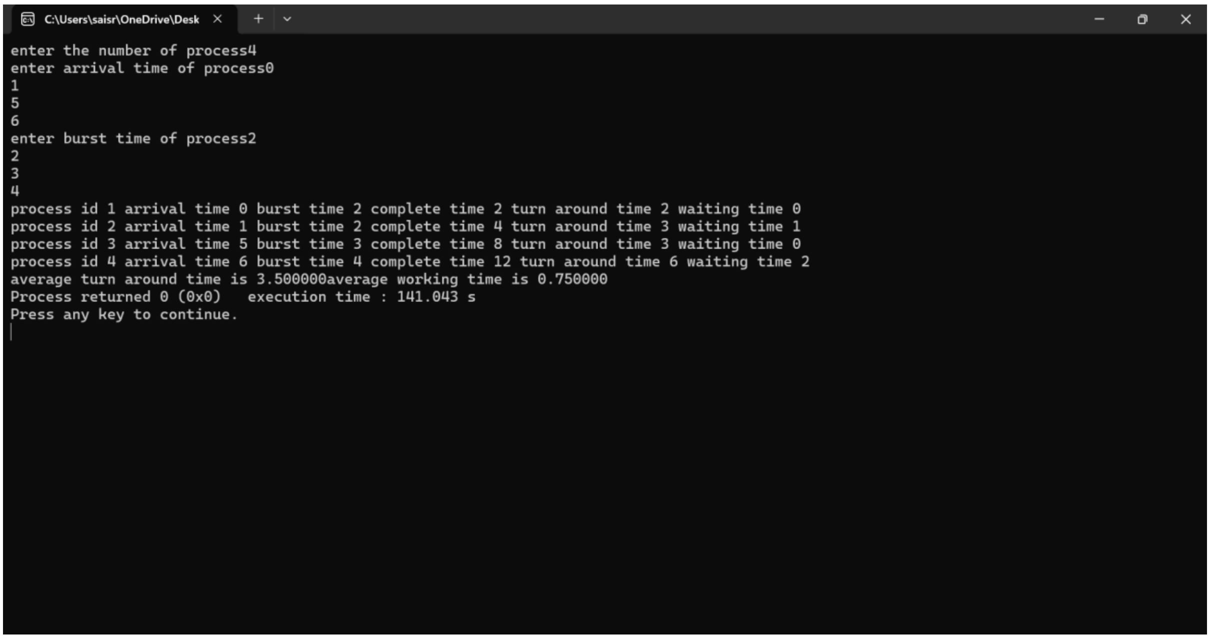
1.FCFS scheduling using array

```
#include<stdio.h>
int main(){
   int n,i;
   float atat=0,awt=0;
   printf("enter the number of process");
   scanf("%d",&n);
   int atime1[n],btime2[n],ctime3[n],tattime4[n],wtime5[n];
   printf("enter arrival time of process");
   for(i=0;i<n;i++){
      scanf("%d",&atime1[i]);
   }
   printf("enter burst time of process");
   for(i=0;i<n;i++){
      scanf("%d",&btime2[i]);
   }
   for(i=0;i<n;i++){
      if(i==0){
         ctime3[i]=atime1[i]+btime2[i];
      }
      else{
         if(ctime3[i-1]<atime1[i]){
            ctime3[i]=(atime1[i]-ctime3[i-1])+ctime3[i-1]+btime2[i];
         }
         else{
            ctime3[i]=ctime3[i-1]+btime2[i];
         }
      }
   }
   for(i=0;i<n;i++){
      tattime4[i]=ctime3[i]-atime1[i];
   }
   for(i=0;i<n;i++){
      wtime5[i]=tattime4[i]-btime2[i];
   }
   for(i=0;i<n;i++){
      atat=atat+tattime4[i];
   }
   atat=(atat/n);
   for(i=0;i<n;i++){
      awt=awt+wtime5[i];
   }
   awt=(awt/n);
   for(i=0;i<n;i++){
      printf("process id %d arrival time %d burst time %d complete time %d turn around time %d waiting time %d\n",i+1,atime1[i],btime2[i],ctime3[i],tattime4[i],wtime5[i]);
   }
   printf("average turn around time is %f",atat);
   printf("average working time is %f",awt);
}
```

output:



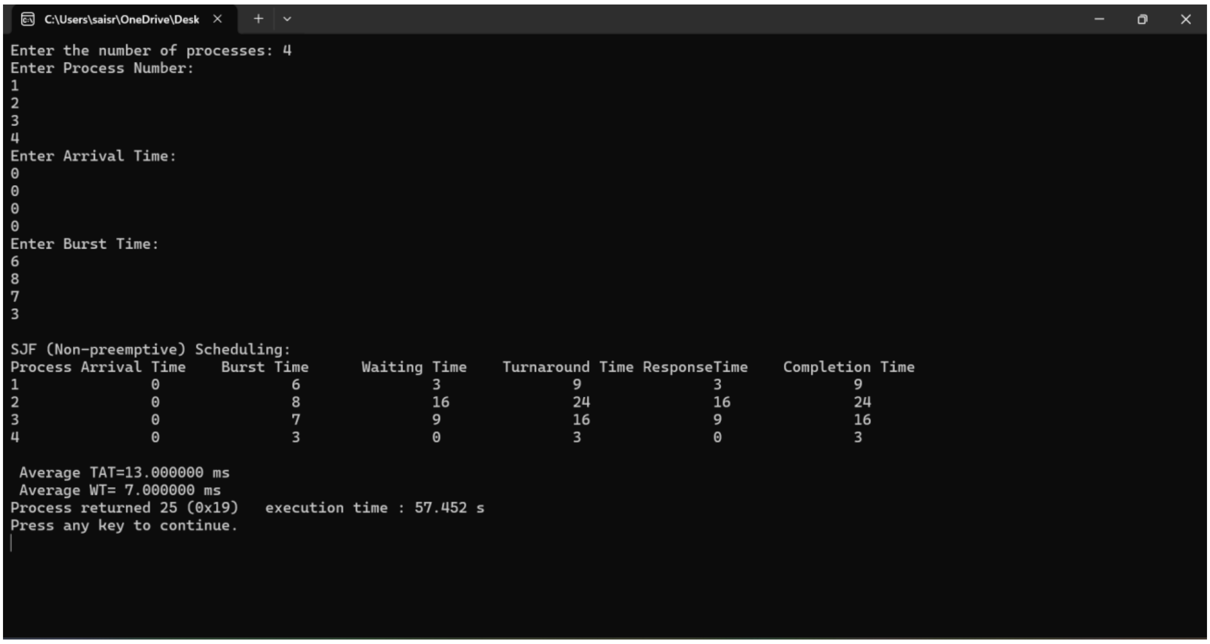
```
enter the number of process4
enter arrival time of process0
1
5
6
enter burst time of process2
2
3
4
process id 1 arrival time 0 burst time 2 complete time 2 turn around time 2 waiting time 0
process id 2 arrival time 1 burst time 2 complete time 4 turn around time 3 waiting time 1
process id 3 arrival time 5 burst time 3 complete time 8 turn around time 3 waiting time 0
process id 4 arrival time 6 burst time 4 complete time 12 turn around time 6 waiting time 2
average turn around time is 3.500000average working time is 0.750000
Process returned 0 (0x0)   execution time : 141.043 s
Press any key to continue.
```

2.SJF(non-preemptive)scheduling using array

```
#include<stdio.h>
 void findCompletionTime(int processes[], int n, int bt[], int at[], int wt[], int tat[], int rt[], int ct[])
 {
 int completion[n];
 int remaining[n];
 for (int i = 0; i < n; i++)
 remaining[i] = bt[i];
 int currentTime = 0;
 for (int i = 0; i < n; i++)
 {
 int shortest =-1;
 for (int j = 0; j < n; j++)
 {
 if (at[j] <= currentTime && remaining[j] > 0)
 {
 if (shortest ==-1 || remaining[j] < remaining[shortest])
 shortest = j;
 }
 }
 if (shortest ==-1)
 {
 currentTime++;
 continue;
 }
 completion[shortest] = currentTime + remaining[shortest];
 currentTime = completion[shortest];
 wt[shortest] = currentTime- bt[shortest]- at[shortest];
 tat[shortest] = currentTime- at[shortest];
 rt[shortest] = wt[shortest];
 remaining[shortest] = 0;
 }
 printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\tResponseTime\tCompletion Time\n");
 for (int i = 0; i < n; i++)
 {
 ct[i] = completion[i];
 printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], at[i], bt[i], wt[i], tat[i], rt[i], ct[i]);
 }
 float avg_tat=tat[0];
 for(int i=1;i<n;i++)
 {
avg_tat+=tat[i];
 }
 printf("\n Average TAT=%f ms",avg_tat/n);
 float avg_wt=wt[0];
 for(int i=1;i<n;i++)
 {
 avg_wt+=wt[i];
 }
 printf("\n Average WT= %f ms",avg_wt/n);
 }
 void main()
 {
 int n;
 printf("Enter the number of processes: ");
 scanf("%d", &n);
 int processes[n];
 int burst_time[n];
 int arrival_time[n];
 printf("Enter Process Number:\n");
 for (int i = 0; i < n; i++)
```

```
{
scanf("%d", & processes[i]);
}
printf("Enter Arrival Time:\n");
for (int i = 0; i < n; i++)
{
scanf("%d", &arrival_time[i]);
}
printf("Enter Burst Time:\n");
for (int i = 0; i < n; i++)
{
scanf("%d", &burst_time[i]);
}
int wt[n], tat[n], rt[n], ct[n];
for (int i = 0; i < n; i++)
rt[i] =-1;
printf("\nSJF (Non-preemptive) Scheduling:\n");
findCompletionTime(processes, n, burst_time, arrival_time, wt, tat, rt, ct);
}
```

output:

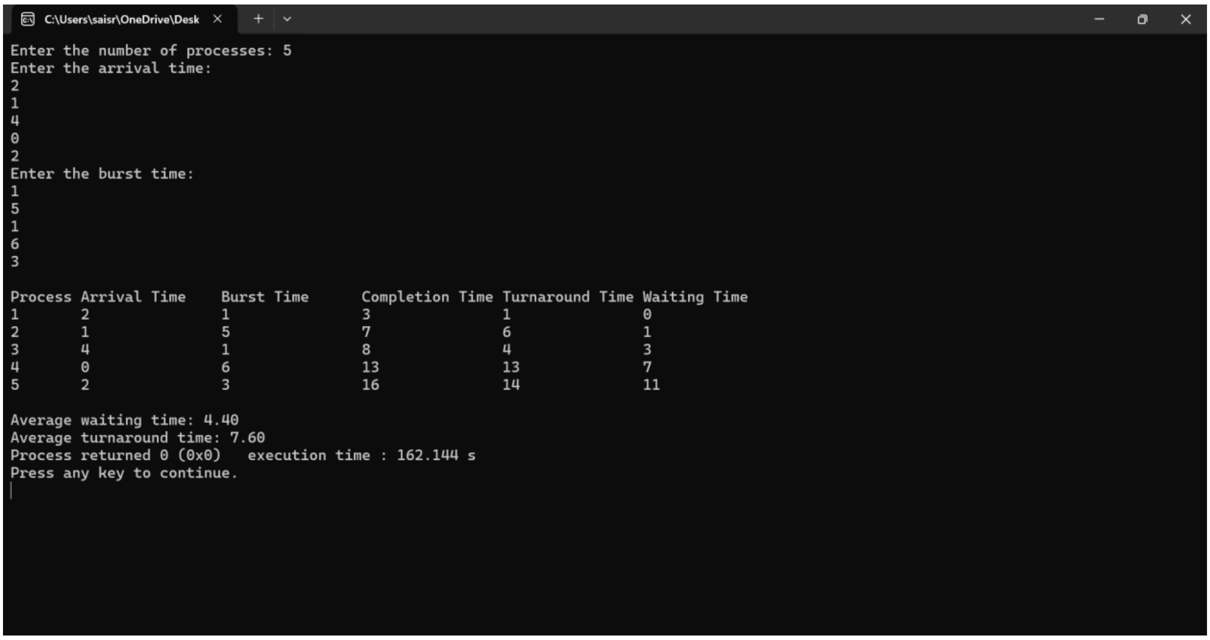


```
Enter the number of processes: 4
Enter Process Number:
1
2
3
4
Enter Arrival Time:
0
0
0
0
Enter Burst Time:
6
8
7
3

SJF (Non-preemptive) Scheduling:
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time ResponseTime    Completion Time
1               0               6               3               9               3               9
2               0               8               16              24              16              24
3               0               7               9               16              9               16
4               0               3               0               3               0               3

 Average TAT=13.000000 ms
 Average WT= 7.000000 ms
Process returned 25 (0x19)   execution time : 57.452 s
Press any key to continue.
```

3.SJF(preemptive)scheduling using array

```
#include <stdio.h>
#define MAX 10
int find_min(int arr[], int n) {
    int min = arr[0];
    int index = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
            index = i;
        }
    }
    return index;
}
void sjf_preemptive(int n, int at[], int bt[]) {
    int ct[MAX] = {0};
    int tat[MAX] = {0};
    int wt[MAX] = {0};
    int rt[MAX];
    int total_wt = 0;
    int total_tat = 0;
    for (int i = 0; i < n; i++) {
        rt[i] = bt[i];
    }
    int current_time = 0;
    int completed_processes = 0;
    while (completed_processes < n) {
        int available_processes[MAX];
        int available_count = 0;
        for (int i = 0; i < n; i++) {
            if (at[i] <= current_time && rt[i] > 0) {
                available_processes[available_count] = i;
                available_count++;
            }
        }
        if (available_count == 0) {
            current_time++;
            continue;
        }
        int shortest_job_index = available_processes[find_min(rt, available_count)];
        rt[shortest_job_index]--;
        current_time++;
        if (rt[shortest_job_index] == 0) {
            completed_processes++;
            ct[shortest_job_index] = current_time;
            tat[shortest_job_index] = ct[shortest_job_index] - at[shortest_job_index];
            wt[shortest_job_index] = tat[shortest_job_index] - bt[shortest_job_index];
            total_wt += wt[shortest_job_index];
            total_tat += tat[shortest_job_index];
        }
    }
    printf("\nProcess\tArrival Time\tBurst 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", i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    printf("\nAverage waiting time: %.2f", (float)total_wt / n);
    printf("\nAverage turnaround time: %.2f", (float)total_tat / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[MAX], bt[MAX];
    printf("Enter the arrival time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &at[i]);
    }
    printf("Enter the burst time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
    }
    sjf_preemptive(n, at, bt);
    return 0;
}
```

output:



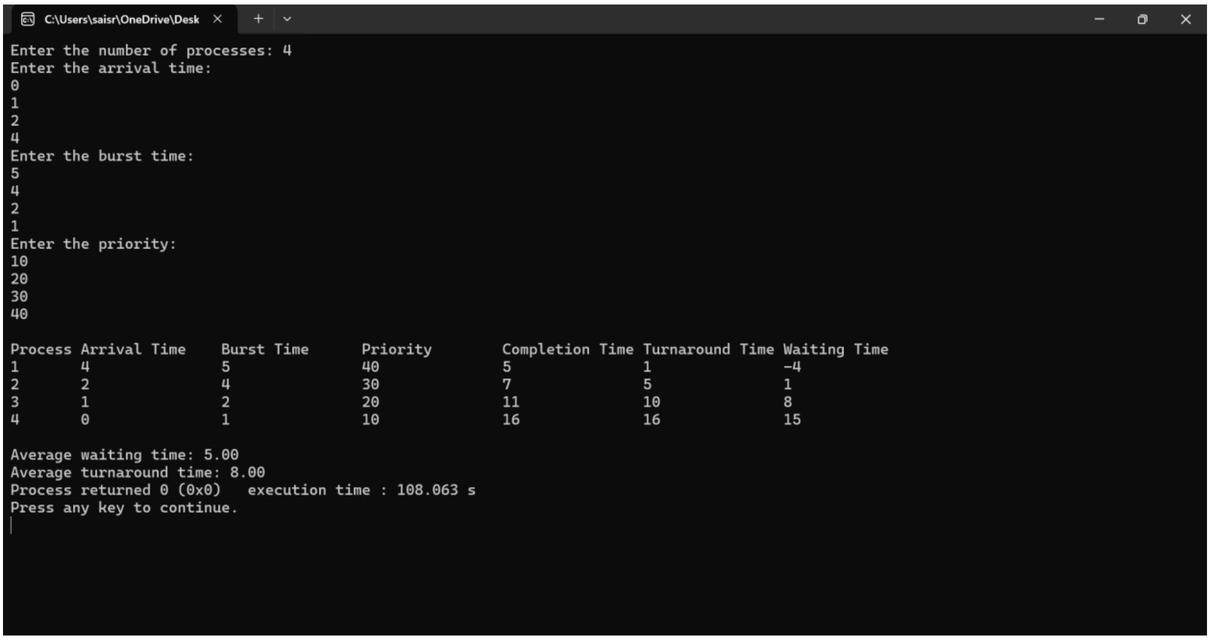
```
Enter the number of processes: 5
Enter the arrival time:
2
1
4
0
2
Enter the burst time:
1
5
1
6
3

Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       2               1               3               1               0
2       1               5               7               6               1
3       4               1               8               4               3
4       0               6               13              13              7
5       2               3               16              14              11

Average waiting time: 4.40
Average turnaround time: 7.60
Process returned 0 (0x0)   execution time : 162.144 s
Press any key to continue.
```

4.priority(non preemptive)scheduling

```
#include<stdio.h>
#define MAX 10
void priority_non_preemptive(int n, int at[], int bt[], int p[]) {
    int ct[MAX] = {0};
    int tat[MAX] = {0};
    int wt[MAX] = {0};
    int total_wt = 0;
    int total_tat = 0;
    int bt_copy[MAX];
    for (int i = 0; i < n; i++) {
        bt_copy[i] = bt[i];
    }
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i] < p[j]) {
                int temp = at[i];
                at[i] = at[j];
                at[j] = temp;
                temp = bt[i];
                bt[i] = bt[j];
                bt[j] = temp;
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
    ct[0] = at[0] + bt[0];
    tat[0] = ct[0] - at[0];
    wt[0] = tat[0] - bt_copy[0];
    total_wt += wt[0];
    total_tat += tat[0];
    for (int i = 1; i < n; i++) {
        ct[i] = ct[i - 1] + bt[i];
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt_copy[i];
```

```
        total_wt += wt[i];
        total_tat += tat[i];
    }
    printf("\nProcess\tArrival Time\tBurst Time\tPriority\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\t\t%d\n", i+1, at[i], bt_copy[i], p[i], ct[i], tat[i], wt[i]);
    }
    printf("\nAverage waiting time: %.2f", (float)total_wt / n);
    printf("\nAverage turnaround time: %.2f", (float)total_tat / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int at[MAX], bt[MAX], p[MAX];
    printf("Enter the arrival time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &at[i]);
    }
    printf("Enter the burst time:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &bt[i]);
    }
    printf("Enter the priority:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i]);
    }
    priority_non_preemptive(n, at, bt, p);
    return 0;
}
```

output:



```
Enter the number of processes: 4
Enter the arrival time:
0
1
2
4
Enter the burst time:
5
4
2
1
Enter the priority:
10
20
30
40

Process Arrival Time    Burst Time      Priority        Completion Time Turnaround Time Waiting Time
1       4               5               40              5               1               -4
2       2               4               30              7               5               1
3       1               2               20              11              10              8
4       0               1               10              16              16              15

Average waiting time: 5.00
Average turnaround time: 8.00
Process returned 0 (0x0)   execution time : 108.063 s
Press any key to continue.
```

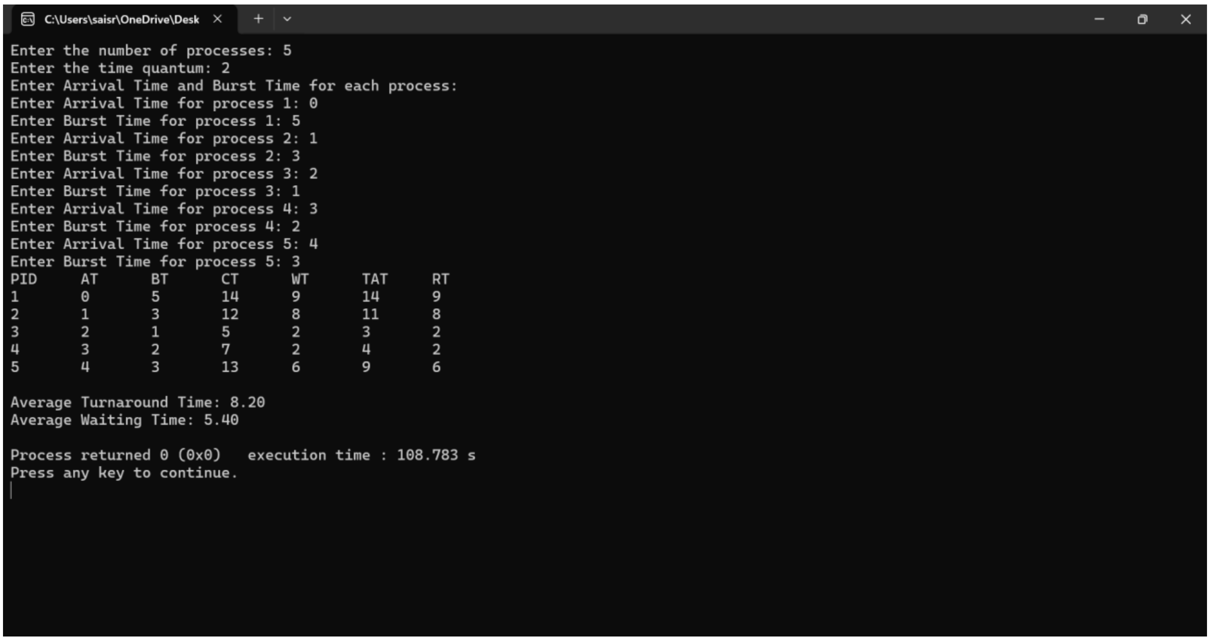
5.Round robin scheduling

```
#include <stdio.h>
 struct Process {
 int pid;
 int burst_time;
 int arrival_time;
 int remaining_time;
 };
 void roundRobin(struct Process processes[], int n, int time_quantum) {
 int remaining_processes = n;
 int current_time = 0;
 int completed[n];
 int ct[n], wt[n], tat[n], rt[n];
 for (int i = 0; i < n; i++) {
 completed[i] = 0;
 }
 while (remaining_processes > 0) {
 for (int i = 0; i < n; i++) {
 if (completed[i] == 0 && processes[i].arrival_time <= current_time) {
```

```
if (processes[i].remaining_time > 0) {
if (processes[i].remaining_time <= time_quantum) {
current_time += processes[i].remaining_time;
processes[i].remaining_time = 0;
completed[i] = 1;
remaining_processes--;
ct[i] = current_time;
tat[i] = ct[i]- processes[i].arrival_time;
} else {
current_time += time_quantum;
processes[i].remaining_time-= time_quantum;
}
wt[i] = ct[i]- processes[i].arrival_time- processes[i].burst_time;
rt[i] = wt[i];
}
}
}
}
printf("PID\tAT\tBT\tCT\tWT\tTAT\tRT\n");
float avg_tat = 0, avg_wt = 0;
for (int i = 0; i < n; i++) {
printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i].pid, processes[i].arrival_time,
processes[i].burst_time, ct[i], wt[i], tat[i], rt[i]);
avg_tat += tat[i];
avg_wt += wt[i];
}
avg_tat /= n;
avg_wt /= n;
printf("\nAverage Turnaround Time: %.2f\n", avg_tat);
printf("Average Waiting Time: %.2f\n", avg_wt);
}
```

```
int main() {
int n, time_quantum;
printf("Enter the number of processes: ");
scanf("%d", &n);
printf("Enter the time quantum: ");
scanf("%d", &time_quantum);
struct Process processes[n];
printf("Enter Arrival Time and Burst Time for each process:\n");
for (int i = 0; i < n; i++) {
printf("Enter Arrival Time for process %d: ", i+1);
scanf("%d", & processes[i].arrival_time);
printf("Enter Burst Time for process %d: ", i+1);
scanf("%d", & processes[i].burst_time);
processes[i].pid = i+1;
processes[i].remaining_time = processes[i].burst_time;
}
roundRobin(processes, n, time_quantum);
}
```

output:



```
Enter the number of processes: 5
Enter the time quantum: 2
Enter Arrival Time and Burst Time for each process:
Enter Arrival Time for process 1: 0
Enter Burst Time for process 1: 5
Enter Arrival Time for process 2: 1
Enter Burst Time for process 2: 3
Enter Arrival Time for process 3: 2
Enter Burst Time for process 3: 1
Enter Arrival Time for process 4: 3
Enter Burst Time for process 4: 2
Enter Arrival Time for process 5: 4
Enter Burst Time for process 5: 3
PID     AT      BT      CT      WT      TAT     RT
1       0       5       14      9       14      9
2       1       3       12      8       11      8
3       2       1       5       2       3       2
4       3       2       7       2       4       2
5       4       3       13      6       9       6

Average Turnaround Time: 8.20
Average Waiting Time: 5.40

Process returned 0 (0x0)   execution time : 108.783 s
Press any key to continue.
```