[i].tat;
    }
    avwt/=i;
    avtat/=i;
    cout<<"\n\nAverage Waiting Time:"<<avwt;
    cout<<"\nAverage Turnaround Time:"<<avtat;
    return 0;
}
```

Output :-



```
C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250- Ishan Jogalekar
Enter no of patients:5

Enter Patient Time
P[1]:9
P[2]:11
P[3]:13
P[4]:15
P[5]:21

Enter Patient Arrival Time
P[1]:11
P[2]:12
P[3]:14
P[4]:18
P[5]:19

*-*-*-*-*-*-*-*-*-*-*-*-*

Process          Time   Arival Time
P[1]            9               11
P[2]            11              12
P[3]            13              14
P[4]            15              18
P[5]            21              19
Process          Time   Waiting Time    Turnaround Time
P[1]            9               0               9
P[2]            11              8               19
P[3]            13              17              30
P[4]            15              26              41
P[5]            21              40              61

Average Waiting Time:18
Average Turnaround Time:32
C:\Users\Dell\Desktop\OS DA>
```



```
C:\Users\Dell\Desktop\OS DA>g++ fcfs_hospital.cpp

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250- Ishan Jogalekar
Enter no of patients:4

Enter Patient Time
P[1]:2
P[2]:3
P[3]:5
P[4]:7

Enter Patient Arrival Time
P[1]:1
P[2]:2
P[3]:3
P[4]:4

*-*-*-*-*-*-*-*-*-*-*-*-*

Process          Time   Arival Time
P[1]            2               1
P[2]            3               2
P[3]            5               3
P[4]            7               4
Process          Time   Waiting Time    Turnaround Time
P[1]            2               0               2
P[2]            3               1               4
P[3]            5               3               8
P[4]            7               7               14

Average Waiting Time:2
Average Turnaround Time:7
C:\Users\Dell\Desktop\OS DA>
```

2.

a. SJF pre-emptive

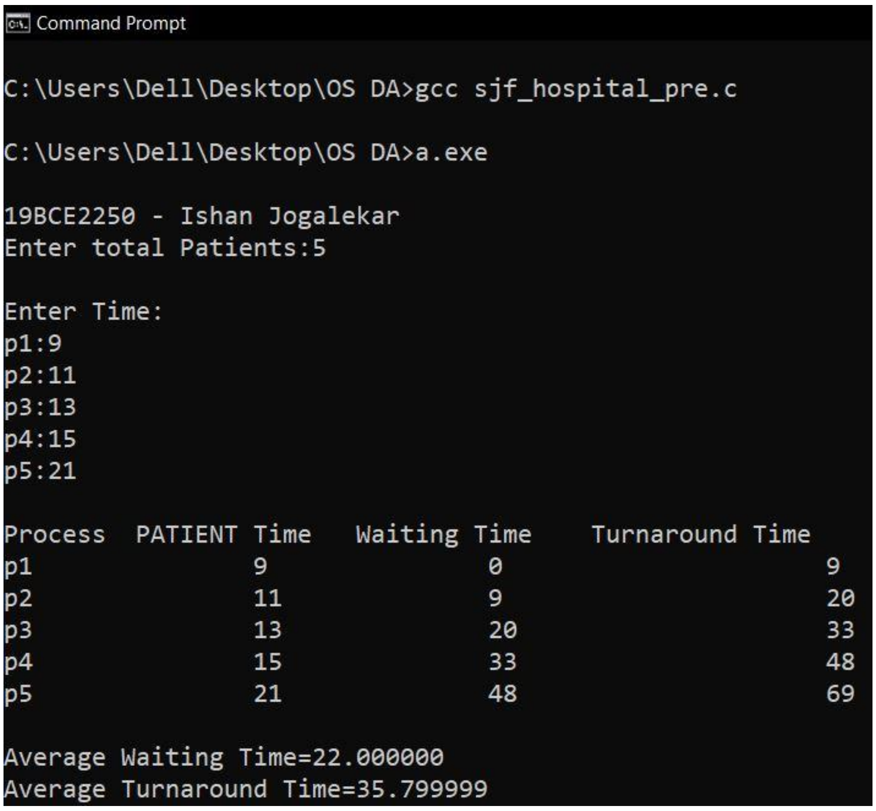
Algorithm :

1- Traverse until all process gets completely executed.
a) Find process with minimum remaining time at every single time lap.
b) Reduce its time by 1.
c) Check if its remaining time becomes 0
d) Increment the counter of process completion.
e) Completion time of current process = current_time +1;
e) Calculate waiting time for each completed process.
wt[i]= Completion time - arrival_time-burst_time
f)Increment time lap by one.
2- Find turnaround time (waiting_time+burst_time).

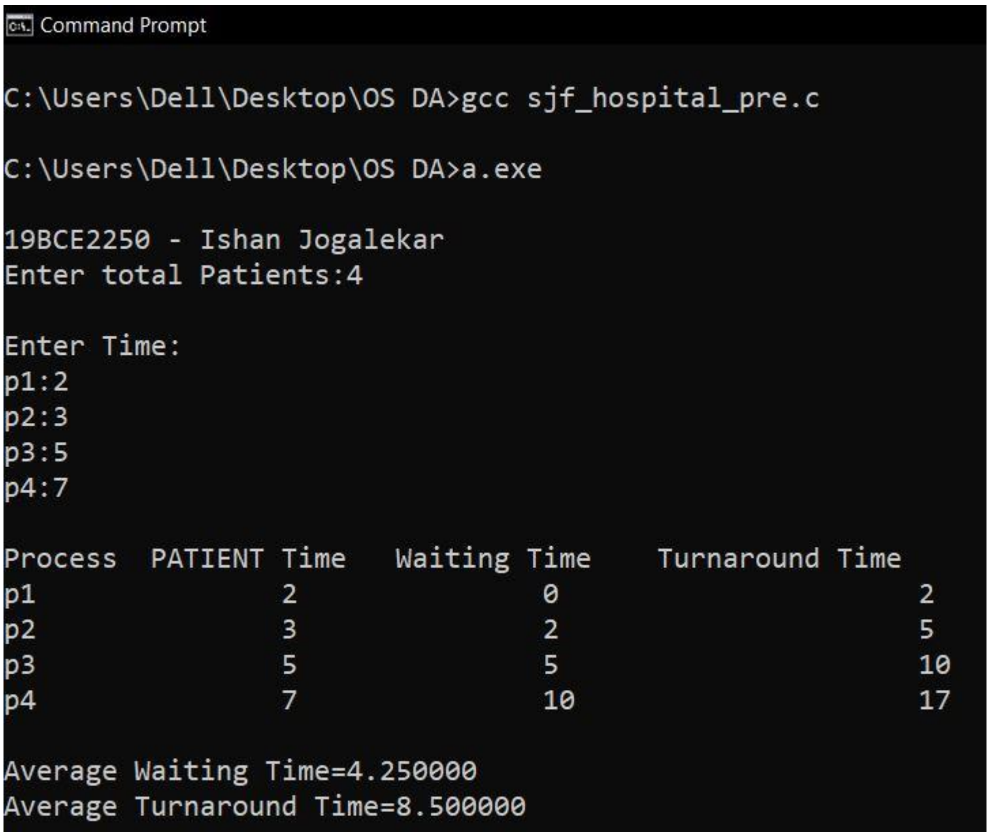
Code :

```
#include<stdio.h>
void main()
{
    printf("\n19BCE2250 - Ishan Jogalekar");
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("\nEnter total Patients:");
    scanf("%d",&n);
    printf("\nEnter Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
            pos=j;
        }
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
        {
            wt[i]+=bt[j];
        }
        total+=wt[i];
    }
    avg_wt=(float)total/n;
    total=0;
    printf("\nProcess\t PATIENT Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output :-



```
Command Prompt

C:\Users\Dell\Desktop\OS DA>gcc sjf_hospital_pre.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Enter total Patients:5

Enter Time:
p1:9
p2:11
p3:13
p4:15
p5:21

Process  PATIENT Time   Waiting Time    Turnaround Time
p1               9              0                      9
p2               11             9                      20
p3               13             20                     33
p4               15             33                     48
p5               21             48                     69

Average Waiting Time=22.000000
Average Turnaround Time=35.799999
```



```
Command Prompt

C:\Users\Dell\Desktop\OS DA>gcc sjf_hospital_pre.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Enter total Patients:4

Enter Time:
p1:2
p2:3
p3:5
p4:7

Process  PATIENT Time   Waiting Time    Turnaround Time
p1               2              0                      2
p2               3              2                      5
p3               5              5                      10
p4               7              10                     17

Average Waiting Time=4.250000
Average Turnaround Time=8.500000
```

b. SJF non- non-pre-emptive

Algorithm :-

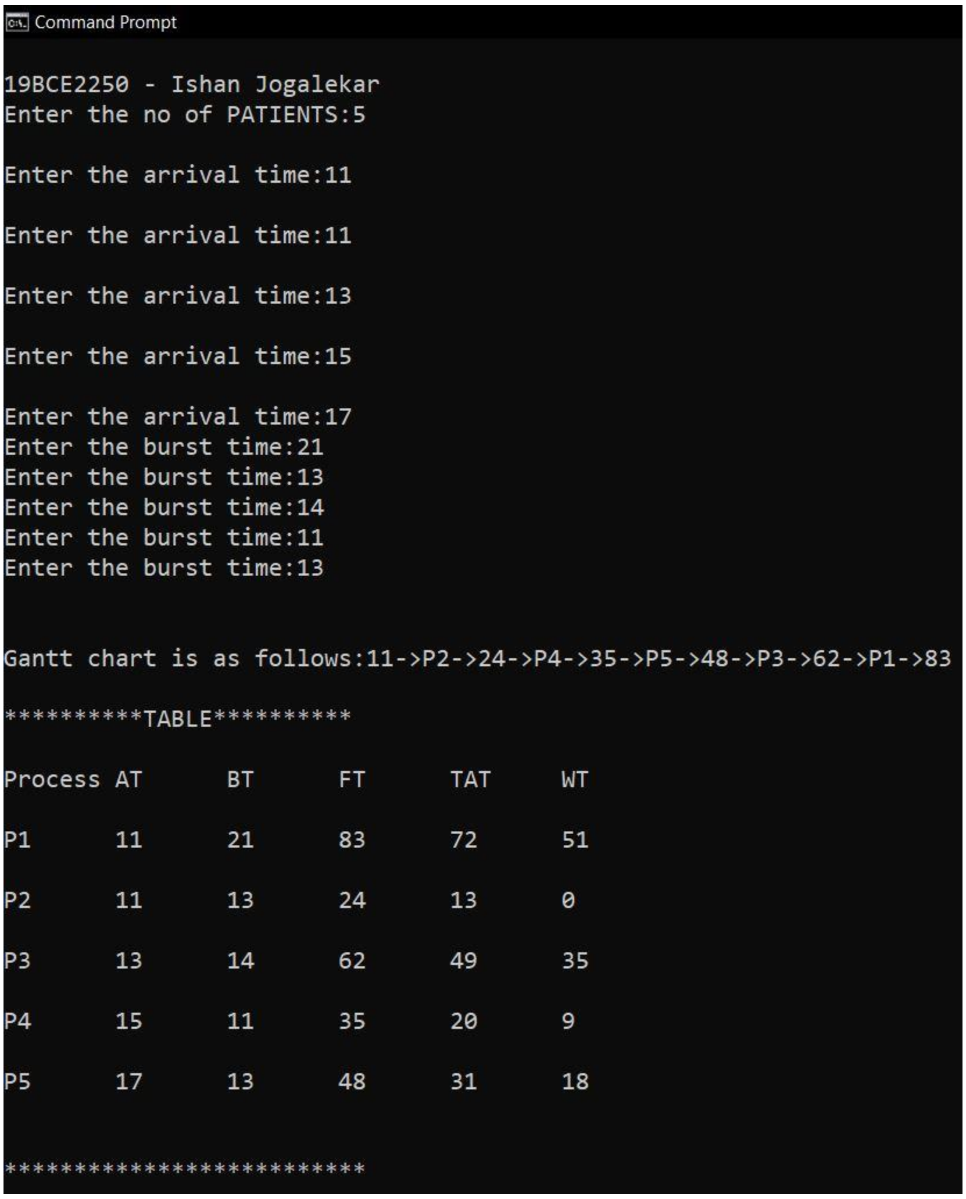
1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

Code :-

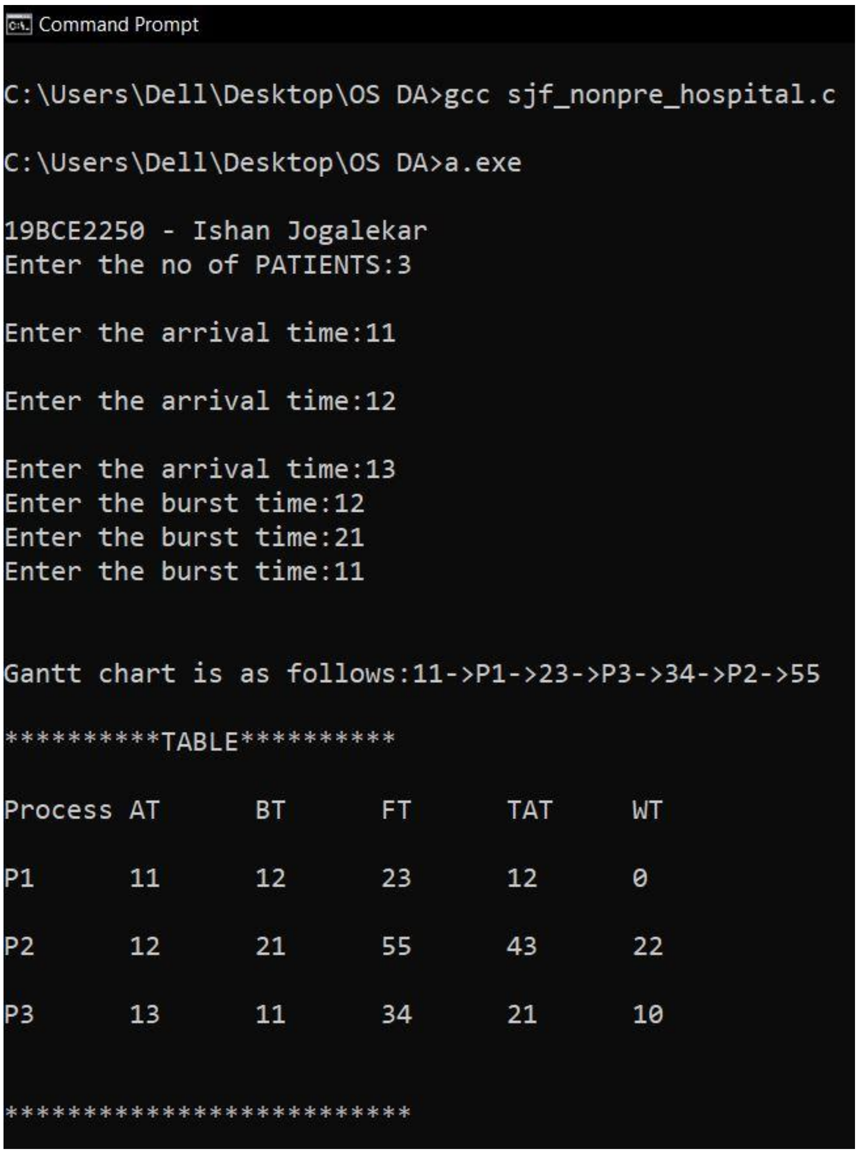
```
#include<stdio.h>
typedef struct nonpresjf
{
int at,bt,ft,tat,wt;
}nonpresjf;
nonpresjf p[20],p1[20];
int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar");
    int i,limit,nextval,m,min,n;
    p[0].wt=p[0].tat=0;
    printf("\nEnter the no of PATIENTS:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\nEnter the arrival time:");
        scanf("%d",&p[i].at);
    }
    limit=p[1].at;
    for(i=1;i<=n;i++)
    {
        printf("Enter the burst time:");
        scanf("%d",&p[i].bt);
    }

   for(i=1;i<=n;i++)
```

```
     limit+=p[i].bt;
   for(i=1;i<=n;i++)
     p1[i]=p[i];
   printf("\n\nGantt chart is as follows:");
   printf("%d",p[1].at);
   nextval=p[1].at;
   m=1;
   do
   {
       min = 9999;
       for(i=1;p[i].at<=nextval && i<=n ;i++)
         if(p[i].bt<min && p[i].bt>0)
         {
             m=i;
             min=p[i].bt;
         }
        nextval+=p[m].bt;
        p[m].bt=0;
        printf("->P%d->%d",m,nextval);
        if(p[m].bt==0)
        {
            p[m].ft=nextval;
            p[m].tat=p[m].ft-p[m].at;
            p[m].wt=p[m].tat-p1[m].bt;
            p[0].tat+=p[m].tat;
            p[0].wt+=p[m].wt;
        }
    }while(nextval<limit);
    p[0].tat=p[0].tat/n;
    p[0].wt=p[0].wt/n;
    printf("\n\n**********TABLE**********\n");
    printf("\nProcess\tAT\tBT\tFT\tTAT\tWT\n");
    for(i=1;i<=n;i++)

printf("\nP%d\t%d\t%d\t%d\t%d\t%d\n",i,p[i].at,p1[i].bt,p[i].ft,p[i].tat,p[i].wt)
;
    printf("\n\n************************\n");
    return 0;
}
```

Output :-



```
19BCE2250 - Ishan Jogalekar
Enter the no of PATIENTS:5

Enter the arrival time:11

Enter the arrival time:11

Enter the arrival time:13

Enter the arrival time:15

Enter the arrival time:17
Enter the burst time:21
Enter the burst time:13
Enter the burst time:14
Enter the burst time:11
Enter the burst time:13


Gantt chart is as follows:11->P2->24->P4->35->P5->48->P3->62->P1->83

**********TABLE**********

Process AT      BT      FT      TAT     WT

P1      11      21      83      72      51

P2      11      13      24      13      0

P3      13      14      62      49      35

P4      15      11      35      20      9

P5      17      13      48      31      18


**************************
```



```
Command Prompt

C:\Users\Dell\Desktop\OS DA>gcc sjf_nonpre_hospital.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Enter the no of PATIENTS:3

Enter the arrival time:11

Enter the arrival time:12

Enter the arrival time:13
Enter the burst time:12
Enter the burst time:21
Enter the burst time:11


Gantt chart is as follows:11->P1->23->P3->34->P2->55

**********TABLE**********

Process AT      BT      FT      TAT     WT

P1      11      12      23      12      0

P2      12      21      55      43      22

P3      13      11      34      21      10


*************************
```

(d) Simulate with a program to provide deadlock avoidance of Banker's Algorithm including Safe state and additional resource request (High).

Ans :

Algorithm :-

Available :

- It is a 1-d array of size **‘m’** indicating the number of available resources of each type.
- Available[j] = k means there are **‘k’** instances of resource type **R$_j$**

Max :

- It is a 2-d array of size ‘**n*m’** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process **P$_i$** may request at most **‘k’** instances of resource type **R$_j$**.

Allocation :

- It is a 2-d array of size **‘n*m’** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process **P$_i$** is currently allocated **‘k’** instances of resource type **R$_j$**

Need :

- It is a 2-d array of size **‘n*m’** that indicates the remaining resource need of each process.
- Need [i, j] = k means process **P$_i$** currently need **‘k’** instances of resource type **R$_j$** for its execution.
- Need [i, j] = Max [i, j] – Allocation [i, j]

Code :-

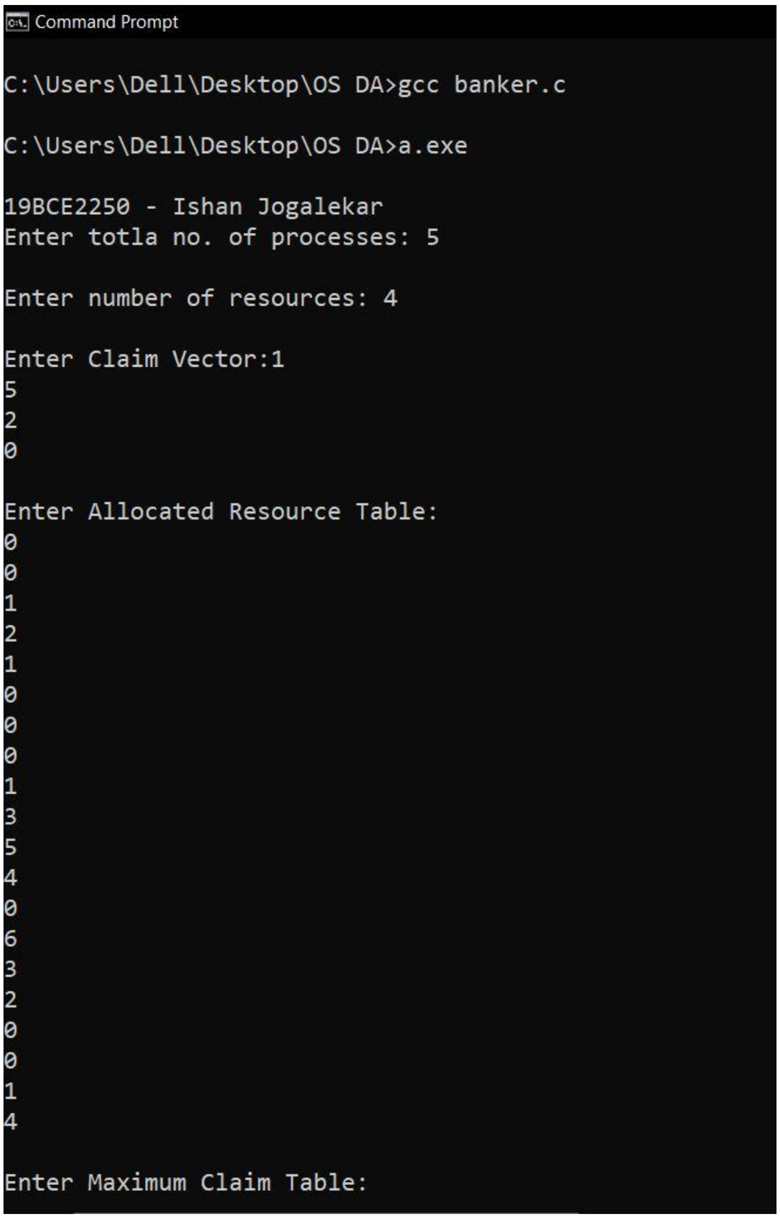
a.

```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
    printf("\n19BCE2250 - Ishan Jogalekar");
printf("\nEnter totla no. of processes: ");
scanf("%d", &processes);

for (i = 0; i < processes; i++)
{
running[i] = 1;
counter++;
}

printf("\nEnter number of resources: ");
scanf("%d", &resources);

printf("\nEnter Claim Vector:");
for (i = 0; i < resources; i++)
{
        scanf("%d", &maxres[i]);
}

printf("\nEnter Allocated Resource Table:\n");
for (i = 0; i < processes; i++)
{
        for(j = 0; j < resources; j++)
         {
                  scanf("%d", &current[i][j]);
}
}

printf("\nEnter Maximum Claim Table:\n");
for (i = 0; i < processes; i++)
{
for(j = 0; j < resources; j++)
         {
                  scanf("%d", &maximum_claim[i][j]);
}
}

printf("\nThe Claim Vector is: ");
for (i = 0; i < resources; i++)
{
        printf("\t%d", maxres[i]);
}

printf("\nThe Allocated Resource Table:\n");
```

```
for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
         {
                  printf("\t%d", current[i][j]);
}
         printf("\n");
}

printf("\nThe Maximum Claim Table:\n");
for (i = 0; i < processes; i++)
{
for (j = 0; j < resources; j++)
         {
                 printf("\t%d", maximum_claim[i][j]);
}
printf("\n");
}

for (i = 0; i < processes; i++)
{
for (j = 0; j < resources; j++)
         {
                  allocation[j] += current[i][j];
}
}

printf("\nAllocated resources:");
for (i = 0; i < resources; i++)
{
printf("\t%d", allocation[i]);
}

for (i = 0; i < resources; i++)
{
        available[i] = maxres[i] - allocation[i];
}

printf("\nAvailable resources:");
for (i = 0; i < resources; i++)
{
printf("\t%d", available[i]);
}
```

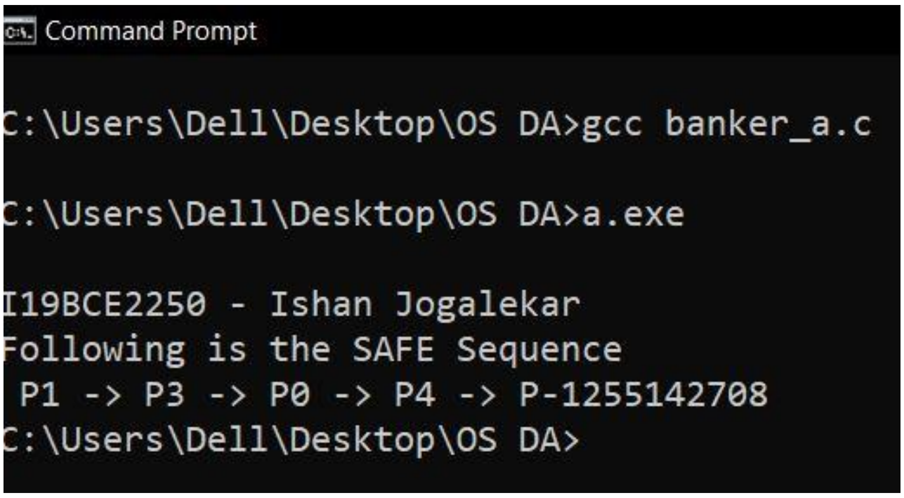
```
        printf("\n");

        while (counter != 0)
        {
        safe = 0;
        for (i = 0; i < processes; i++)
                {
                        if (running[i])
                        {
                        exec = 1;
                        for (j = 0; j < resources; j++)
                                {
                                        if (maximum_claim[i][j] - current[i][j] >
    available[j])
                                        {
                                        exec = 0;
                                        break;
                                        }
                        }
                        if (exec)
                                {
                                        printf("\nProcess%d is executing\n", i + 1);
                                        running[i] = 0;
                                        counter--;
                                        safe = 1;

                                        for (j = 0; j < resources; j++)
                                        {
                                        available[j] += current[i][j];
                                        }
                                       break;
                        }
                        }
        }
        if (!safe)
                {
                        printf("\nThe processes are in unsafe state.\n");
                        break;
        }
                else
                {
                        printf("\nThe process is in safe state");
                        printf("\nAvailable vector:");
                        for (i = 0; i < resources; i++)
                        {
                        printf("\t%d", available[i]);
                        }

                       printf("\n");
        }
        }
        return 0;
}
```

Output :-



Command Prompt

```
C:\Users\Dell\Desktop\OS DA>gcc banker.c

C:\Users\Dell\Desktop\OS DA>a.exe

19BCE2250 - Ishan Jogalekar
Enter totla no. of processes: 5

Enter number of resources: 4

Enter Claim Vector:1
5
2
0


Enter Allocated Resource Table:
0
0
1
2
1
0
0
0
1
3
5
4
0
6
3
2
0
0
1
4

Enter Maximum Claim Table:
```



Command Prompt

```
C:\Users\Dell\Desktop\OS DA>gcc banker_a.c

C:\Users\Dell\Desktop\OS DA>a.exe

I19BCE2250 - Ishan Jogalekar
Following is the SAFE Sequence
 P1 -> P3 -> P0 -> P4 -> P-1255142708
C:\Users\Dell\Desktop\OS DA>
```

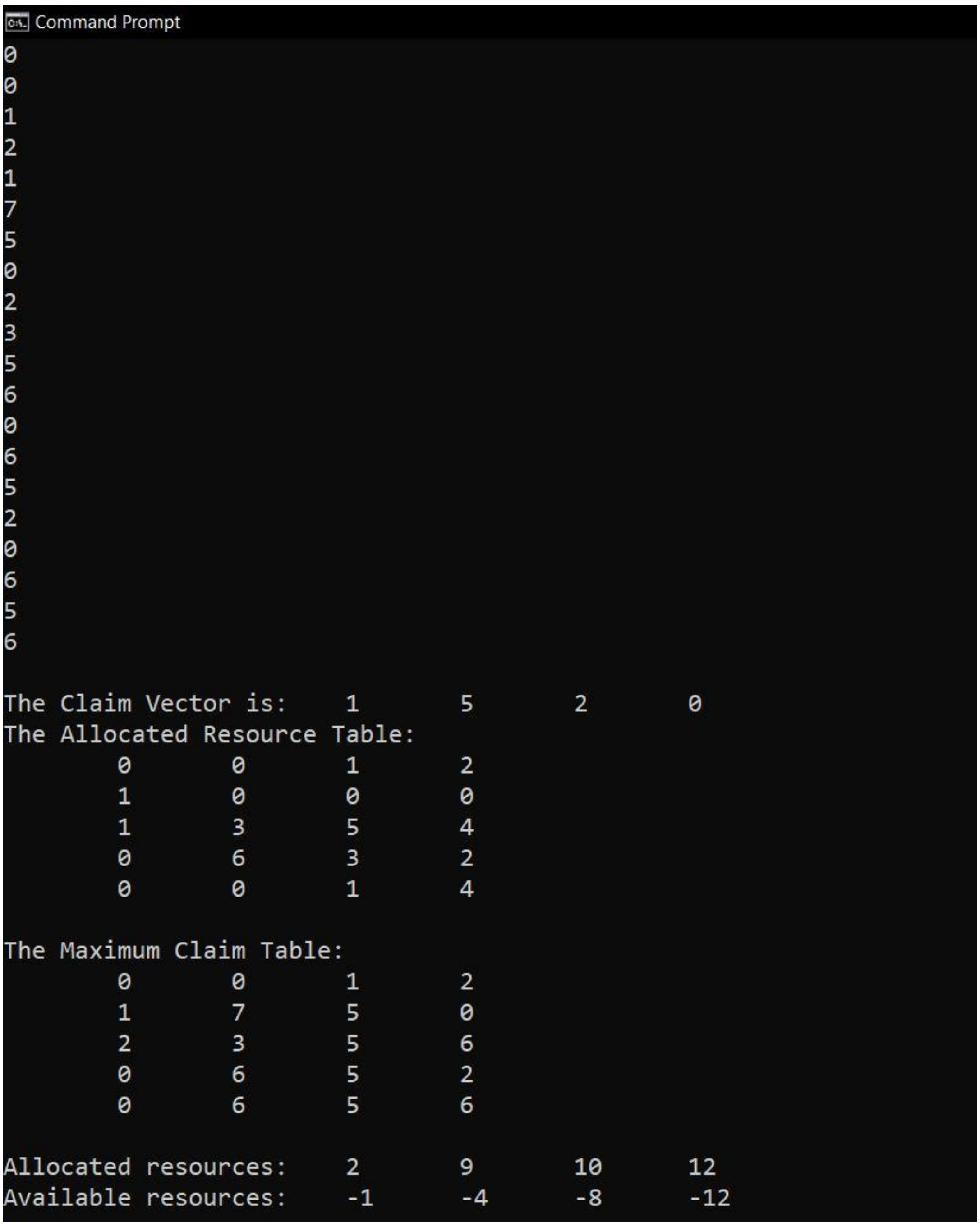
b.

Code :-

```
#include <stdio.h>
int main()
{
        printf("\nI19BCE2250 - Ishan Jogalekar\n");
        int n, m, i, j, k;
        n = 5;
        m = 3;
        int alloc[5][3] = { { 1, 3, 0 },
                                        { 2, 0, 1 },
                                        { 4, 1, 2 },
                                        { 2, 0, 0 },
                                        { 0, 0, 2 } };

        int max[5][3] = { { 7, 5, 1 },
                                { 3, 0, 1 },
                                { 2, 9, 2 },
                                { 2, 3, 2 },
                                { 8, 3, 3 } };

        int avail[3] = { 3, 3, 2 };

        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
```

```
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {

                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break;
                    }
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);

    return (0);
}
```

Output :-



```
0
0
1
2
1
7
5
0
2
3
5
6
0
6
5
2
0
6
5
6

The Claim Vector is:    1       5       2       0
The Allocated Resource Table:
        0       0       1       2
        1       0       0       0
        1       3       5       4
        0       6       3       2
        0       0       1       4

The Maximum Claim Table:
        0       0       1       2
        1       7       5       0
        2       3       5       6
        0       6       5       2
        0       6       5       6

Allocated resources:    2       9       10      12
Available resources:    -1      -4      -8      -12
```