Student Name: Rutvik Surani

Student ID Email Address: rutvik.surani2012@gmail.com

GitHub Link: https://github.com/rutvik2012/Operating-System.git

Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i, x,limit, total = 0, z, counters = 0, tq=4;
    int wait_time=0, turnaround_time=0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;
        printf("\t\t\t\t\t\t**------Welcome to multilevel Queue Scheduling-------**\n");
        printf("\t\t\t\t\t\t 1. Shortest Time Remaining First(SRTF)\n");
        printf("\t\t\t\t\t\t 2. Round Robin(RR)\n");
        int in=0;
        printf("\t\t\t\t\t\t User enter your choice:\n");
        scanf("%d",&in);
        if(in==1)
        {
                int a[20],b[20],z[20],mini; //mini=smallest
                int wt_time[20],ta_time[20],complete_time[20];
                double avg=0,tt=0,end;
                int j,i,counter=0,t,n; //time = t

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

```
            for(i=0;i<n;i++)

            {

            printf("\nEnter arrival time of process %d : ",i+1);

            scanf("%d",&a[i]);

            printf("\nEnter burst time of process %d : ",i+1);

            scanf("%d",&b[i]);

            }

            for(i=0;i<n;i++)

            {

            z[i]=b[i];

            b[9]=9999;

    }

            for(t=0;counter!=n;t++)

            {

            mini=9;

            for(i=0;i<n;i++)

            {

            if(a[i]<=t && b[i]<b[mini] && b[i]>0 )

            mini=i;

            }

            b[mini]--;


            if(b[mini]==0)

            {

            counter++;

            end=t+1;

            complete_time[mini] = end;

            wt_time[mini] = end - a[mini] - z[mini];

            ta_time[mini] = end - a[mini];
```

```
                    }

                    }

                    printf("processes \t burst_time \t arrival_time \twaiting_time \tturnaround_time
\tcompletion_time");

                     for(i=0;i<n;i++)

                    {

                    printf("\n %d \t\t   %d \t\t %d\t\t\t%d  \t
\t\t%d\t\t\t%d",i+1,z[i],a[i],wt_time[i],ta_time[i],complete_time[i]);

                    avg = avg + wt_time[i];

                    tt = tt + ta_time[i];

                    }

                    printf("\n  \t\t\t\t\t%lf \t %lf",avg,tt);

                     printf("\n\nAverage waiting time = %lf\n",avg/n);

                    printf("Average Turnaround time = %lf",tt/n);

                    getch();

          }


          else if(in==2)

          {


          int i, x,limit, total = 0, z, counters = 0, tq=2;

    int wait_time=0, turnaround_time=0, arrival_time[10], burst_time[10], temp[10];

    float average_wait_time, average_turnaround_time;

    printf("\nEnter Total Number of Processes:\t");

    scanf("%d", &limit);

    x = limit;

    for(i = 0; i < limit; i++)

    {
        printf("\nEnter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");

        scanf("%d", &arrival_time[i]);

        printf("Burst Time:\t");

        scanf("%d", &burst_time[i]);

        temp[i] = burst_time[i];
    }

    //printf("nEnter Time Quantum:\t");  //for user to enter different time quantums
    //scanf("%d", &tq); //tq=Time qunataum
    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
    for(total = 0, i=0; x != 0;)
    {
        if(temp[i] <= tq && temp[i] > 0)
        {

            total = total + temp[i];
            temp[i] = 0;
            counters = 1;
            wait_time=total-temp[i];

        }
        else if(temp[i]>0)
        {
```

```
                    if(temp[i]){
                        total = total + tq;


                        temp[i] = temp[i] - tq;


        }
         }


         if(temp[i] == 0 && counters == 1)
         {
             x--;
             printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total-
arrival_time[i]-burst_time[i]);
             //wait_time = wait_time + total - arrival_time[i] - burst_time[i];
             turnaround_time = turnaround_time + total - arrival_time[i];
             counters = 0;
         }
         if(i == limit - 1)
         {
             i = 0;
         }
         else if(arrival_time[i + 1] <= total)
         {
             i++;
         }
         else
         {
             i = 0;
         }
```

```
    }

    average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
    printf("\n\nAverage Turnaround Time:\t%fn", average_turnaround_time);
    return 0;
    }
    else{
        if(in!=1||2)  // it will by default implement SRTF as it has the higher priority
        {
                int a[10],b[10],z[10];
                int wt_time[10],ta_time[10],complete_time[10]; //wt_time=waiting time  ta_time=turn
around time
                int i,j,mini,counter=0,t,n;
                double avg=0,tt=0,end;
                printf("\nEnter the number of Processes: ");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                printf("\nEnter arrival time of process %d : ",i+1);
                scanf("%d",&a[i]);
                printf("\nEnter burst time of process %d : ",i+1);
                scanf("%d",&b[i]);
                }
                for(i=0;i<n;i++)
                {
                z[i]=b[i];
                b[9]=9999;
```

```
                    }
                    for(t=0;counter!=n;t++)
                    {
                    mini=9;
                    for(i=0;i<n;i++){
                    if(a[i]<=t && b[i]<b[mini] && b[i]>0 )
                    mini=i;}b[mini]--;
                    if(b[mini]==0){
                    counter++;
                    end=t+1;
                    complete_time[mini] = end;
                    wt_time[mini] = end - a[mini] - z[mini];
                    ta_time[mini] = end - a[mini];}
                    }
                    printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");
                     for(i=0;i<n;i++)
                    {
                    printf("\n %d \t   %d \t %d\t\t%d
\t\t%d\t\t%d",i+1,z[i],a[i],wt_time[i],ta_time[i],complete_time[i]);
                    avg = avg + wt_time[i];
                    tt = tt + ta_time[i];
                    }
                    printf("\n  \t\t\t\t\t%lf \t %lf",avg,tt);
                     printf("\n\nAverage waiting time = %lf\n",avg/n);
                    printf("Average Turnaround time = %lf",tt/n);
                    getch();
                                        }
    }
```

1.Explain the problem in terms of operating system concept?

Multilevel Queue Scheduling divides the processes in groups based on the process type, CPU consumption, Memory size, input-output access, and many more. There will be ‘n’ no. of ques made for different groups of processes. Similarly, in the given problem there are two queues formed. 1st is Smallest Remaining Time First (SRTF) and the 2nd is Round Robin (RR). The SRTF scheduling queue has been given a higher priority as they run INTERACTIVE PROCESSES compared to RR scheduling queue which holds lower priority to run BATCH PROCESSES. In the RR scheduling queue the TIME QUANTUM is fixed as 2 units. The scheduler will take input from the users for 3 things:

- No. of processes
- Arrival time of each process
- Burst time of each process.
  (The scheduler also asks for the time quantum which will always be take as 2 units)

As for the output the scheduler needs to display the calculated AVERAGE TURNAROUND TIME and AVERAGE WAITING TIME, these both calculations will be done using the following formulas:

- Turnaround time=Burst time + Waiting time
- Turnaround time=Completion time – Arrival time
- Waiting time= Turnaround time – Burst time
- Completion time= Arrival time + Turnaround time

2.Calculate complexity of implemented algorithm.

```
#include<stdio.h>
#include<conio.h>
int main()
{
        printf("\t\t\t\t\t\t**------Welcome to multilevel Queue Scheduling-------**\n");
        printf("\t\t\t\t\t\t 1. Shortest Time Remaining First(SRTF)\n");
        printf("\t\t\t\t\t\t 2. Round Robin(RR)\n");
        int in=0;
        printf("\t\t\t\t\t\t User enter your choice:\n");
        scanf("%d",&in);
        if(in==1)
        {
                int a[20],b[20],z[20],mini; //mini=smallest
                int wt_time[20],ta_time[20],complete_time[20];
                double avg=0,tt=0,end;
                int j,i,counter=0,t,n; //time = t

                printf("\nEnter the number of Processes: ");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                printf("\nEnter arrival time of process %d : ",i+1);
                scanf("%d",&a[i]);                                        N
                printf("\nEnter burst time of process %d : ",i+1);
                scanf("%d",&b[i]);
                }

                for(i=0;i<n;i++)
                {z[i]=b[i];
```

```
                    b[9]=9999;}

                    for(t=0;counter!=n;t++)
                    {
                    mini=9;
                    for(i=0;i<n;i++)
                    {
                    if(a[i]<=t && b[i]<b[mini] && b[i]>0 )
                    mini=i;                                                        N
                    }
                    b[mini]--;                                                            N^2

                    if(b[mini]==0)
                    {
                    counter++;
                    end=t+1;
                    complete_time[mini] = end;
                    wt_time[mini] = end - a[mini] - z[mini];
                    ta_time[mini] = end - a[mini];

                    }
                    }

               printf("processes     \t
          burst_time \t arrival_time \twaiting_time \tturnaround_time \tcompletion_time");
                     for(i=0;i<n;i++)
                    {
                    printf("\n %d \t\t  %d \t\t %d\t\t\t%d  \t \t\t%d\t\t\t%d",i+1,z[i],a[i],wt_time[i],ta_time[i],complete_time[i]);
                    avg = avg + wt_time[i];                                                   N
                    tt = tt + ta_time[i];
                    }


                    printf("\n  \t\t\t\t%lf \t %lf",avg,tt);
                     printf("\n\nAverage waiting time = %lf\n",avg/n);
                    printf("Average Turnaround time = %lf",tt/n);
                    getch();
               }

               else if(in==2){

                int i, x,limit, total = 0, z, counters = 0, tq;
            int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
            float average_wait_time, average_turnaround_time;
            printf("\nEnter Total Number of Processes:\t");
            scanf("%d", &limit);
            x = limit;


            for(i = 0; i < limit; i++)
            {
                printf("nEnter Details of Process[%d]\n", i + 1);

                printf("Arrival Time:\t");
                                                                                              N
                scanf("%d", &arrival_time[i]);

                printf("Burst Time:\t");
```

```
        scanf("%d", &burst_time[i]);

        temp[i] = burst_time[i];
    }

    printf("nEnter Time Quantum:\t");
    scanf("%d", &tq); //tq=Time qunataum
    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i]   <=   tq   &&
temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counters = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - tq;
            total = total + tq;
        }
        if(temp[i] == 0 && counters == 1)
        {
            x--;
            printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total - arrival_time[i] -
burst_time[i]);
            wait_time = wait_time + total - arrival_time[i] - burst_time[i];
            turnaround_time = turnaround_time + total - arrival_time[i];
            counters = 0;
        }
        if(i == limit - 1)
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= total)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }


average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
    printf("\nAvg Turnaround Time:\t%fn", average_turnaround_time);
    return 0;
    }
    else{
            if(in!=1||2)
            {
                        int a[10],b[10],z[10];
                        int wt_time[10],ta_time[10],complete_time[10]; //wt_time=waiting time  ta_time=turn around time
```

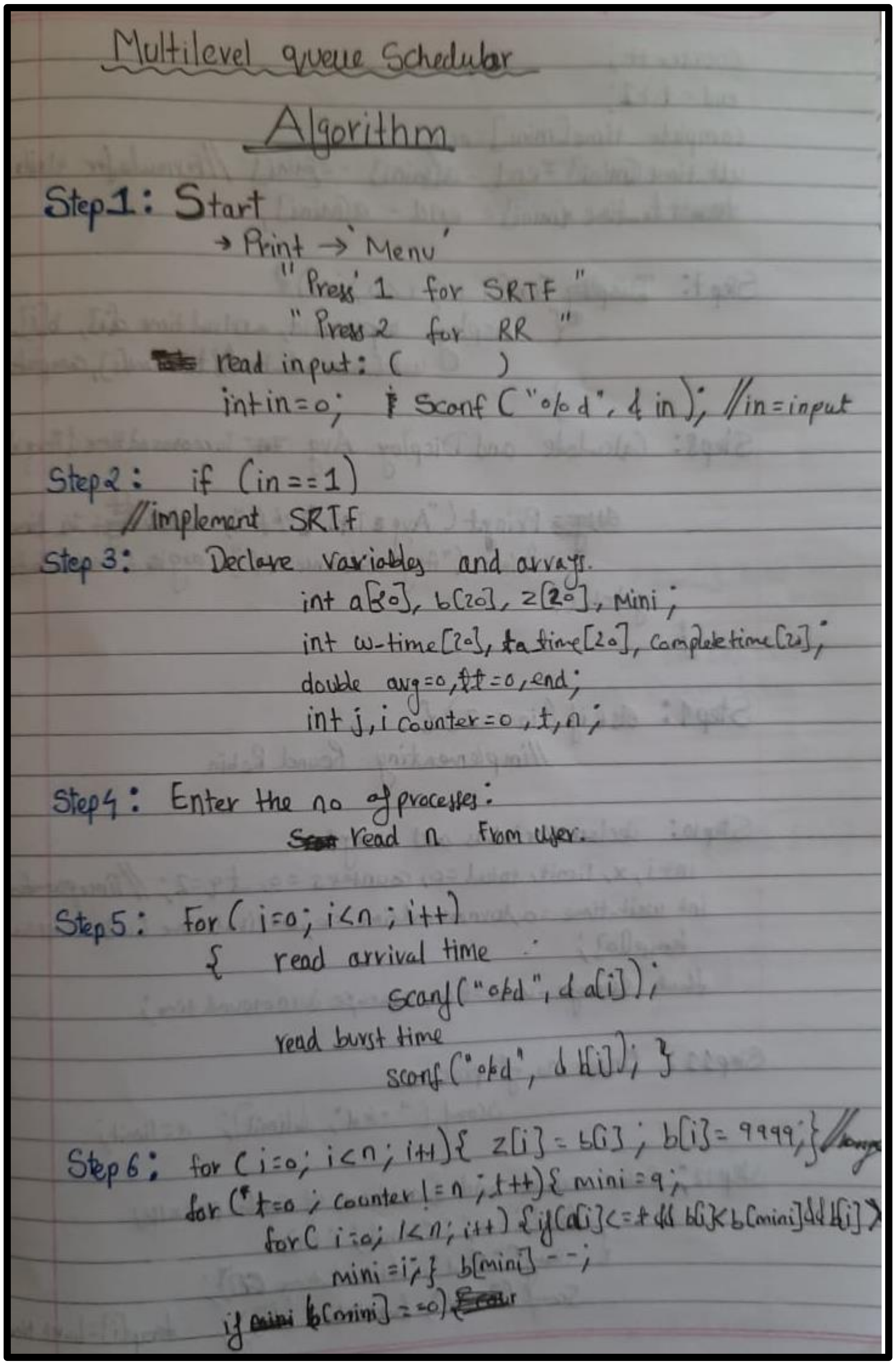
$\boxed{N}$

```
                    int i,j,mini,counter=0,t,n;
                    double avg=0,tt=0,end;
                    printf("\nEnter the number of Processes: ");
                    scanf("%d",&n);
                    for(i=0;i<n;i++)
                    {
          printf("\nEnter arrival time of process %d : ",i+1);
                    scanf("%d",&a[i]);                                              N
                    printf("\nEnter burst time of process %d : ",i+1);
                    scanf("%d",&b[i]);
                    }
                    for(i=0;i<n;i++)
                    {z[i]=b[i];
                    b[9]=9999;}

                    for(t=0;counter!=n;t++)
                    {
                    mini=9;
                    for(i=0;i<n;i++)
                    {                                                             N
                    if(a[i]<=t && b[i]<b[mini] && b[i]>0 )
                    mini=i;
                    }
                    b[mini]--;                                                          N^2

                    if(b[mini]==0)
                    {
                    counter++;
                    end=t+1;
                    complete_time[mini] = end;
                    wt_time[mini] = end - a[mini] - z[mini];
                    ta_time[mini] = end - a[mini];

                    }
                    }

          printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");
                     for(i=0;i<n;i++)
                    {
                    printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",i+1,z[i],a[i],wt_time[i],ta_time[i],complete_time[i]);
                    avg = avg + wt_time[i];
                    tt = tt + ta_time[i];                                               N
                    }

                    printf("\n
\t\t\t\t%lf \t %lf",avg,tt);
                     printf("\n\nAverage waiting time = %lf\n",avg/n);
                    printf("Average Turnaround time = %lf",tt/n);
                    getch();

                      }
   }
   }
```

Overall complexity of implemented algorithm.

- Overall Time complexity: O(n^2)
- Overall Space Complexity: O(n)

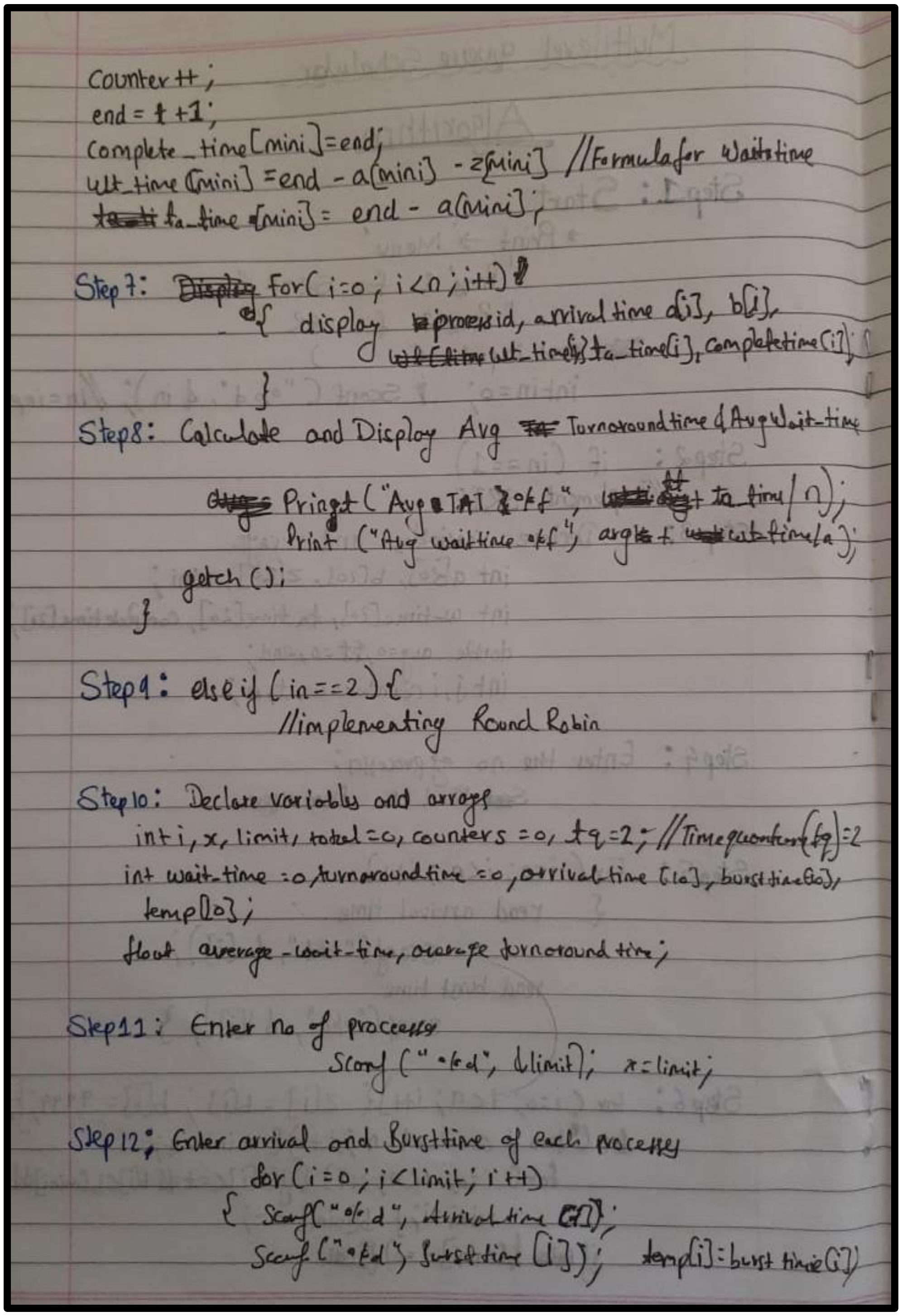
3: Write the algorithm for proposed solution of the assigned problem.



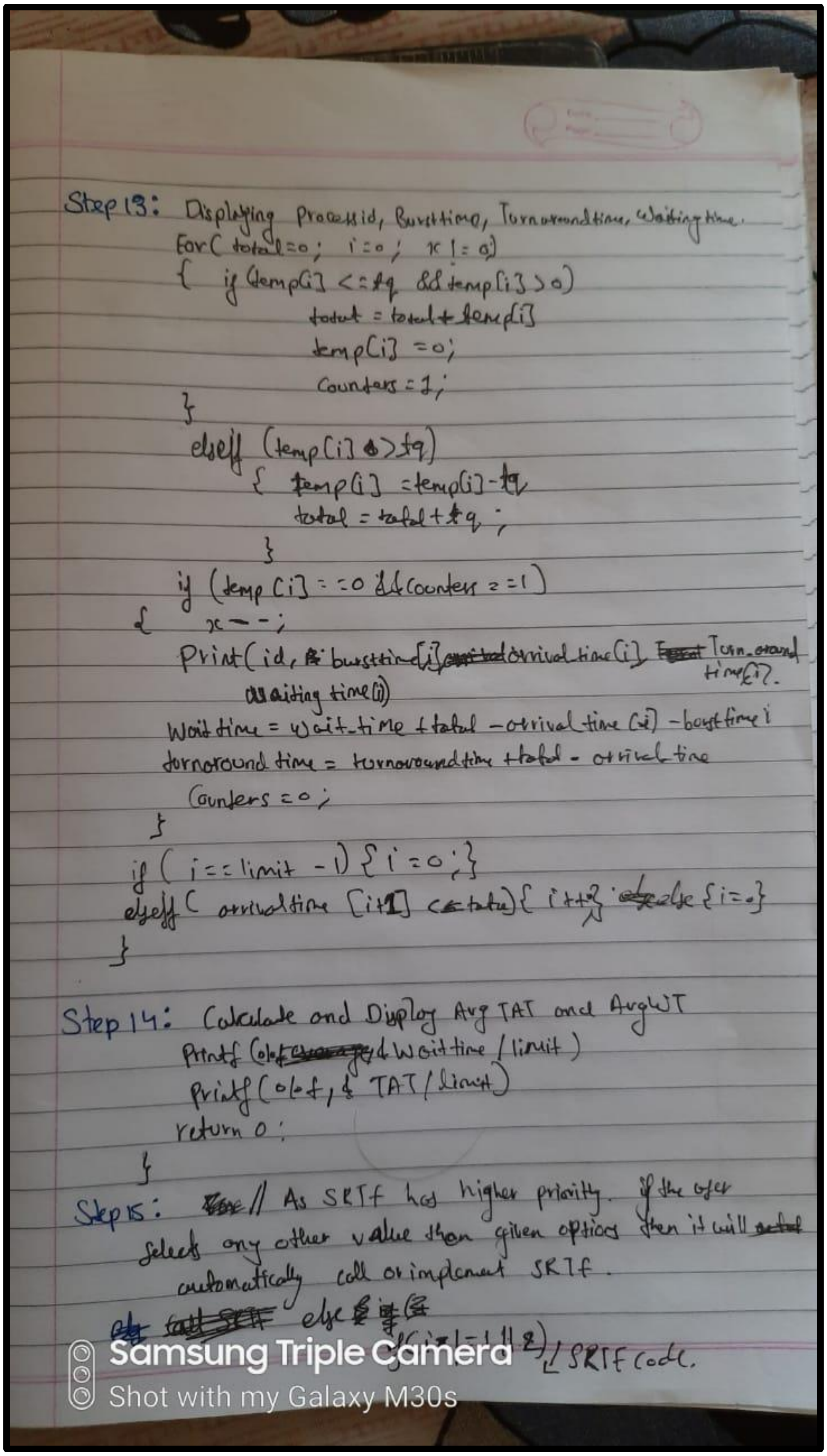
Multilevel queue Schedular

Algorithm

```
Step1: Start
              → Print → `Menu'
                  " Press' 1  for SRTF "
                  " Press 2  for  RR  "
        ~~[illegible]~~ read input: (        )
              int in=0;   ~~[illegible]~~ Sconf ("%d", & in); //in=input

Step2:  if (in==1)
     //implement SRTF
Step 3:     Declare variables and arrays.
                  int a[20], b[20], z[20], Mini;
                  int w-time[20], ~~ta~~ time[20], completetime[20];
                  double avg=0, ~~tt~~=0, end;
                  int j, i counter=0 , t, n;

Step4: Enter the no of processes:
                  ~~Scan~~ read  n   From user.

Step5:  For ( i=0; i<n ; i++)
           {   read arrival time
                        scanf("%d", & a[i]);
           read burst time
                        scanf("%d", & b[i]); }

Step 6;  for ( i=0; i<n; i++){ z[i] = b[i] ; b[i]= 9999;} //[illegible]
         for ( t=0 ; counter != n ; t++){ mini = 9;
              for( i=0; i<n; i++) { if(a[i]<=t && b[i]<b[mini]&& b[i]>
                     mini =i; } b[mini] --;
           if ~~mini~~ b[mini] ==0) { ~~[illegible]~~
```



```
Counter ++;
end = t +1;
Complete_time[mini]=end;
wt_time [mini] =end - a[mini] - z[mini] //Formula for Waitstime
~~ta~~ ta_time [mini] = end - a[mini];

Step 7: ~~Display~~ For( i=0 ; i<n ; i++)
        { display ~~p~~ processid, arrival time a[i], b[i],
                ~~wt[time~~ wt_time[i] ta_time[i], completetime[i]);
        }
Step8: Calculate and Display Avg ~~TA~~ Turnaroundtime & Avg Wait-time

        ~~[illegible]~~ Pringt ("Avg TAT %f", ~~[illegible]~~ tt ta_time/n);
              Print ("Avg waittime %f", arg ~~t~~ t ~~[illegible]~~ wt-time/n);
    getch ();
}

Step 9: else if (in==2){
            //implementing Round Robin

Step 10: Declare variables and arrays
    int i, x, limit, total=0, counters =0, tq=2; //Timequantum(tq)=2
    int wait-time =0, turnaroundtime =0, arrival-time [10], bursttime[10],
     temp[10];
    float average -wait-time, average turnaround time;

Step 11: Enter no of processes
                        Scanf ("%d", &limit); x=limit;

Step 12: Enter arrival and Bursttime of each processes
                for (i=0 ; i<limit; i++)
              { Scanf("%d", Arrival time [i]);
                Scanf ("%d", bursttime [i]); temp[i]=burst time[i]
```



Step 13: Displaying Processid, Bursttime, Turnaroundtime, Waiting time.

```
For( total=0; i=0; x != 0)
{  if (temp[i] <= tq && temp[i] > 0)
          total = total + temp[i]
          temp[i] = 0;
          Counters = 1;
}
 elseif (temp[i] > tq)
     { temp[i] = temp[i] - tq
       total = total + tq ;
     }
 if (temp [i] = = 0 && Counters = = 1)
{   x - - ;
 Print( id, bursttime[i], arrival time (i), Turn-around time[i].
      waiting time(i))
 Wait time = wait_time + total - arrival time (i) - burst time i
 turnaround time = turnaround time + total - arrival time
   Counters = 0 ;
}
if ( i == limit - 1) { i = 0 ; }
elseif ( arrivaltime [i+1] <= total) { i++ } else { i = 0 }
}
```

Step 14: Calculate and Display Avg TAT and Avg WT

```
Printf (%f, Waittime / limit )
Printf (%f, TAT / limit)
return 0 ;
}
```

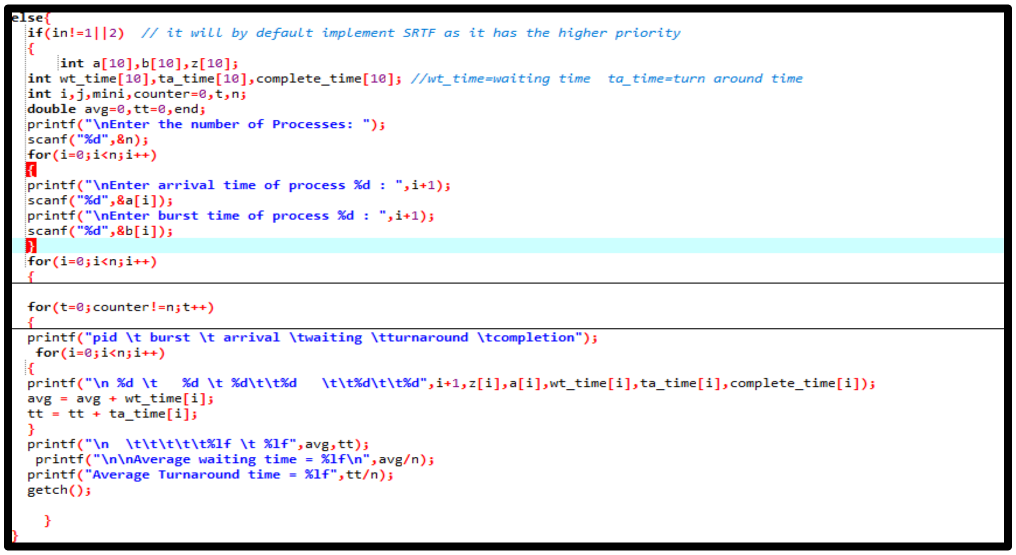
Step 15: // As SRTF has higher priority. if the user select any other value than given options then it will automatically call or implement SRTF.

~~call SRTF~~ else if(i != 1 || 2) // SRTF code.

Samsung Triple Camera
Shot with my Galaxy M30s

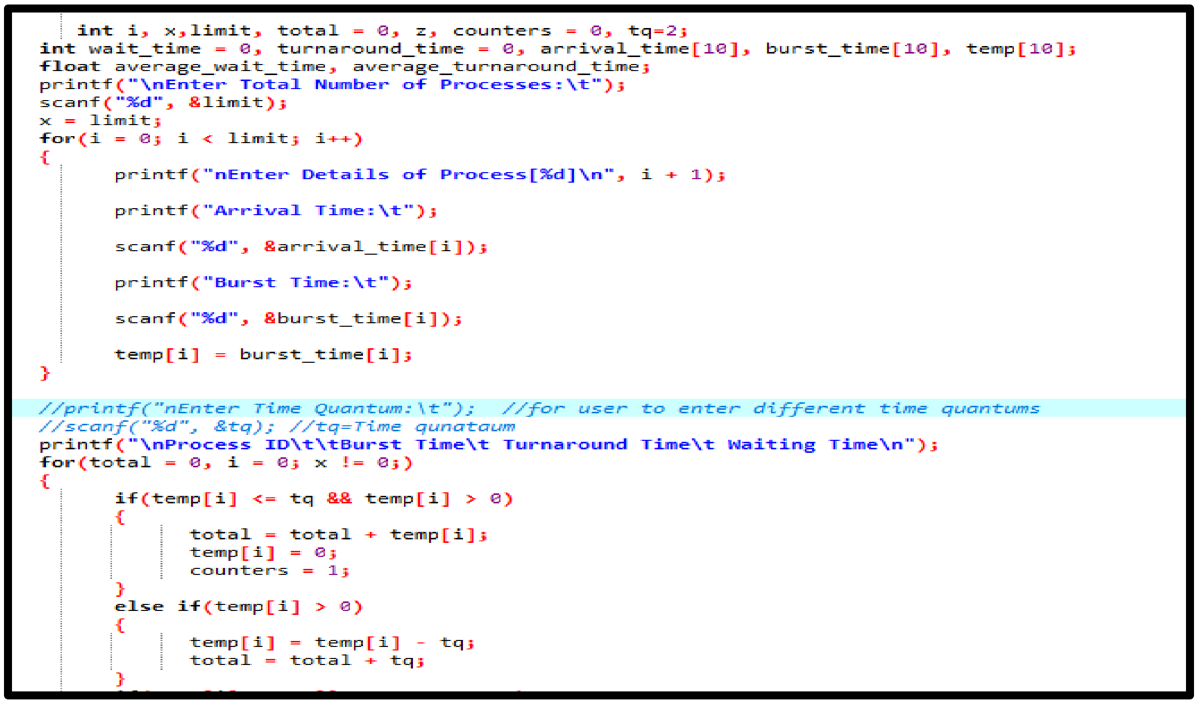
4.Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.
Code snippet:

- In the designed module of multilevel queue, the first queue of pre-emptive shortest remaining processing time first has the higher priority than the second queue which is Round robin. The priority is giving to the queue based on how the scheduling algorithm handles the incoming process and looks at various factor of waiting time, turnaround time and many more. Similarly, in our scheduler SRTF is one of the most efficient scheduling algorithm, it has minimum turnaround time, waiting time, and maximum CPU utilization, throughput.



```
else{
 if(in!=1||2)  // it will by default implement SRTF as it has the higher priority
 {
     int a[10],b[10],z[10];
 int wt_time[10],ta_time[10],complete_time[10]; //wt_time=waiting time  ta_time=turn around time
 int i,j,mini,counter=0,t,n;
 double avg=0,tt=0,end;
 printf("\nEnter the number of Processes: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("\nEnter arrival time of process %d : ",i+1);
 scanf("%d",&a[i]);
 printf("\nEnter burst time of process %d : ",i+1);
 scanf("%d",&b[i]);
 }
 for(i=0;i<n;i++)
 {

 for(t=0;counter!=n;t++)
 {
 printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");
  for(i=0;i<n;i++)
 {
 printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",i+1,z[i],a[i],wt_time[i],ta_time[i],complete_time[i]);
 avg = avg + wt_time[i];
 tt = tt + ta_time[i];
 }
 printf("\n  \t\t\t\t\t%lf \t %lf",avg,tt);
  printf("\n\nAverage waiting time = %lf\n",avg/n);
 printf("Average Turnaround time = %lf",tt/n);
 getch();

   }
}
```

- The time quantum for the Round Robin scheduling queue is fixed as 2 units. Therefore, all the processes coming in will be processed as per the default Quantum time which is of 2 units.



```
    int i, x,limit, total = 0, z, counters = 0, tq=2;
int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
float average_wait_time, average_turnaround_time;
printf("\nEnter Total Number of Processes:\t");
scanf("%d", &limit);
x = limit;
for(i = 0; i < limit; i++)
{
      printf("nEnter Details of Process[%d]\n", i + 1);

      printf("Arrival Time:\t");

      scanf("%d", &arrival_time[i]);

      printf("Burst Time:\t");

      scanf("%d", &burst_time[i]);

      temp[i] = burst_time[i];
}

//printf("nEnter Time Quantum:\t");   //for user to enter different time quantums
//scanf("%d", &tq); //tq=Time qunataum
printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
for(total = 0, i = 0; x != 0;)
{
      if(temp[i] <= tq && temp[i] > 0)
      {
            total = total + temp[i];
            temp[i] = 0;
            counters = 1;
      }
      else if(temp[i] > 0)
      {
            temp[i] = temp[i] - tq;
            total = total + tq;
      }
```

5.If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.   Description:

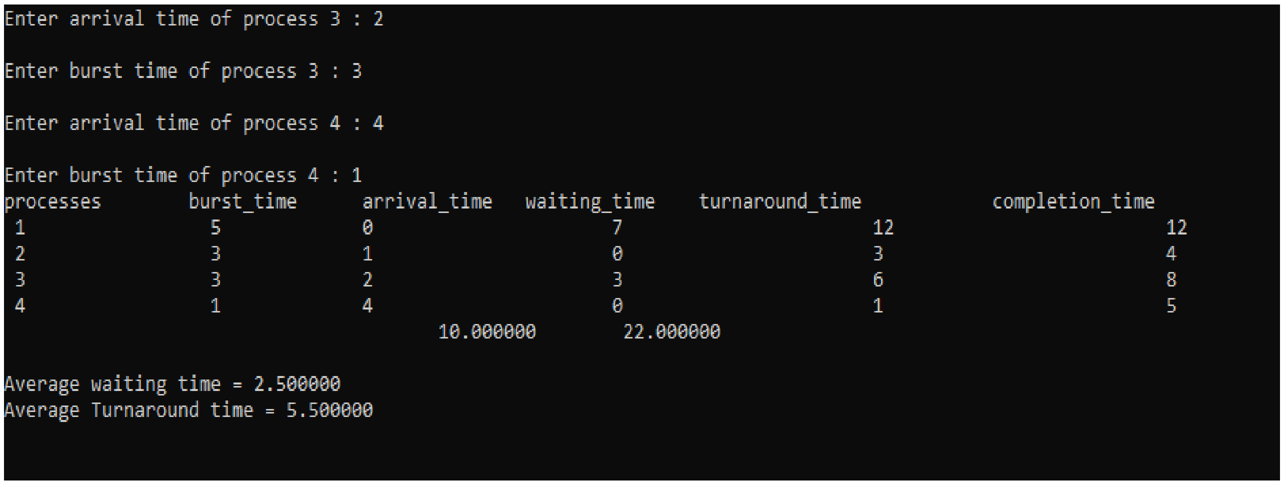
Before designing a multilevel scheduler, I have implemented SRTF scheduling algorithm and Round Robin scheduling algorithm differently, just to get the understanding of how it works And for the final multilevel schedular I have inculcate both the queues in it.

6. Explain the boundary conditions of the implemented code. Description:

- Boundary condition for Arrival time: As per users choice, (preferably below 10)
  The arrival time for all process is total based on the users choice, preferably below 10 as per the array capacity. With the value entered by the user all the other factors like turn around time and waiting time will be calculated.
- Boundary condition for Burst time: greater than 0
  The burst time for all process will be greater than 0. As a single process will always take at least 1 unit of time to process. 0 is not applicable.
- Time quantum value for Round Robin is fixed as 2 Units.
- All the values are considered in milliseconds.

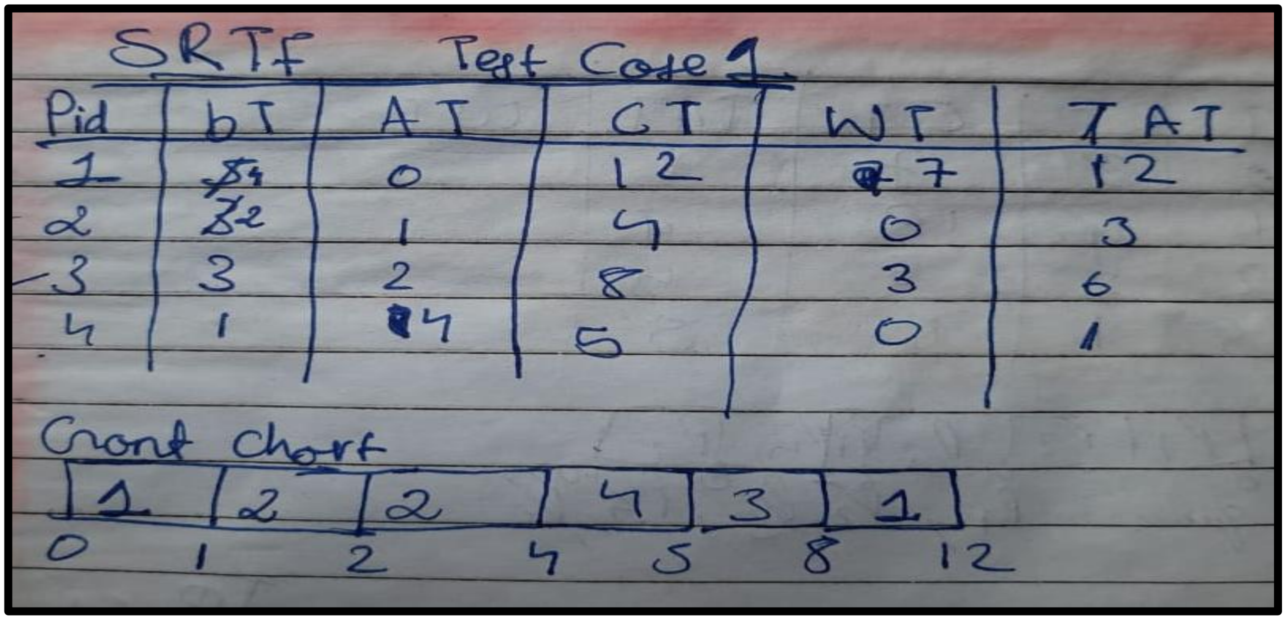
7.Explain all the test cases applied on the solution of assigned problem. Description:

Test case1: In this the designed multilevel scheduler is working on the SRTF queue. It is taking in value of no of process, arrival time of each processes, and burst time of each processes. With the help of above details, it calculates Waiting time, Turnaround time, and also the completion time of all processes.



```
Enter arrival time of process 3 : 2

Enter burst time of process 3 : 3

Enter arrival time of process 4 : 4

Enter burst time of process 4 : 1
processes        burst_time      arrival_time   waiting_time    turnaround_time           completion_time
 1                5              0                  7                      12                         12
 2                3              1                  0                      3                          4
 3                3              2                  3                      6                          8
 4                1              4                  0                      1                          5
                                       10.000000        22.000000

Average waiting time = 2.500000
Average Turnaround time = 5.500000
```

To verify if the code is giving out the correct output. I have manually calculated all the values and have followed the same procedure to get the desired outcome.



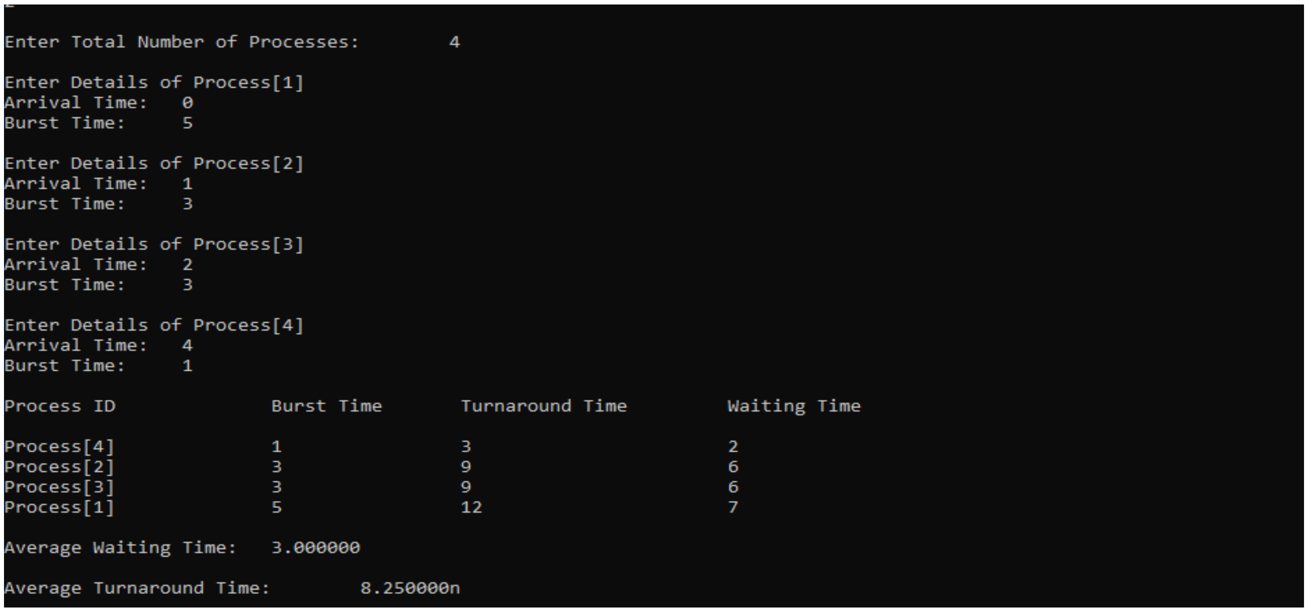
SRTf Test Case 1

Pid	bT	AT	CT	WT	TAT
1	~~5~~4	0	12	7	12
2	~~3~~2	1	4	0	3
3	3	2	8	3	6
4	1	4	5	0	1

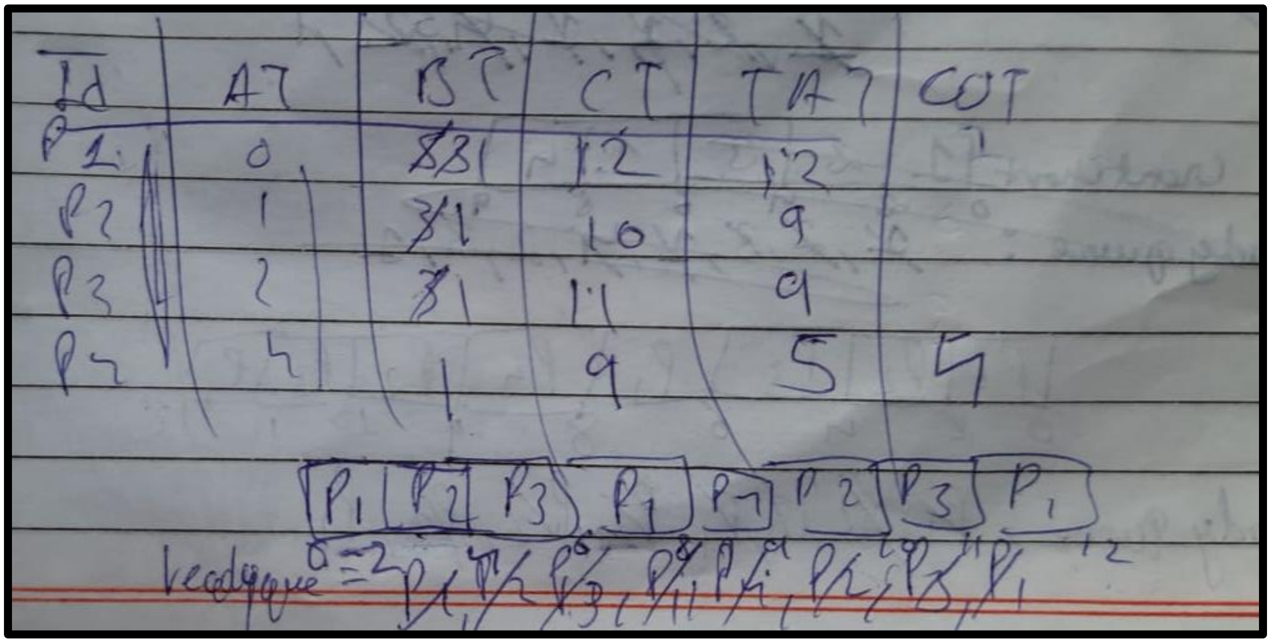
Gant Chart

	1	2	2	4	3	1
0	1	2	4	5	8	12

Test case2: The second testcase is based on the lower priority queue which is Round robin. In this the values of no of processes, arrival time, and burst time is given by the user. The scheduler has fixed time quantum of 2 units. with the given inputs the scheduler calculates the turnaround time and waiting time, as well as the average values of both. In order to verify the answers I have implemented the same problem manually.



```
Enter Total Number of Processes:        4

Enter Details of Process[1]
Arrival Time:   0
Burst Time:     5

Enter Details of Process[2]
Arrival Time:   1
Burst Time:     3

Enter Details of Process[3]
Arrival Time:   2
Burst Time:     3

Enter Details of Process[4]
Arrival Time:   4
Burst Time:     1

Process ID              Burst Time       Turnaround Time         Waiting Time

Process[4]              1                3                       2
Process[2]              3                9                       6
Process[3]              3                9                       6
Process[1]              5                12                      7

Average Waiting Time:   3.000000

Average Turnaround Time:        8.250000n
```



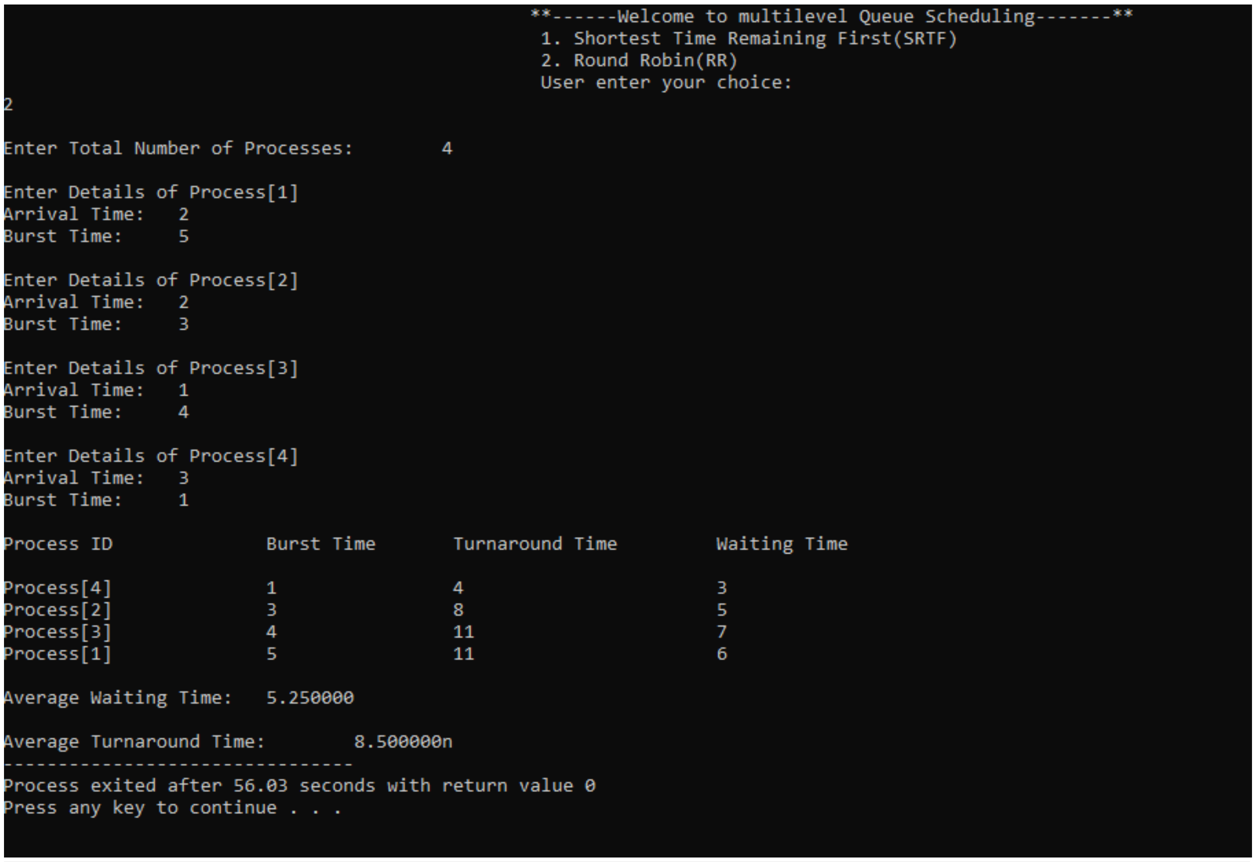
Id	AT	BT	CT	TAT	WT
P1	0	~~5~~ ~~3~~ 1	12	12	7
P2	1	~~3~~ 1	10	9	
P3	2	~~3~~ 1	11	9	
P4	4	1	9	5	4

P1	P2	P3	P1	P4	P2	P3	P1

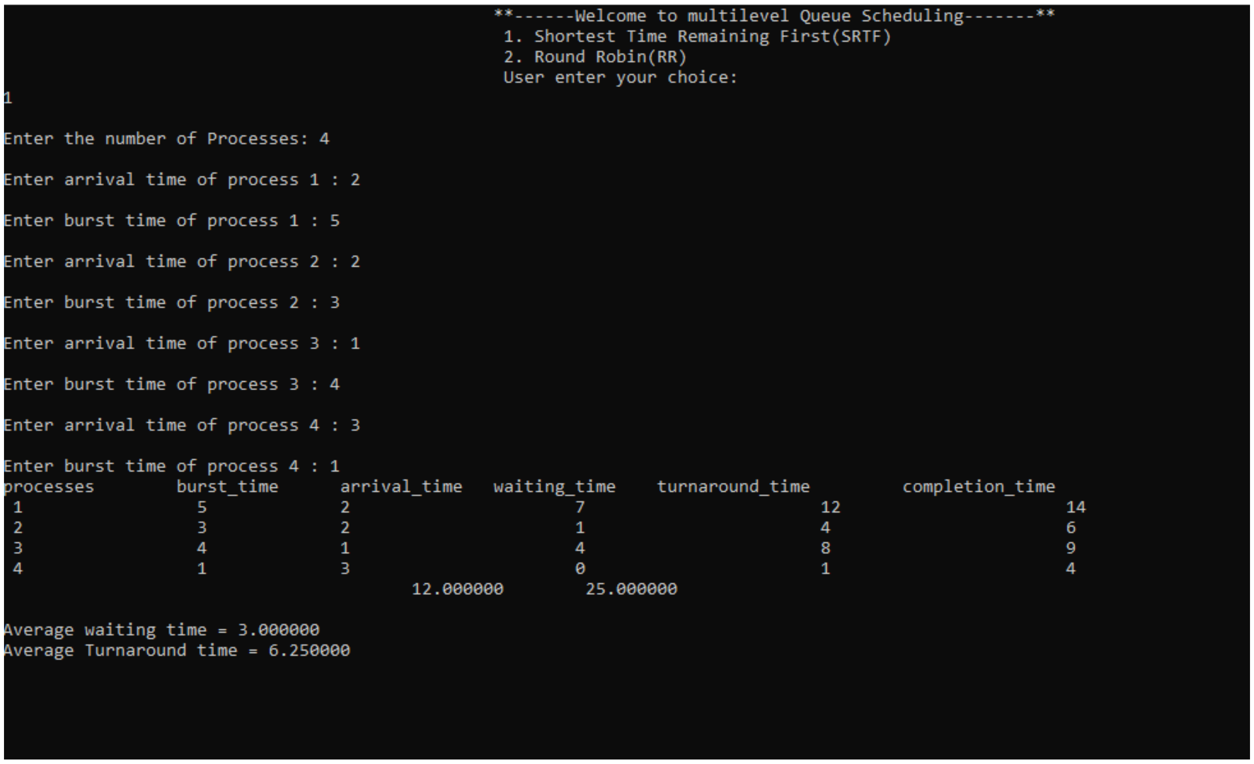
0 2 4 6 8 9 10 11 12

readyqueue = ~~P1~~, ~~P2~~, ~~P3~~, ~~P1~~, ~~P4~~, ~~P2~~, ~~P3~~, ~~P1~~

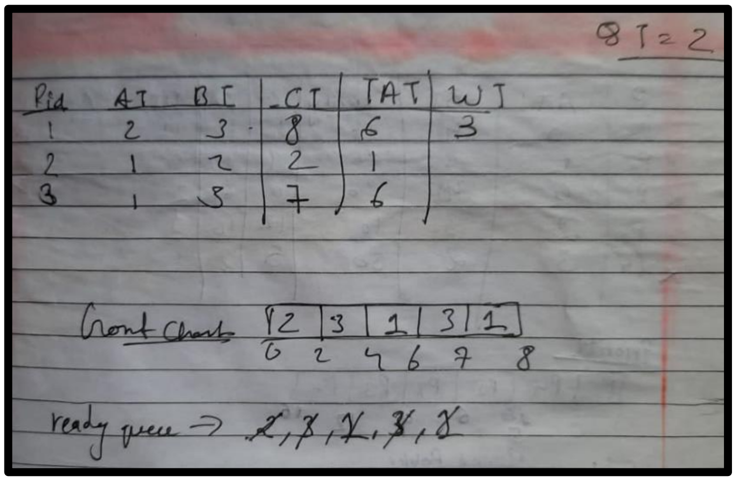
Other test cases with its manual implementation:



```
                                                **------Welcome to multilevel Queue Scheduling-------**
                                                 1. Shortest Time Remaining First(SRTF)
                                                 2. Round Robin(RR)
                                                 User enter your choice:
2

Enter Total Number of Processes:        4

Enter Details of Process[1]
Arrival Time:   2
Burst Time:     5

Enter Details of Process[2]
Arrival Time:   2
Burst Time:     3

Enter Details of Process[3]
Arrival Time:   1
Burst Time:     4

Enter Details of Process[4]
Arrival Time:   3
Burst Time:     1

Process ID              Burst Time       Turnaround Time         Waiting Time

Process[4]              1                4                       3
Process[2]              3                8                       5
Process[3]              4                11                      7
Process[1]              5                11                      6

Average Waiting Time:   5.250000

Average Turnaround Time:        8.500000n
--------------------------------
Process exited after 56.03 seconds with return value 0
Press any key to continue . . .
```



```
                                                **------Welcome to multilevel Queue Scheduling-------**
                                                 1. Shortest Time Remaining First(SRTF)
                                                 2. Round Robin(RR)
                                                 User enter your choice:
1

Enter the number of Processes: 4

Enter arrival time of process 1 : 2

Enter burst time of process 1 : 5

Enter arrival time of process 2 : 2

Enter burst time of process 2 : 3

Enter arrival time of process 3 : 1

Enter burst time of process 3 : 4

Enter arrival time of process 4 : 3

Enter burst time of process 4 : 1
processes        burst_time      arrival_time   waiting_time    turnaround_time        completion_time
 1                5              2                      7                       12                      14
 2                3              2                      1                       4                       6
 3                4              1                      4                       8                       9
 4                1              3                      0                       1                       4
                                        12.000000        25.000000

Average waiting time = 3.000000
Average Turnaround time = 6.250000
```



QT = 2

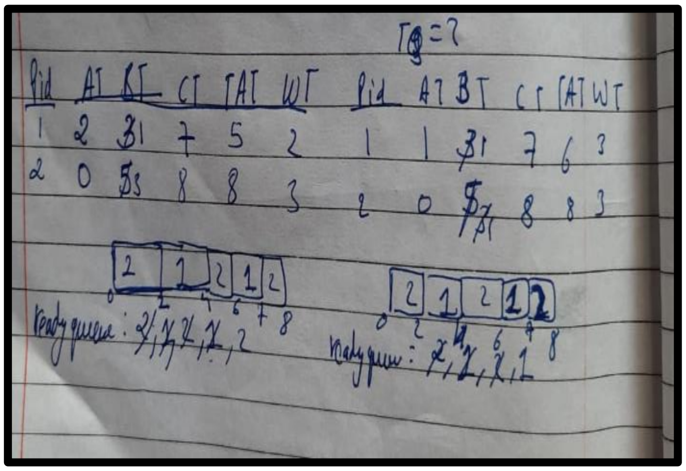
Pid	AT	BT	CT	TAT	WT
1	2	3	8	6	3
2	1	2	2	1	
3	1	3	7	6	

Gant Chart

2	3	1	3	1

0 2 4 6 7 8

ready queue → ~~2~~, ~~3~~, ~~1~~, ~~3~~, ~~1~~



TQ = 2

Pid	AT	BT	CT	TAT	WT
1	2	~~3~~ 1	7	5	2
2	0	~~5~~ 3	8	8	3

Pid	AT	BT	CT	TAT	WT
1	1	~~3~~ 1	7	6	3
2	0	~~5~~ ~~3~~ 1	8	8	3

2	1	2	1	2

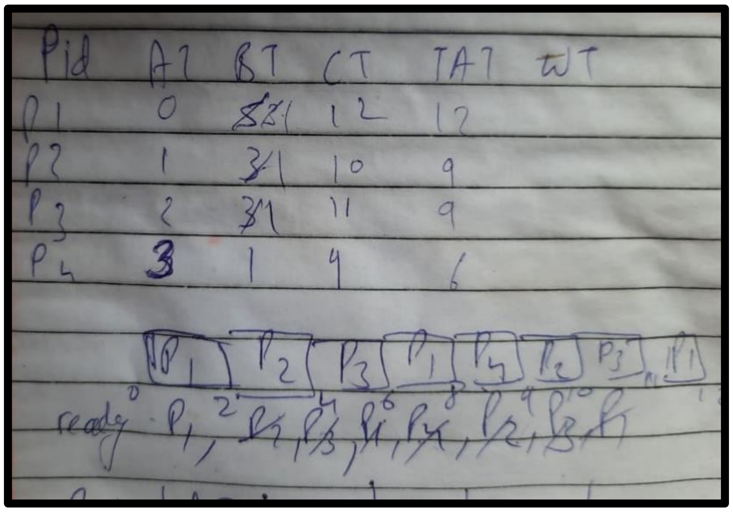
0 2 4 6 7 8

ready queue : ~~2~~, ~~1~~, ~~2~~, ~~1~~, 2

2	1	2	1	2

0 2 4 6 7 8

ready queue : ~~2~~, ~~1~~, ~~2~~, 1



Pid	AT	BT	CT	TAT	WT
P1	0	~~8~~ ~~3~~ 1	12	12	
P2	1	~~3~~ 1	10	9	
P3	2	~~3~~ 1	11	9	
P4	3	1	4	6	

P1	P2	P3	P1	P4	P2	P3	P1

0 2 4 6 8 9 10 11 12

ready : P1, ~~P2~~, ~~P3~~, ~~P1~~, ~~P4~~, ~~P2~~, ~~P3~~, P1

8.Have you made minimum 5 revisions of solution on GitHub? GitHub Link:

Yes,

https://github.com/rutvik2012/Operating-System.git.