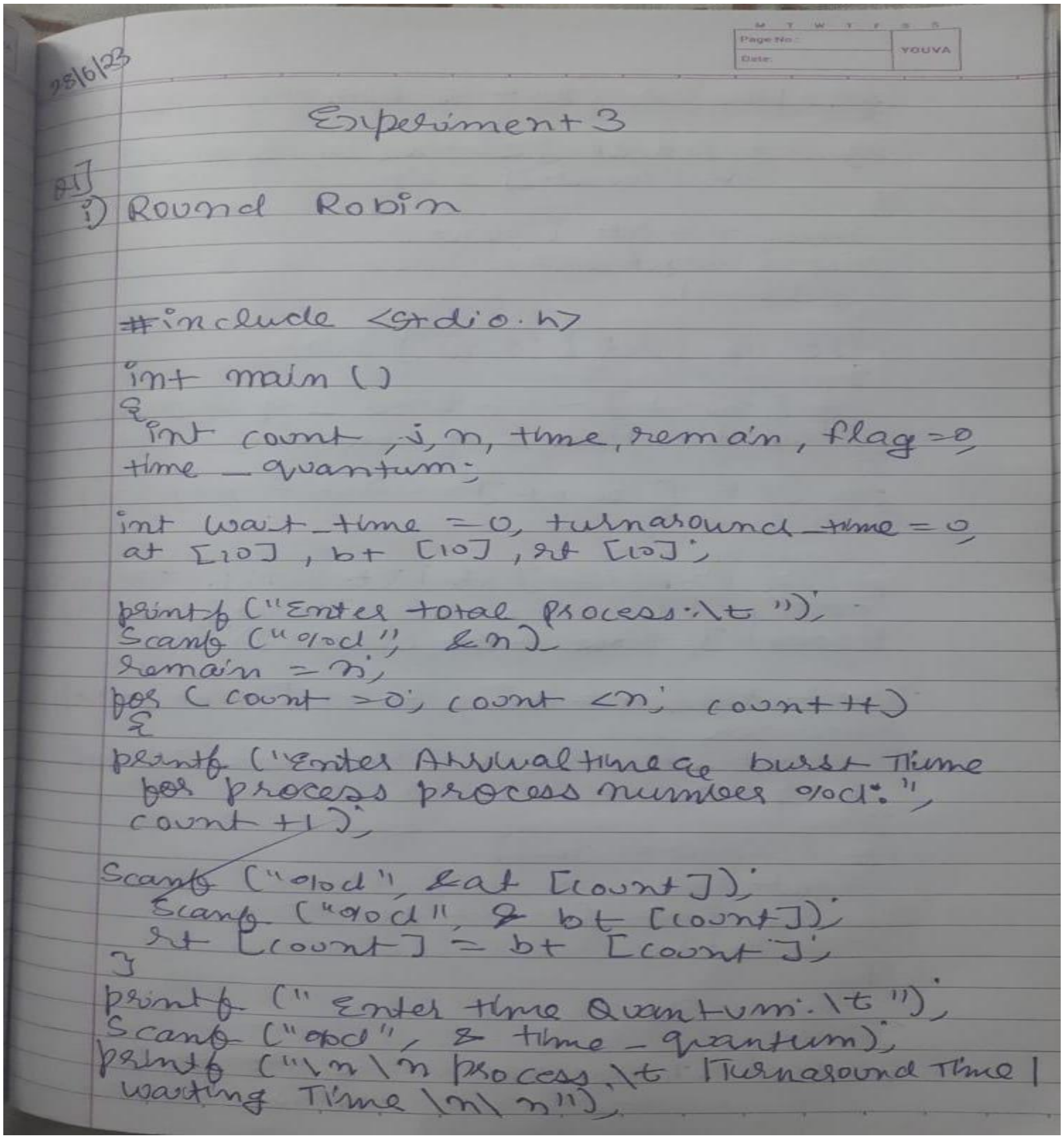
Q. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

Priority (pre-emptive or Non-pre-emptive) Round Robin (Experiment with different quantum sizes for RR algorithm)

OBSERVATION:

ROUND ROBIN:



28/6/23

Experiment 3

Q1]

i) Round Robin

```
#include <stdio.h>
int main ()
{
 int count , i, n, time, remain , flag = 0,
time _ quantum;

int wait_time = 0, turnaround_time = 0,
 at [10] , bt [10] , rt [10];

printf ("Enter total process:\t ");
Scanf ("%d", &n);
remain = n;
for ( count = 0; count <n; count++)
 {
printf ("Enter Arrival time & burst Time
 for process process number %d:",
 count +1);

Scanf ("%d", &at [count]);
  Scanf ("%d", & bt [count]);
  rt [count] = bt [count];
}
printf (" Enter time Quantum:\t ");
Scanf ("%d", & time - quantum);
printf ("\n\n Process\t |Turnaround Time |
 waiting Time \n\n");
```

```
for (time =0, count =0; remain! =0;)
{
if (rt [count ] <= time_quantum &&
      rt [count ] >0)
{
 time += rt [count];
 rt [count ]=0;
  flag =1;
 }
else if (rt [count] >0)
 {
 rt [count] - = time _ quantum;
 time += time _ qoantum;
 }
if (rt [count ]= =0 && flag ==1)
{
 remain --;
 printf (" P[%d] \t | \t %d \t | \t%d\n
 count +1, time -at [count], time -at
 [count], ~~time~~ -bt [count]);

wait_time += time -at [count] -bt[c
~~flag =0;~~
turnaround _time += time -at [count];
  flag =0;
 }
if (count== n-1)
 count =0;
else if (at [count +1] <= time)
 count ++;
else
 count =0;
}
printf ("\n Average waiting time = %f \n",
          wait _time * 1.0/(n));
printf ("Avg Turnaround Time = %.A",
   turnaround_time * 1.0 /n);
printfrant Chart (process ID, start time, endtime, n);
return 0;

}
```

O/P

```
Enter AT and BT for process Number 1 : 0 5
  "    "   "   "  "     "       "   2: 2 3
  "    "   "   "  "     "       "   3: 2 1
  "    "   "   "  "     "       "   4: 3 2
  "    "   "   "  "     "       "   5: 4 3
```

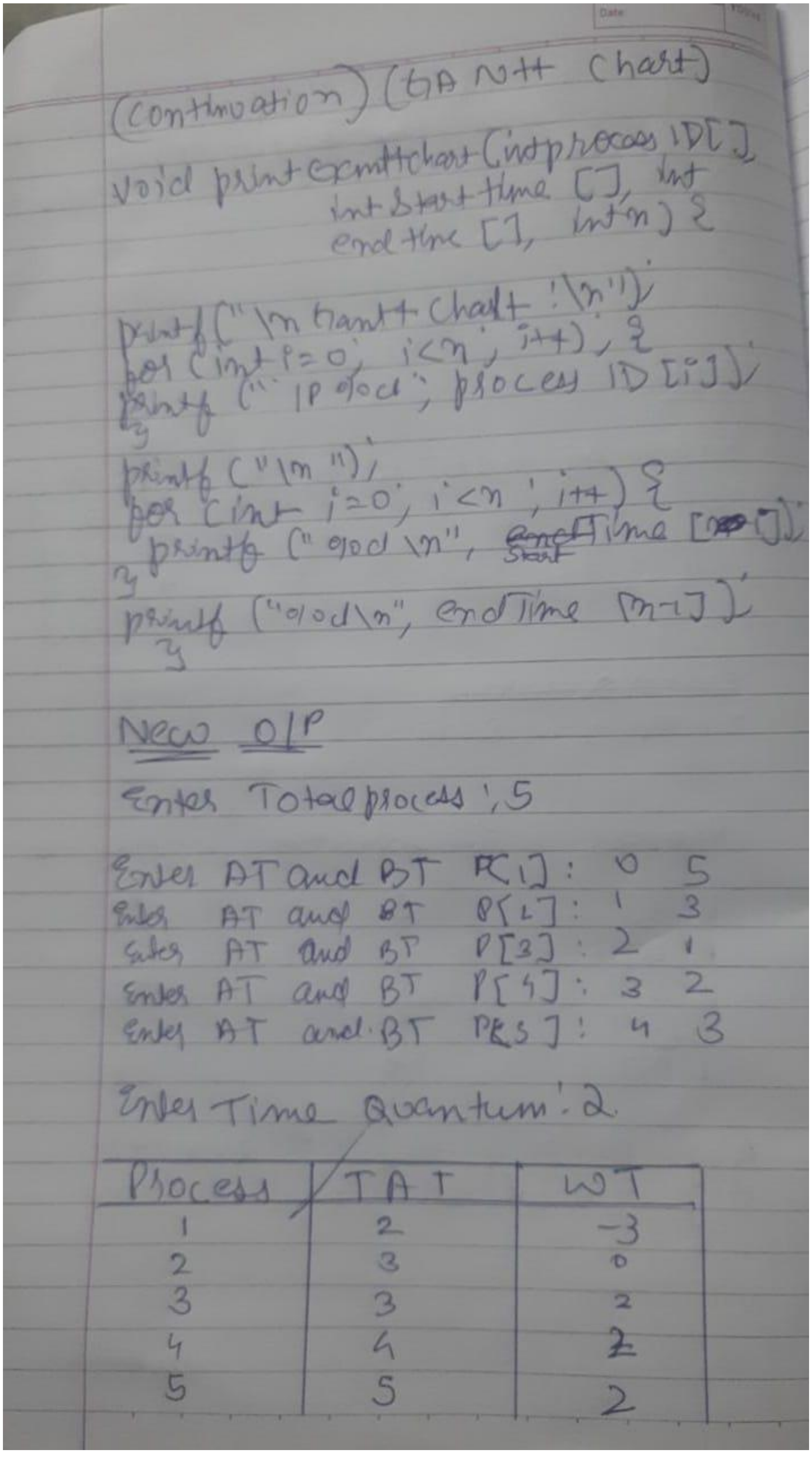
~~Enter Quantum~~

```
Enter time Quantum: 2

Process      Turnaround time    waiting time

P[3]              3                 2
P[4]              4                 2
P[2]              10                7
P[5]              9                 6
P[1]              14                9

Avg waiting time = 5.200000
Avg Turnaround time = 8.000000
```



(Continuation) (GAntt Chart)

```
void print Ganttchart (int process ID[],
               int Start time [], int
               end time [], int n) {

printf("\n Gantt Chart :\n");
for (int i=0; i<n; i++), {
printf(" |P %d", process ID[i]);
}

printf("\n");
for (int i=0; i<n ; i++) {
  printf(" %d \n", ~~end~~ Start Time [i]);
}
printf("%d\n", endTime [n-1]);
 }
```

New O/P

```
Enter Total process : 5

Enter AT and BT  P[1] :  0  5
Enter AT and BT  P[2] :  1  3
Enter AT and BT  P[3] :  2  1
Enter AT and BT  P[4] :  3  2
Enter AT and BT  P[5] :  4  3

Enter Time Quantum : 2
```

Process	TAT	WT
1	2	-3
2	3	0
3	3	2
4	4	2
5	5	2

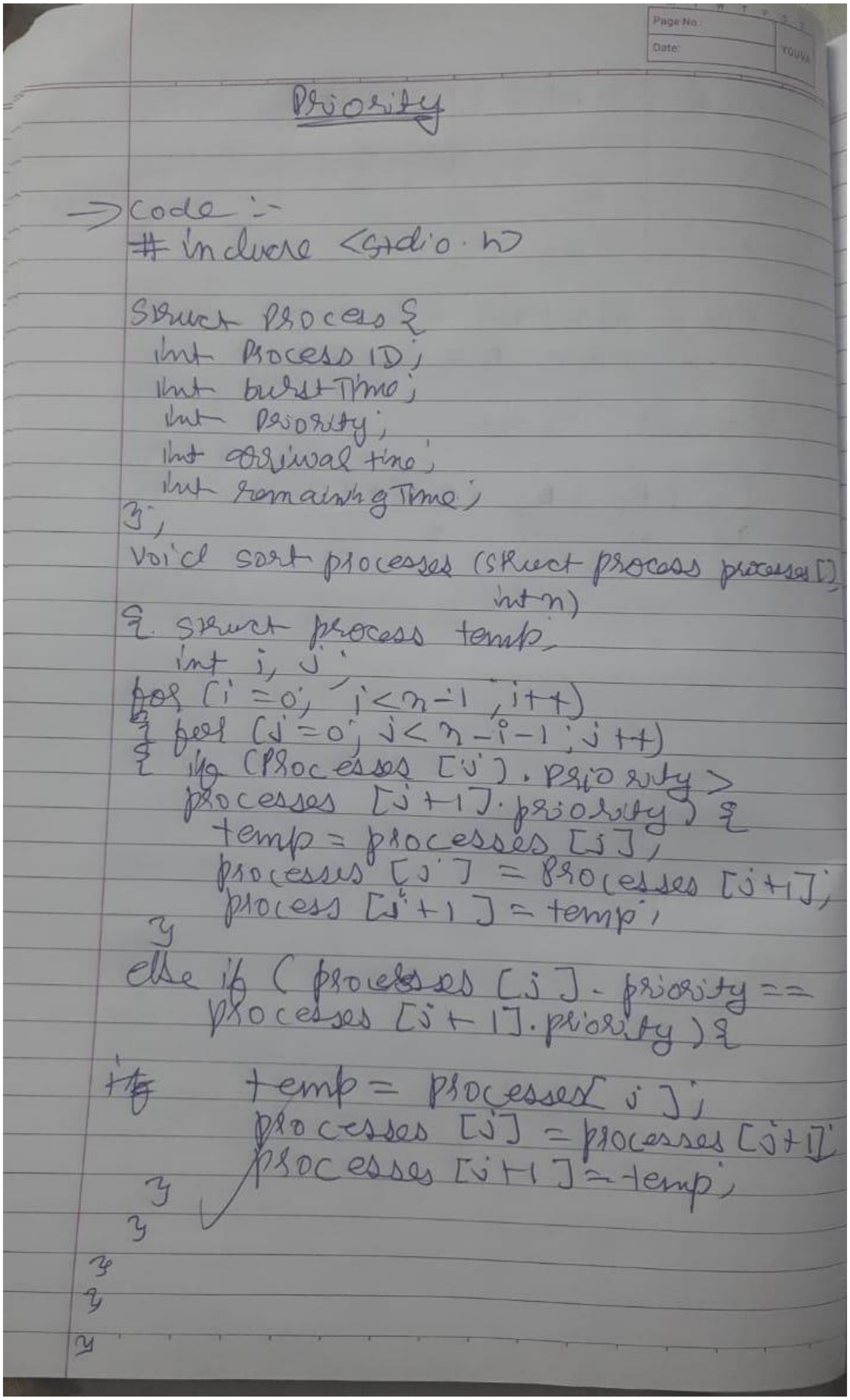
Process	TAT	WT
1	11	6
2	11	8
5	9	6
1	14	9

Avg waiting time =5. 40
Avg TAT ~~R~~ = 8.20

Ghant chart:

P1	P2	P3	P4	P5
13	11	4	5	12

PRIORITY:



Priority

→ code :-

```
#include <stdio.h>

struct process {
 int processID;
 int burstTime;
 int priority;
 int arrivaltime;
 int remainingTime;
};
void sort processes (struct process processes[],
                                  int n)
{ struct process temp,
   int i, j;
for (i = 0; i<n-1, i++)
{ for (j = 0; j< n-i-1; j++)
{ if (processes [j]. priority >
   processes [j+1]. priority) {
     temp = processes [j];
     processes [j] = processes [j+1];
     process [j+1] = temp;
  }
else if ( processes [j]. priority ==
     processes [j+1]. priority) {
~~if~~     temp = processes [j];
       processes [j] = processes [j+1];
       processes [j+1] = temp;
    }
  }
}
}
}
```

```
void schedule processesNonpreemptive(
 struct process processes[], int n){

int waitingTime = 0, turnaroundTime=0;
int i;

printf("\n process\tBurst
 Time\t priority\t ArrivalTime \twaiting
 Time \t Turnaround Time "),

for (i=0; i<n; i++) {
waitingTime += turnaroundTime -
process[i].arrivalTime;
    turnaroundTime += process[i].
burstTime, ~~processes~~
printf("\n %d\t %d\t \t%d\t\
        t%d \t\ t%d\t\t%d",
processes[i].processID, process[i].
burstTime, ~~p~~
process[i].priority, process[i].arrivalTime,
waitingTime, turnaroundtime);
}
printf("\nAWT: %.2f", (float) waiting
        time /n),
printf("\nATAT: %.2f", (float)
        turnaroundtime\n ),
}
void preemptive(struct process proc[],
                    int n) {
int waitingTime = 0, turnaroundTime =0;
int completedprocesses =0,
int currentTime =0;
int i;
printf("\n Gantt chart:\n");
```

```
while(completed processes (n)} {

int highest priority = -1;
int selected process = -1;
for (i=0; i<n; i++){
if (processes [i] .arrivalTime <=
 currentTime && processes [i]
 remainingTime = 0) {
if (highestpriority == -1 || process[i].
   priority <highest priority){
highestpriority = processes [i].priority;
selectedprocess = i;
  }
 }
}
if (selected process == -1){
    currentTime ++;
    continue;
 }
processes [selected process ]. remaining
time --;
current Time ++;
if (processes [selected process ]. remaining
Time == 0)
{
waitingTime += currentTime - processes
 [selected process] .arrivaltime - processes
 [selected process ]. burstTime;

turnaround time += currentTime -
 processes [selectedprocess] arrival time;
completed processes ++;
printf ("P %d", processes [selected
 process] . processID);
 }
}
```

```
printf ("\n \n process\t BT \ priority \t
   AT \WT \t TAT"),
for (i = 0; i < n; i++) {
printf (" \n %d\t %d \t %d\t
      %d \t %d\t %d\t "
 process [i] . processID , process [i].
  burstTime , processes [i] . priority,
process [i] . arrivalTime . waitingTime,
turnaroundTime);
}
print f (" \n AWT : %.2f", (float)
        waitingTime \n),
printf (" \n ATAT : %.2f ", (float)
            turnaroundTime \n),
}
int main () {
 int n, i, option;
 printf ("Enter the number of processes:")
 Scanf ("%d", &n),

struct process processes [n];
for (i = 0; i < n; i++) {
printf (" \n Enter details for process %d",
          i +1),
processes [i] . process ID = i+1;
print f ("\n Enter burst Time : "),
scanf ("%d" & process [i]. burstTime)
printf ("\n Enter priority : "),
scanf ("%d, & process [i] . priority);
printf ("Enter arrival time : "),
 scanf (" %d ", & processes [i].
          arrival Time)
process [i] . remaining Time = process[i]
                burst time;
}
```

```
printf("select the scheduling algorith
printf(\n 1.nonpreimtive \n 2 Preemitive"
 printf ("enter your choice : ");
 Scanf ("%d", & option);
 switch (option) {
 case 1: Short processes (Processes, n);

Non preemptive (processes, n);
 break;

case 2:
    preemptive (processes, n);
    break;

default: printf("\n Invalid choice");

   break
 }
return 0;
}
```

O/P

```
Enter the number of processes : 4
Enter details of process 1:
 BT : 4
 Prt : 3
 AT : 0
Enter details of process 2:
  BT : 3
  Prt : 4
  AT : 1
```

Enter details of process 3.

BT : 3
Prt : 6
AT : 2

Enter details of process 4 :

BT : 5
Prt : 5
AT : 3

select scheduling algorithm :

1. Non preemptive
2. preemptive.

Enter your choice : 2

Gantt chart :

| P_1 | P_2 | P_4 | P_3 |

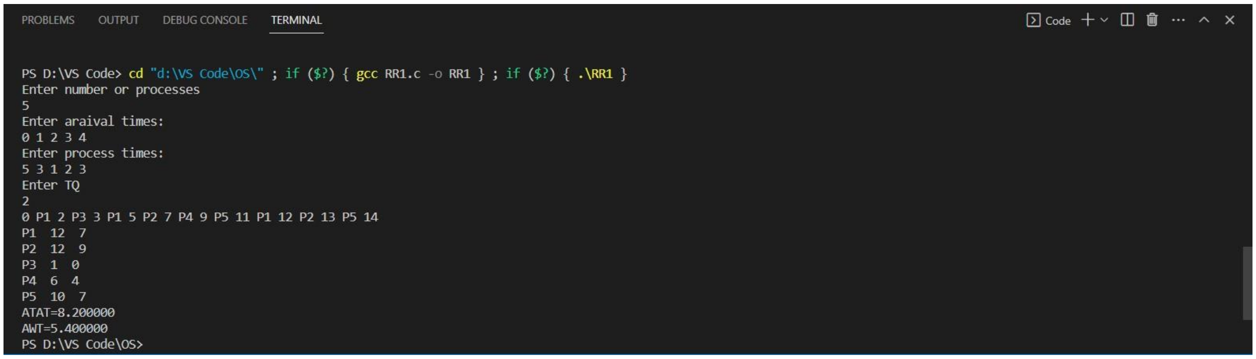
Process	BT	Prt	AT	WT	TAT
1	4	3	0	17	32
2	3	4	1	17	32
3	3	6	2	17	32
4	5	5	3	17	32

AwT : 4.25
ATAT : 8.00

8/7/23

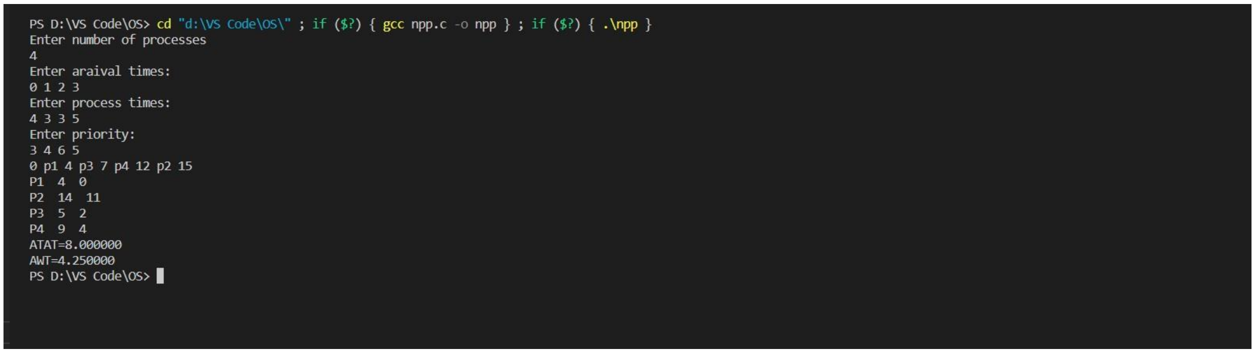
OUTPUT:

ROUND ROBIN OUTPUT:



```
PS D:\VS Code> cd "d:\VS Code\OS\" ; if ($?) { gcc RR1.c -o RR1 } ; if ($?) { .\RR1 }
Enter number or processes
5
Enter araival times:
0 1 2 3 4
Enter process times:
5 3 1 2 3
Enter TQ
2
0 P1 2 P3 3 P1 5 P2 7 P4 9 P5 11 P1 12 P2 13 P5 14
P1  12  7
P2  12  9
P3  1  0
P4  6  4
P5  10  7
ATAT=8.200000
AWT=5.400000
PS D:\VS Code\OS>
```

PRIORITY OUTPUT:



```
PS D:\VS Code\OS> cd "d:\VS Code\OS\" ; if ($?) { gcc npp.c -o npp } ; if ($?) { .\npp }
Enter number of processes
4
Enter araival times:
0 1 2 3
Enter process times:
4 3 3 5
Enter priority:
3 4 6 5
0 p1 4 p3 7 p4 12 p2 15
P1  4  0
P2  14  11
P3  5  2
P4  9  4
ATAT=8.000000
AWT=4.250000
PS D:\VS Code\OS>
```