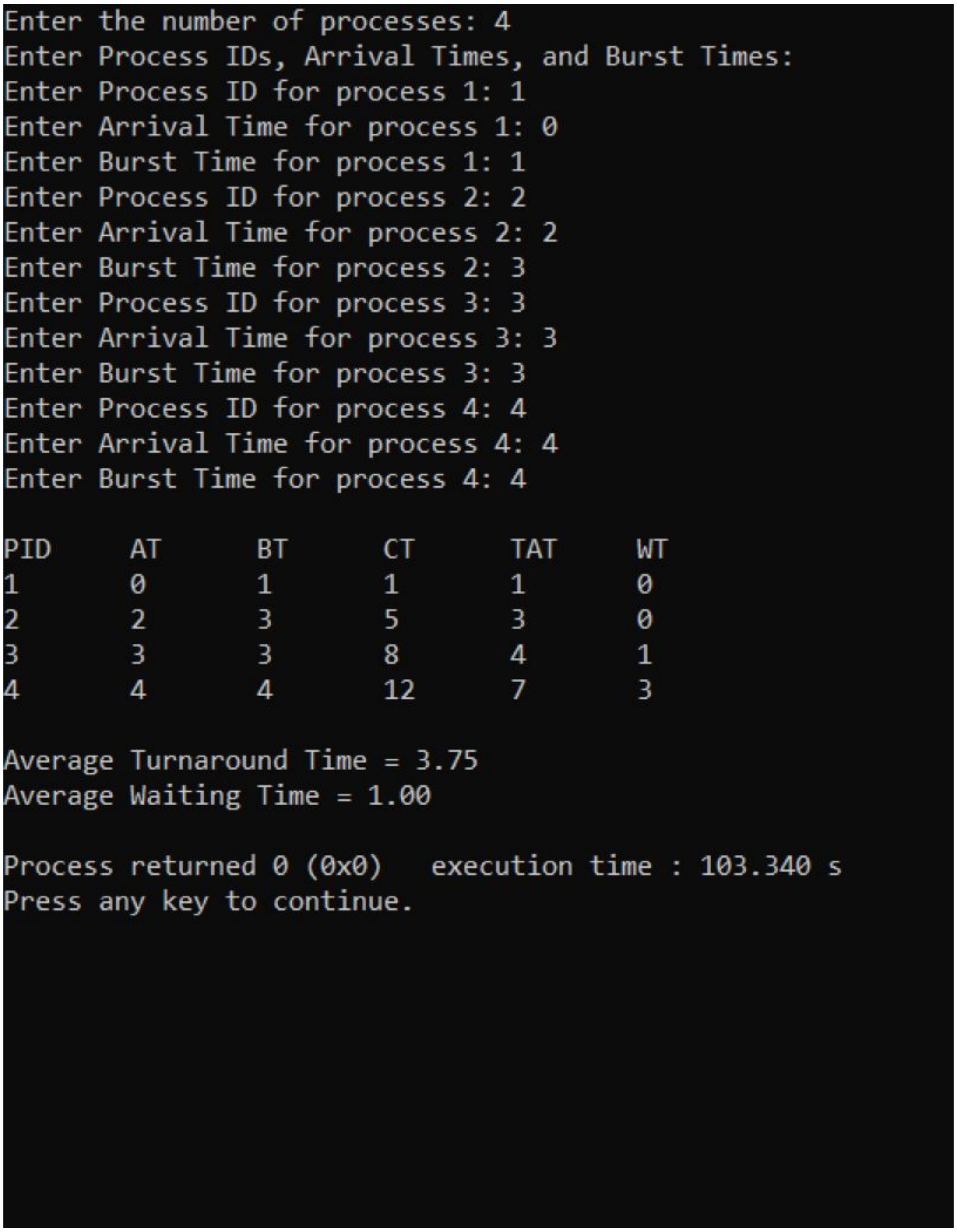
1.**FCFS scheduling using array.**

```
 #include <stdio.h>
 void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[], int ct[])
    { int service_time[n];
    service_time[0] = 0;
    for (int i = 1; i < n; i++) {
        service_time[i] = service_time[i - 1] + bt[i - 1];
    }


    for (int i = 0; i < n; i++) {
        wt[i] = service_time[i] - at[i];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}
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 findCompletionTime(int processes[], int n, int bt[], int at[], int ct[])
    { int service_time[n];
    service_time[0] = at[0] + bt[0];
    for (int i = 1; i < n; i++) {

        service_time[i] = (service_time[i - 1] > at[i]) ? service_time[i - 1] + bt[i] : at[i] + bt[i];
    }
    for (int i = 0; i < n; i++)
        ct[i] = service_time[i];
}
void findAvgTime(int processes[], int n, int bt[], int at[]) {
    int wt[n], tat[n], ct[n];
    findCompletionTime(processes, n, bt, at, ct);
    findWaitingTime(processes, n, bt, wt, at, ct);
    findTurnAroundTime(processes, n, bt, wt, tat);
    float total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
    }
    float avg_wt = total_wt / n;
    float avg_tat = total_tat / n;
    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    printf("\nAverage Turnaround Time = %.2f\n", avg_tat);
    printf("Average Waiting Time = %.2f\n", avg_wt);
}
int main()
    { int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    int processes[n];
    int bt[n];
    int at[n];
    printf("Enter Process IDs, Arrival Times, and Burst Times:\n");
    for (int i = 0; i < n; i++) {
        printf("Enter Process ID for process %d: ", i + 1);
        scanf("%d", & processes[i]);
        printf("Enter Arrival Time for process %d: ", i + 1);
        scanf("%d", &at[i]);
        printf("Enter Burst Time for process %d: ", i + 1);
        scanf("%d", &bt[i]);
    }
    findAvgTime(processes, n, bt, at);
    return 0;
}
```

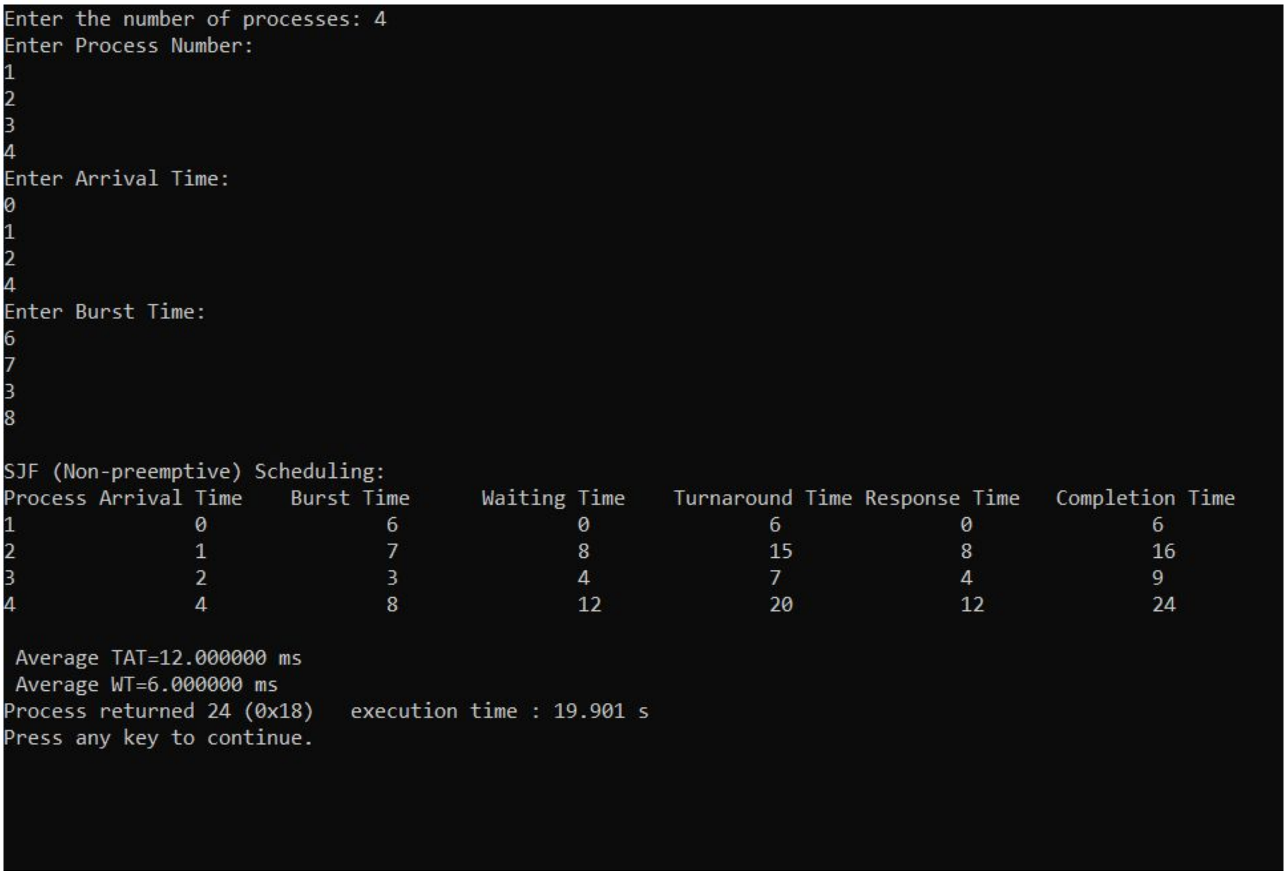


```
Enter the number of processes: 4
Enter Process IDs, Arrival Times, and Burst Times:
Enter Process ID for process 1: 1
Enter Arrival Time for process 1: 0
Enter Burst Time for process 1: 1
Enter Process ID for process 2: 2
Enter Arrival Time for process 2: 2
Enter Burst Time for process 2: 3
Enter Process ID for process 3: 3
Enter Arrival Time for process 3: 3
Enter Burst Time for process 3: 3
Enter Process ID for process 4: 4
Enter Arrival Time for process 4: 4
Enter Burst Time for process 4: 4

PID     AT      BT      CT      TAT     WT
1       0       1       1       1       0
2       2       3       5       3       0
3       3       3       8       4       1
4       4       4       12      7       3

Average Turnaround Time = 3.75
Average Waiting Time = 1.00

Process returned 0 (0x0)   execution time : 103.340 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\tResponse
Time\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++)
{
```

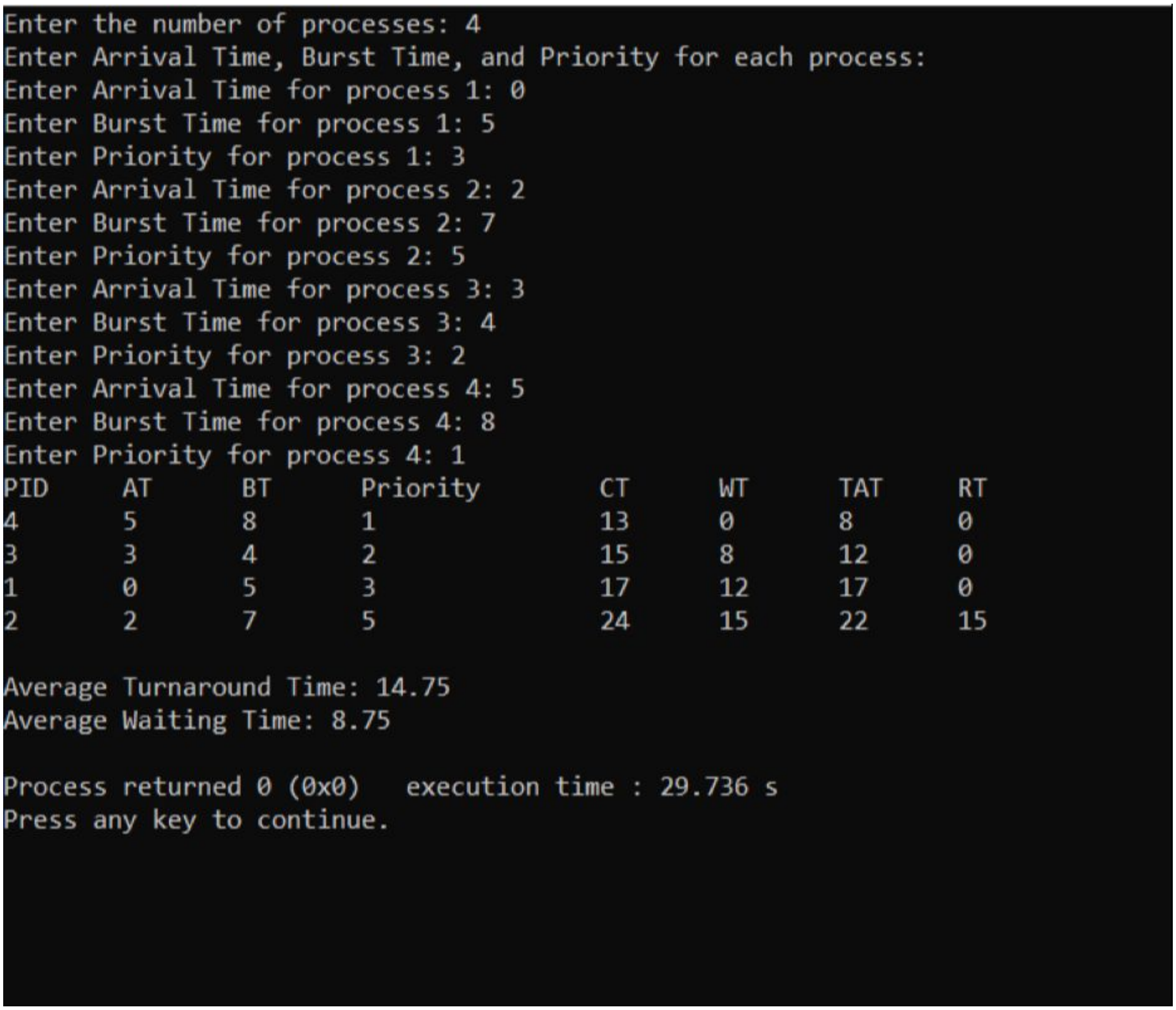
```
avg_tat+=tat[i];
}
printf("\n Average TAT=%f ms",avg_tat/n);
float avg_wt=wt[0];
for(int i=1;i<n;i++)
{
avg_wt+=wt[i];
}
printf("\n Average WT= %f ms",avg_wt/n);
}
void main()
{
int n;
printf("Enter the number of processes: ");
scanf("%d", &n);
int   processes[n];
int   burst_time[n];
int arrival_time[n];
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);
}
```



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

SJF (Non-preemptive) Scheduling:
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time Response Time   Completion Time
1               0               6               0               6               0               6
2               1               7               8               15              8               16
3               2               3               4               7               4               9
4               4               8               12              20              12              24

 Average TAT=12.000000 ms
 Average WT=6.000000 ms
Process returned 24 (0x18)   execution time : 19.901 s
Press any key to continue.
```

3.**priority(preemptive)scheduling**

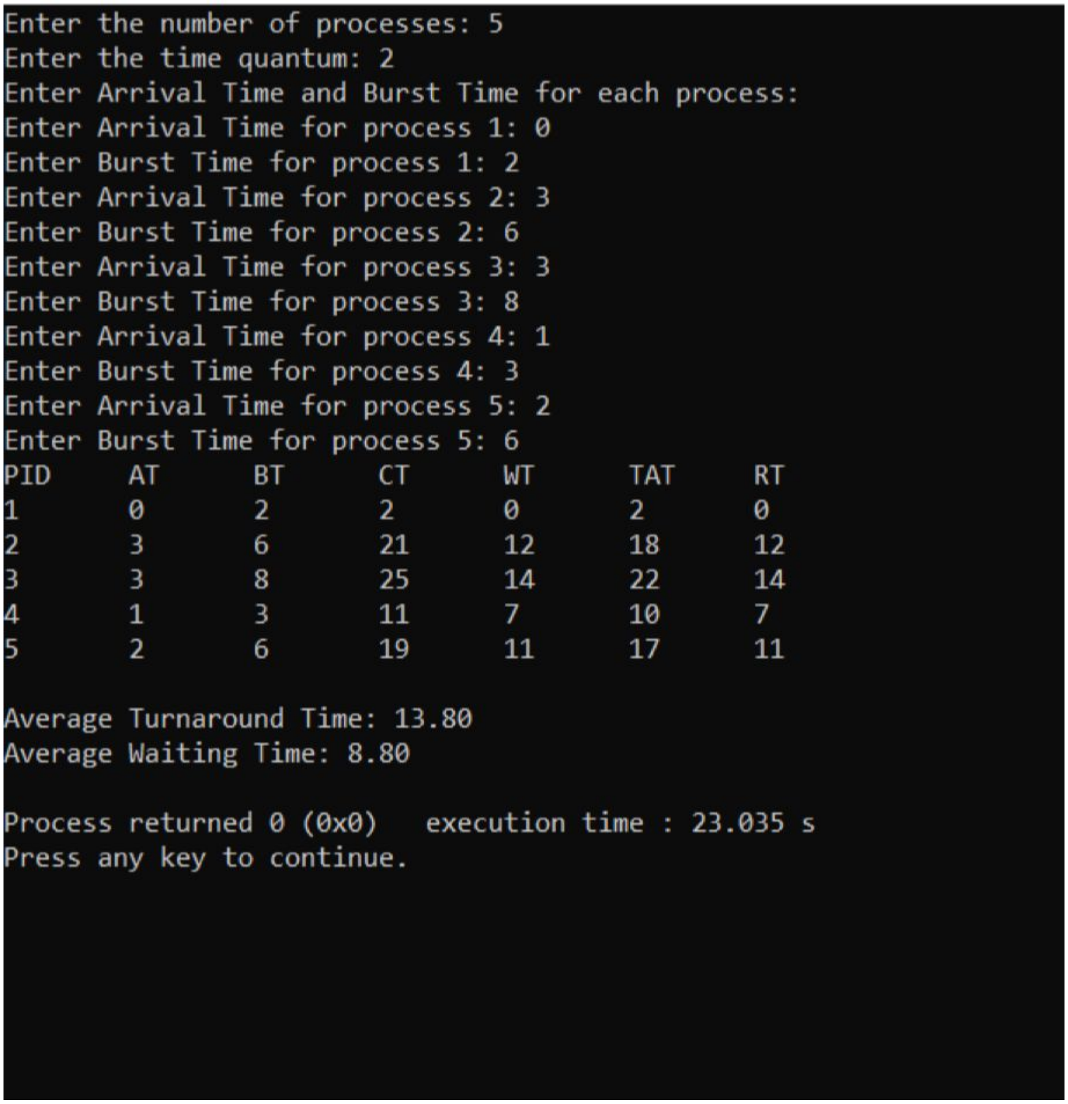
```
#include<stdio.h>
#include<limits.h>
struct Process {
    int pid;
    int burst_time;
    int priority;
    int arrival_time;
    int remaining_time;
    int start_time;
};
void swap(struct Process *a, struct Process *b)
    { struct Process temp = *a;
    *a = *b;
    *b = temp;
}
void sort(struct Process processes[], int n)
    { for (int i = 0; i < n-1; i++) {
        int min_index = i;
        for (int j = i+1; j < n; j++) {
            if (processes[j].priority < processes[min_index].priority)
                { min_index = j;
            }
        }
        swap( & processes[min_index], & processes[i]);
    }
}
void findCompletionTime(struct Process processes[], int n, int ct[])
    { int current_time = 0;
    int completed = 0;

    while (completed != n)
        { int selected = -1;
        int highest_priority = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= current_time && processes[i].remaining_time > 0)
                { if (processes[i].priority < highest_priority) {
                    highest_priority = processes[i].priority;
                    selected = i;
                }
            }
        }
        if (selected == -1)
            { current_time++;
        } else {
            if (processes[selected].start_time == -1)
                { processes[selected].start_time = current_time;
            }
            processes[selected].remaining_time--;
            current_time++;
            if (processes[selected].remaining_time == 0)
                 { Completed ++;
                ct[selected] = current_time;
            }
        }
    }
}
void findTurnAroundTime(struct Process processes[], int n, int ct[], int tat[]) {
    for (int i = 0; i < n; i++) {
        tat[i] = ct[i] - processes[i].arrival_time;
    }
}
void findWaitingTime(struct Process processes[], int n, int wt[], int tat[])
    { for (int i = 0; i < n; i++) {
        wt[i] = tat[i] - processes[i].burst_time;
}
void display(struct Process processes[], int n, int ct[], int wt[], int tat[], int rt[])
    { printf("PID\tAT\tBT\tPriority\tCT\tWT\tTAT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\t%d\n", processes[i].pid, processes[i].arrival_time,
processes[i].burst_time, processes[i].priority, ct[i], wt[i], tat[i], rt[i]);
    }
}
int main()
    { int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    int ct[n], wt[n], tat[n], rt[n];
    printf("Enter Arrival Time, Burst Time, and Priority 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);
        printf("Enter Priority for process %d: ", i+1);
        scanf("%d", & processes[i].priority);
        processes[i].pid = i+1;
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].start_time = -1;
    }
    sort(processes, n);
    findCompletionTime(processes, n, ct);
    findTurnAroundTime(processes, n, ct, tat);
    findWaitingTime(processes, n, wt, tat);
    for (int i = 0; i < n; i++) {
        rt[i] = processes[i].start_time - processes[i].arrival_time;
    }
    display(processes, n, ct, wt, tat, rt);
    float avg_tat = 0, avg_wt = 0;
    for (int i = 0; i < n; 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);

    return 0;
}
```



```
Enter the number of processes: 4
Enter Arrival Time, Burst Time, and Priority for each process:
Enter Arrival Time for process 1: 0
Enter Burst Time for process 1: 5
Enter Priority for process 1: 3
Enter Arrival Time for process 2: 2
Enter Burst Time for process 2: 7
Enter Priority for process 2: 5
Enter Arrival Time for process 3: 3
Enter Burst Time for process 3: 4
Enter Priority for process 3: 2
Enter Arrival Time for process 4: 5
Enter Burst Time for process 4: 8
Enter Priority for process 4: 1
PID     AT      BT      Priority        CT      WT      TAT     RT
4       5       8       1               13      0       8       0
3       3       4       2               15      8       12      0
1       0       5       3               17      12      17      0
2       2       7       5               24      15      22      15

Average Turnaround Time: 14.75
Average Waiting Time: 8.75

Process returned 0 (0x0)   execution time : 29.736 s
Press any key to continue.
```

4. **Round robin**

```
  #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);
    return 0;
}
```



```
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: 2
Enter Arrival Time for process 2: 3
Enter Burst Time for process 2: 6
Enter Arrival Time for process 3: 3
Enter Burst Time for process 3: 8
Enter Arrival Time for process 4: 1
Enter Burst Time for process 4: 3
Enter Arrival Time for process 5: 2
Enter Burst Time for process 5: 6
PID     AT      BT      CT      WT      TAT     RT
1       0       2       2       0       2       0
2       3       6       21      12      18      12
3       3       8       25      14      22      14
4       1       3       11      7       10      7
5       2       6       19      11      17      11

Average Turnaround Time: 13.80
Average Waiting Time: 8.80

Process returned 0 (0x0)   execution time : 23.035 s
Press any key to continue.
```