LAB - 4

NAME: SOHIL AGARWAL
REG. NO.: 21BCE5985

1) Develop a C code for performing scheduling algorithm that schedules the processes in the order in which they arrive directly. Test your algorithm for the below scenario.

Process	Burst Time	Arrival Time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Find out the completion time of the process, Average TAT and Average WT of the processes.

CODE:

```c
#include <stdio.h>

// Function to calculate average waiting time and average turnaround time
void calculateAvgTimes(int processes[], int n, int burst_time[], int arrival_time[]) {
    int waiting_time[n], turnaround_time[n], completion_time[n], total_wt = 0, total_tat =
0;

    // Calculate completion time for the first process
    completion_time[0] = burst_time[0] + arrival_time[0];

    // Calculate completion time and waiting time for each process
    for (int i = 1; i < n; i++) {
        completion_time[i] = completion_time[i - 1] + burst_time[i];

        // Calculate waiting time
        waiting_time[i] = completion_time[i] - arrival_time[i] - burst_time[i];

        // If waiting time is negative, make it zero
        if (waiting_time[i] < 0)
            waiting_time[i] = 0;
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = completion_time[i] - arrival_time[i];
```

```c
        // Calculate total waiting time and total turnaround time
        total_wt += waiting_time[i];
        total_tat += turnaround_time[i];
    }

    // Print process details
    printf("\nProcess\tBurst Time\tArrival Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i], arrival_time[i], completion_time[i], turnaround_time[i], waiting_time[i]);
    }

    // Print average waiting time and average turnaround time
    printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", (float)total_tat / n);
}

int main() {
    int n; // Number of processes

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

    int processes[n]; // Array to store process IDs
    int burst_time[n]; // Array to store burst times
    int arrival_time[n]; // Array to store arrival times

    // Input process details
    for (int i = 0; i < n; i++) {
        printf("\nEnter the details for Process %d:\n", i + 1);
        printf("Process ID: ");
        scanf("%d", &processes[i]);
        printf("Burst Time: ");
        scanf("%d", &burst_time[i]);
        printf("Arrival Time: ");
        scanf("%d", &arrival_time[i]);
    }

    // Sort the processes based on arrival time (using bubble sort)
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arrival_time[j] > arrival_time[j + 1]) {
                // Swap arrival time
                int temp = arrival_time[j];
                arrival_time[j] = arrival_time[j + 1];
                arrival_time[j + 1] = temp;

                // Swap burst time
                temp = burst_time[j];
                burst_time[j] = burst_time[j + 1];
                burst_time[j + 1] = temp;
```

```
                // Swap process ID
                temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    // Calculate and display average waiting time and average turnaround time
    calculateAvgTimes(processes, n, burst_time, arrival_time);

    return 0;
}
```

OUTPUT:

```
PS C:\Users\Sohil\Desktop\VS CODES\OS> cd "c:\Users\Sohil\Desktop\VS CODES\OS\" ;
FS }
Enter the number of processes: 5

Enter the details for Process 1:
Process ID: 1
Burst Time: 6
Arrival Time: 2
```

```
Enter the details for Process 2:
Process ID: 2
Burst Time: 2
Arrival Time: 5

Enter the details for Process 3:
Process ID: 3
Burst Time: 8
Arrival Time: 1

Enter the details for Process 4:
Process ID: 4
Burst Time: 3
Arrival Time: 0

Enter the details for Process 5:
Process ID: 5
Burst Time: 4
Arrival Time: 4

Process Burst Time      Arrival Time    Completion Time Turnaround Time Waiting Time
4       3               0               3               3               0
3       8               1               11              10              2
1       6               2               17              15              9
5       4               4               21              17              13
2       2               5               23              18              16

Average Waiting Time: 8.00
Average Turnaround Time: 12.60
```

2) Develop a C code for performing scheduling algorithm with priority assigned for the processes as below.

Process No.	Priority	Arrival Time (AT)	Burst Time (BT)
P1	2	0	2
P2	4	1	5
P3	6	2	1
P4	10	3	2
P5	8	4	3
P6	12	5	6

Find out the completion time of the process, Average TAT and Average WT of the processes.

CODE:

```
#include <stdio.h>

// Function to calculate average waiting time and average turnaround time
void calculateAvgTimes(int processes[], int n, int burst_time[], int arrival_time[], int
priority[]) {
    int waiting_time[n], turnaround_time[n], completion_time[n], total_wt = 0, total_tat =
0;

    // Calculate completion time for the first process
    completion_time[0] = burst_time[0] + arrival_time[0];

    // Calculate completion time and waiting time for each process
    for (int i = 1; i < n; i++) {
        int min_priority = priority[i];
        int min_priority_index = i;

        // Find the process with the minimum priority among the remaining processes
        for (int j = i + 1; j < n; j++) {
            if (priority[j] < min_priority) {
                min_priority = priority[j];
                min_priority_index = j;
            }
        }

        // Swap the process details (priority, burst time, arrival time) of the minimum
priority process with the current process
        int temp = priority[i];
        priority[i] = priority[min_priority_index];
        priority[min_priority_index] = temp;

        temp = burst_time[i];
        burst_time[i] = burst_time[min_priority_index];
        burst_time[min_priority_index] = temp;
```

```
        temp = arrival_time[i];
        arrival_time[i] = arrival_time[min_priority_index];
        arrival_time[min_priority_index] = temp;

        // Calculate completion time
        completion_time[i] = completion_time[i - 1] + burst_time[i];

        // Calculate waiting time
        waiting_time[i] = completion_time[i] - arrival_time[i] - burst_time[i];

        // If waiting time is negative, make it zero
        if (waiting_time[i] < 0)
            waiting_time[i] = 0;
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = completion_time[i] - arrival_time[i];

        // Calculate total waiting time and total turnaround time
        total_wt += waiting_time[i];
        total_tat += turnaround_time[i];
    }

    // Print process details
    printf("\nProcess\tPriority\tBurst Time\tArrival Time\tTurnaround Time\tWaiting
Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], priority[i],
burst_time[i], arrival_time[i], turnaround_time[i], waiting_time[i]);
    }

    // Print average waiting time and average turnaround time
    printf("\nAverage Waiting Time: %.2f", (float)total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", (float)total_tat / n);
}

int main() {
    int n; // Number of processes

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

    int processes[n]; // Array to store process IDs
    int burst_time[n]; // Array to store burst times
    int arrival_time[n]; // Array to store arrival times
    int priority[n]; // Array to store priorities

    // Input process details
    for (int i = 0; i < n; i++) {
        printf("\nEnter the details for Process %d:\n", i + 1);
        printf("Process ID: ");
```

```
        scanf("%d", &processes[i]);
        printf("Burst Time: ");
        scanf("%d", &burst_time[i]);
        printf("Arrival Time: ");
        scanf("%d", &arrival_time[i]);
        printf("Priority: ");
        scanf("%d", &priority[i]);
    }

    // Sort the processes based on priority (using bubble sort)
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (priority[j] > priority[j + 1]) {
                // Swap priority
                int temp = priority[j];
                priority[j] = priority[j + 1];
                priority[j + 1] = temp;

                // Swap burst time
                temp = burst_time[j];
                burst_time[j] = burst_time[j + 1];
                burst_time[j + 1] = temp;

                // Swap arrival time
                temp = arrival_time[j];
                arrival_time[j] = arrival_time[j + 1];
                arrival_time[j + 1] = temp;

                // Swap process ID
                temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    // Calculate and display average waiting time and average turnaround time
    calculateAvgTimes(processes, n, burst_time, arrival_time, priority);

    return 0;
}
```

OUTPUT:

```
Enter the number of processes: 6

Enter the details for Process 1:
Process ID: 1
Burst Time: 2
Arrival Time: 0
Priority: 2

Enter the details for Process 2:
Process ID: 2
Burst Time: 5
Arrival Time: 1
Priority: 4

Enter the details for Process 3:
Process ID: 3
Burst Time: 1
Arrival Time: 2
Priority: 6

Enter the details for Process 4:
Process ID: 4
Burst Time: 2
Arrival Time: 3
Priority: 10

Enter the details for Process 5:
Process ID: 5
Burst Time: 3
Arrival Time: 4
Priority: 8

Enter the details for Process 6:
Process ID: 6
Burst Time: 6
Arrival Time: 5
Priority: 12

Process Priority        Burst Time      Arrival Time    Turnaround Time Waiting Time
1       2               2               0               2               0
2       4               5               1               6               1
3       6               1               2               6               5
5       8               3               4               7               4
4       10              2               3               10              8
6       12              6               5               14              8

Average Waiting Time: 4.33
Average Turnaround Time: 7.50
```

3) In air traffic control systems, there are multiple aircraft that need to be tracked and monitored in real-time. The aircraft with the shortest remaining task/fly time can be given priority to ensure the safety of the aircraft. If the aircrafts have arrived at a Chennai international airport in the following order with their remaining fly time as follows:

Aircraft	Arrival Time	Burst Time
A1	0	7
A2	1	5
A3	2	3
A4	3	1
A5	4	2
A6	5	1

Develop a C code to determine the average waiting time and average turnaround time using the appropriate algorithm for the above scenario.

CODE:

```c
#include<stdio.h>
int main()
{
    int at[10],bt[10],pr[10];
    int n,i,j,temp,time=0,count,over=0,sum_wait=0,sum_turnaround=0,start;
    float avgwait,avgturn;
    printf("Enter the number of processes\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
                printf("Enter the arrival time and execution time for process %d\n",i+1);
                scanf("%d%d",&at[i],&bt[i]);
                pr[i]=i+1;
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(at[i]>at[j])
            {
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
                temp=bt[i];
                bt[i]=bt[j];
                bt[j]=temp;
                temp=pr[i];
                pr[i]=pr[j];
                pr[j]=temp;
```

```
            }

        }
    }
    printf("\n\nProcess\t|Arrival time\t|Execution time\t|Start time\t|End time\t|waiting
time\t|Turnaround time\n\n");
    while(over<n)
    {
        count=0;
        for(i=over;i<n;i++)
        {
                    if(at[i]<=time)
                    count++;
                    else
                    break;
        }
        if(count>1)
        {
            for(i=over;i<over+count-1;i++)
            {
                for(j=i+1;j<over+count;j++)
                {
                    if(bt[i]>bt[j])
                    {
                        temp=at[i];
                        at[i]=at[j];
                        at[j]=temp;
                        temp=bt[i];
                        bt[i]=bt[j];
                        bt[j]=temp;
                        temp=pr[i];
                        pr[i]=pr[j];
                        pr[j]=temp;
                    }
                }
            }
        }
        start=time;
        time+=bt[over];
        printf("p[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\n",pr[over],
        at[over],bt[over],start,time,time-at[over]-bt[over],time-at[over]);
        sum_wait+=time-at[over]-bt[over];
        sum_turnaround+=time-at[over];
        over++;
    }
    avgwait=(float)sum_wait/(float)n;
    avgturn=(float)sum_turnaround/(float)n;
    printf("Average waiting time is %f\n",avgwait);
    printf("Average turnaround time is %f\n",avgturn);
    return 0;
}
```

OUTPUT:

```
PS C:\Users\Sohil\Desktop\VS CODES\OS> cd "c:\Users\Sohil\Desktop\VS CODES\OS\" ; if ($?) { gcc Shortest_time.c -o Shortest_time
} ; if ($?) { .\Shortest_time }
Enter the number of processes
6
Enter the arrival time and execution time for process 1
0 7
Enter the arrival time and execution time for process 2
1 5
Enter the arrival time and execution time for process 3
2 3
Enter the arrival time and execution time for process 4
3 1
Enter the arrival time and execution time for process 5
4 2
Enter the arrival time and execution time for process 6
5 1


Process |Arrival time   |Execution time |Start time     |End time       |waiting time   |Turnaround time

p[1]    |       0       |       7       |       0       |       7       |       0       |       7
p[4]    |       3       |       1       |       7       |       8       |       4       |       5
p[6]    |       5       |       1       |       8       |       9       |       3       |       4
p[5]    |       4       |       2       |       9       |       11      |       5       |       7
p[3]    |       2       |       3       |       11      |       14      |       9       |       12
p[2]    |       1       |       5       |       14      |       19      |       13      |       18
Average waiting time is 5.666667
Average turnaround time is 8.833333
```

4) Develop a C code for performing round robin scheduling algorithm with quantum time has 2ms for the below set of processes.

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Find out the completion time of the process, Average TAT and Average WT of the processes.

CODE:

```c
#include<stdio.h>
#include<limits.h>
#include<stdbool.h>

struct P{
int AT,BT,ST[20],WT,FT,TAT,pos;
};

int quant;
int main(){
int n,i,j;
// Taking Input
printf("Enter the no. of processes :");
scanf("%d",&n);
struct P p[n];

printf("Enter the quantum  \n");
scanf("%d",&quant);
```

```
printf("Enter the process numbers \n");
for(i=0;i<n;i++)
scanf("%d",&(p[i].pos));

printf("Enter the Arrival time of processes \n");
for(i=0;i<n;i++)
scanf("%d",&(p[i].AT));

printf("Enter the Burst time of processes \n");
for(i=0;i<n;i++)
scanf("%d",&(p[i].BT));



// Declaring variables
int c=n,s[n][20];
float time=0,mini=INT_MAX,b[n],a[n];

// Initializing burst and arrival time arrays
int index=-1;
for(i=0;i<n;i++){
        b[i]=p[i].BT;
        a[i]=p[i].AT;
        for(j=0;j<20;j++){
        s[i][j]=-1;
        }
}

int tot_wt,tot_tat;
tot_wt=0;
tot_tat=0;
bool flag=false;

while(c!=0){

mini=INT_MAX;
flag=false;

for(i=0;i<n;i++){
        float p=time+0.1;
        if(a[i]<=p && mini>a[i] && b[i]>0){
        index=i;
        mini=a[i];
        flag=true;

        }
}
// if at =1 then loop gets out  hence set flag to false
if(!flag){
        time++;
        continue;
}

//calculating start time
```

```
j=0;

while(s[index][j]!=-1){
j++;
}

if(s[index][j]==-1){
s[index][j]=time;
p[index].ST[j]=time;
}

if(b[index]<=quant){
time+=b[index];
b[index]=0;
}
else{
time+=quant;
b[index]-=quant;
}

if(b[index]>0){
a[index]=time+0.1;
}

// calculating arrival,burst,final times
if(b[index]==0){
c--;
p[index].FT=time;
p[index].WT=p[index].FT-p[index].AT-p[index].BT;
tot_wt+=p[index].WT;
p[index].TAT=p[index].BT+p[index].WT;
tot_tat+=p[index].TAT;
}
} // end of while loop

// Printing output
printf("Process number ");
printf("Arrival time ");
printf("Burst time ");
printf("\tStart time");
j=0;
while(j!=10){
j+=1;
printf(" ");
}
printf("\t\tFinal time");
printf("\tWait Time ");
printf("\tTurnAround Time \n");


for(i=0;i<n;i++){
printf("%d \t\t",p[i].pos);
printf("%d \t\t",p[i].AT);
printf("%d \t",p[i].BT);
```

```
j=0;
int v=0;
while(s[i][j]!=-1){
printf("%d ",p[i].ST[j]);
j++;
v+=3;
}
while(v!=40){
printf(" ");
v+=1;
}
printf("%d \t\t",p[i].FT);
printf("%d \t\t",p[i].WT);
printf("%d \n",p[i].TAT);

}

//Calculating average wait time and turnaround time
double avg_wt,avg_tat;
avg_wt=tot_wt/(float)n;
avg_tat=tot_tat/(float)n;

//Printing average wait time and turnaround time
printf("The average wait time is : %lf\n",avg_wt);
printf("The average TurnAround time is : %lf\n",avg_tat);

return 0;
}
```

OUTPUT:

```
Enter the no. of processes :5
Enter the quantum
2
Enter the process numbers
1
2
3
4
5
Enter the Arrival time of processes
0
1
2
3
4
Enter the Burst time of processes
5
3
1
2
3
Process number Arrival time Burst time  Start time                    Final time      Wait Time       TurnAround Time
1              0            5          0 5 12                        13              8               13
2              1            3          2 11                           12              8               11
3              2            1          4                              5               2               3
4              3            2          7                              9               4               6
5              4            3          9 13                           14              7               10
The average wait time is : 5.800000
The average TurnAround time is : 8.600000
```